INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Business Process Modeling Framework for Integrated Information Systems

RAFAEL DOS SANTOS OLIVEIRA
Setembro de 2025

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Business Process Modeling Framework for Integrated Information Systems

Rafael dos Santos Oliveira

Dissertation

Master's Degree in Computer Engineering

Specialization in Software Engineering

Porto, September 2025

Declaration of Integrity

I declare that I have conducted this academic work with integrity.

I have not plagiarized or applied any form of misuse of information or falsification of results throughout the process that led to its preparation.

Therefore, the work presented in this document is original and of my authorship and has not been used previously for any other purpose.

I also declare that I am fully aware of P. PORTO's Code of Ethical Conduct.

ISEP, Porto, 17 of September 2025

Rafael dos Santos Oliveira

Resumo

À medida que as organizações dependem cada vez mais de sistemas interconectados, a integração eficiente de tecnologias heterogéneas tornou-se um fator determinante para assegurar eficiência, resiliência e coerência estratégica. A presente dissertação propõe uma framework de integração orientada a processos, concebido para mitigar a fragmentação através da conjugação de Business Process Model Notation (BPMN), padrões de integração empresarial e de tecnologias open-source. Sustentado nos princípios de modularidade, abstração e escalabilidade, a framework estabelece uma base arquitetónica unificada que alinha processos organizacionais com infraestruturas técnicas, garantindo simultaneamente interoperabilidade e capacidade de adaptação ao longo do tempo.

A investigação foi conduzida através de uma revisão sistemática da literatura, à qual se seguiu a conceção, implementação e avaliação de um protótipo plenamente funcional. No âmbito desta framework, o BPMN foi utilizado para formalizar e estruturar processos; o Kong e o Keycloak asseguraram a governação e a gestão centralizada de identidades; a Ballerina permitiu o desenvolvimento contract-first e facilitou a integração poliglota; e o Nuclio disponibilizou o ambiente serverless necessário para uma execução escalável. A combinação destas tecnologias demonstrou ser possível centralizar a orquestração numa camada coerente, aplicar políticas de segurança de forma uniforme no perímetro do sistema e conceber integrações com componentes modulares e reutilizáveis. Deste modo, modelos de processos que poderiam permanecer meras representações abstratas foram convertidos em fluxos de trabalho executáveis, capazes de articular tarefas humanas de tomada de decisão com serviços automatizados, assegurando elevado rigor semântico.

A avaliação confirmou que a framework alcança neutralidade em relação a engines, portabilidade de runtime e aplicação consistente da autenticação e da autorização, promovendo práticas de integração sustentáveis. Um caso prático, envolvendo a sincronização dos serviços de finanças e de stock, validou estas capacidades, demonstrando que princípios teóricos podem ser concretizados em operações reais. Em termos globais, os resultados evidenciam a contribuição desta dissertação em duas dimensões: no plano científico, pela articulação entre princípios de arquitetura empresarial e prática operacional; e, no plano da implementação, pela disponibilização de um artefacto funcional que pode ser adaptado e expandido pelas organizações para enfrentar os desafios contínuos da transformação digital.

Palavras-chave: Modelação de Processos de Negócio, Integração de Sistemas, BPMN, Padrões de Integração Empresarial, Frameworks Open-Source, Automação de Workflows, Arquitetura de Sistemas de Informação.

Abstract

With organizations increasingly relying on interconnected systems, the ability to integrate heterogeneous technologies seamlessly has become a critical determinant of efficiency, resilience, and strategic coherence. This dissertation presents a process-centric integration framework designed to mitigate fragmentation by combining Business Process Model Notation (BPMN), enterprise integration patterns, and open-source technologies. Grounded in the principles of modularity, abstraction, and scalability, the framework provides an unified architectural foundation that aligns organizational processes with technical infrastructures while preserving interoperability and adaptability over time.

This research was conducted upon a systematic review of the literature, followed by the design, implementation, and evaluation of a fully functional prototype. Within this framework, BPMN was adopted to formalise and structure processes; Kong and Keycloak ensured that governance and identity management were handled centrally; Ballerina enabled contract-first development and facilitated polyglot integration; and Nuclio provided the serverless runtime necessary for elastic execution. Together, these technologies prove that orchestration can be centralized in a single coherent layer; security policies enforced uniformly at the system perimeter; and integrations designed as modular, reusable components. As a result, process models that might otherwise remain abstract representations were effectively transformed into executable workflows, capable of bringing together human decision-making tasks and automated services with a high degree of semantic accuracy.

Evaluation confirmed that the framework realises engine neutrality, runtime portability, and consistent enforcement of authentication and authorisation, while fostering sustainable integration practices. A practical scenario involving the synchronisation of finance and stock services validated these capabilities, showing that theoretical principles could be enacted in real operations. Taken together, the findings highlights the dissertation´s contribution on two levels: from a research prespective, by combining enterprise architecture principles with operational practice; and from an implementation standpoint, by providing a functional artifact that organizations can adapt and extend to address the evolving challenges of digital transformation.

Keywords: Business Process Modelling, System Integration, BPMN, Enterprise Integration Patterns, Open-Source Frameworks, Workflow Automation, Information Systems Architecture.

Acknowledgment

This section is dedicated to expressing my heartfelt gratitude to all those who have played a meaningful role in the successful completion of my dissertation.

First of all, I would like to thank my family for always supporting and encouraging me throughout this journey. Their presence and motivation were fundamental in giving me the strength to keep moving forward.

A heartfelt thank you also goes to all the teachers who helped me grow and provided me with the skills necessary to complete this stage. To all my friends, both lifelong and those I met at university, I am deeply grateful. This journey would certainly have been less colourful and joyful without your presence.

I would also like to give special thanks to my colleagues at ISEP who accompanied me and grew together with me along this path: Rogério Sousa, Oscar Folha, Rafael Faísca, and Nuno Marmeleiro.

Finally, I would like to thank my supervisor, Paulo Maio, for his invaluable guidance throughout this process and the dissertation. His insights and encouragement were essential in bringing this work to completion.

Thank you for everything!

Table of contents

List of Figures

List of Tables

Acronyms and Symbols

Acronyms List

API	Application Programming Interface
JSON	JavaScript Object Notation
URL	Uniform Resource Locator
API	Application Programming Interface
BPMN	Business Process Model and Notation
BPMS	Business Process Management Systems
CRM	Customer Relationship Management
DMN	Decision Model and Notation
EIPs	Enterprise Integration Patterns
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
GDPR	General Data Protection Regulation
IEEE	Institute of Electrical and Electronics Engineers
JSON	JavaScript Object Notation
OMG	Object Management Group
REST	Representational State Transfer
URL	Uniform Resource Locator

1 Introduction

In today's dynamic organizational landscape, the integration of diverse information systems has become a fundamental element in achieving operational efficiency and strategic competitiveness. Organizations increasingly rely on interconnected systems to facilitate seamless communication and business process alignment. However, the quick evolution of technology has highlighted significant challenges such as system fragmentation, communication inefficiencies, and a lack of transparency in (business and technical) processes ecosystems. These challenges limit organizations' capacity to respond effectively to evolving market conditions and fully utilize advancements in technology.

This dissertation proposes an innovative framework for information systems development and systems interoperability. The just described challenges are addressed two fold: (i) focusing on the business process through a native and early adoption of Business Process Model and Notation (BPMN) and (ii) putting an emphasizes on interoperability, scalability, and adaptability by integrating enterprise integration patterns and other best practices. Yet, the proposed framework is leveraged by multiple open-source technologies. By doing this, the suggested framework aims to provide companies with the resources they need to improve decision-making, streamline processes, and align their technology infrastructures with their corporate goals. The study emphasizes the necessity of adaptable integration and development strategies that go beyond conventional techniques, such as enterprise service buses. Schmitz and Wimmer in 2023 highlight the critical necessity of adaptable frameworks that can effectively respond to shifting business environments [1]. In parallel, Nuutinen addressed the essential role of enterprise architecture in harmonizing IT systems with organizational objectives, thereby promoting enhanced integration and strategic alignment [2]. By building on these insights, This work aims to rethink integration paradigms, offering a strong basis for contemporary information systems.

1.1 Problem Description

The contemporary organizational environment is marked by dependence on a diverse range of software applications tailored to specific departmental or operational needs. These applications, while effective in isolation, often contribute to duplicated data, inconsistent information, and a lack of transparency in business processes. The fragmentation of workflows between systems that were not designed to communicate effectively further complicates this situation. As businesses evolve, their need for integration grows more urgent, as the absence of unified systems can hinder decision-making and overall performance. Research underscores the importance of data governance in mitigating these issues, as highlighted in [3] by Cheong and Chang , who identified data consistency and synchronization as key drivers of operational success.

Existing technical solutions, such as enterprise service buses (ESBs) [4], microservices [5], and event-driven architectures [6], have addressed some integration aspects but often fail to deliver a holistic approach and lack the adaptability required to accommodate evolving business processes, which hinders scalability and increases maintenance efforts, as noted by Reinprecht et al. in [7]. In 2016, Hinkelmann [8] reinforces the importance of continuous alignment between business and IT systems, emphasizing the role of enterprise ontologies in achieving transparency and adaptability. These solutions are frequently rigid and lack the flexibility to adapt to rapidly changing organizational demands. Moreover, they tend to focus on technical interoperability, neglecting the broader context of business process alignment and transparency. For instance, while frameworks such as BPMN (Business Process Model and Notation*)* emphasize workflow management, they often overlook the critical interplay between technical interoperability and user interfaces [9]. Further, studies on middleware and interoperability, such as those by van der Aalst et al. , point to the need for frameworks that accommodate evolving business demands while maintaining simplicity and scalability [10]. The absence of unified standards for monitoring and managing workflows limits organizations' ability to identify inefficiencies and respond to changing demands effectively. Kohansal in 2024 identifies transparency as a critical enabler of operational efficiency in modern enterprises [11]. The proposed framework aims to bridge this gap, enabling a seamless connection between technical capabilities and organizational objectives. By emphasizing the role of business process modeling and leveraging advances in middleware and data governance, the framework seeks to redefine how systems interact and support operational needs.

1.2 Objectives

The objectives of this dissertation are both strategic and technical, conceived to address the persistent challenges of integrating heterogeneous information systems. Rather than being defined narrowly in terms of concrete technologies, they are rooted in broader principles of enterprise architecture, business process modeling, and integration best practices. As highlighted [12], organizational resilience and competitiveness increasingly depend on the ability to coordinate complex systems while avoiding duplication and fragmentation. Likewise, adherence to established standards such as ISO/IEC 19510 [10], which formalizes the specification of Business Process Model and Notation (BPMN) [11], ensures that integration practices remain transparent, interoperable, and verifiable across contexts. The following objectives articulate these guiding concerns in structured form.

O1. To design and validate a modular and process-centric integration framework capable of mitigating fragmentation in information systems. The focus here is on ensuring that orchestration mechanisms are scalable, transparent, and consistently aligned with organizational objectives, thereby responding to the structural deficiencies of traditional ad hoc integration strategies.

O2. To ensure formalization and transparency in process modeling and orchestration, creating a common ground between technical and business stakeholders. By grounding integration in

standardized modeling practices such as those defined in ISO/IEC 19510 [13], processes can be specified in a way that is not only technically executable but also organizationally intelligible, reducing ambiguity and improving traceability.

O3. To establish mechanisms for flexible and precise data exchange that prevent semantic drift and redundancy across heterogeneous systems. The reliability of decision-making processes depends on maintaining data integrity and contextual meaning across domains, echoing the principles of enterprise integration patterns [9] that emphasize consistency and message semantics as central to robust interoperability.

O4. To promote sustainability and independence through open, configurable, and non-proprietary approaches. Proprietary lock-in remains a critical obstacle to long-term adaptability in enterprise systems. By privileging configurable over hard-coded integration strategies, the framework aims to minimize technological dependency and operational risk, thus following the orientation toward cost-effectiveness and adaptability, emphasized by Chorafas, in the enterprise integration literature [14].

O5. To define a comprehensive methodology that bridges enterprise architecture principles with integration practices, aligning IT systems with strategic organizational goals. This objective reflects the broader ambition of embedding integration into the governance layer, where resilience and innovation capacity are cultivated through systematic and transparent architectural choices. In this respect, the framework aspires to reinforce the dual imperative of interoperability and strategic alignment, as recommended by advanced standards and best practices in enterprise architecture.

Together, these objectives not only clarify the ambitions of the proposed framework but also provide a clear bridge toward the research questions presented in (cf. Section 1.3). In this way, the conceptual design of the objectives facilitates their conversion into evaluable research questions, ensuring that the dissertation can both demonstrate methodological rigor and empirically verify the framework's capacity to address the integration challenges identified in the literature.

1.3 Research Questions

The general and broader context and its assumptions in which this work is being carried out might be resumed in the following research question:

> **RQ0:** How can a modular, business process-centric framework address major software development life cycle concerns for new information systems as well as its construction, integration and interoperability with the ones existing within an organization?

However, it is worth noticing that this work while preserving the previous stated key characteristics of the target framework (i.e., being modular and business process-centric), it is

constrained to the integration and interoperability concern[1] only. Thus, the research questions considered for this study are:

> **RQ1:** What key characteristics and components should a business process-centric framework have to be usable in the wide range of possible information systems architectures?
>
> **RQ2:** How can enterprise integration patterns and open-source technologies contribute to the conception and construction of a business process-centric framework that promotes seamless communication and adaptability in complex system environments?

These two research questions are closely interconnected and complementary. RQ1 focuses on defining the essential principles and components that make a business process-centric framework adaptable across various architectural scenarios. It establishes the foundational design criteria needed for such a framework to be both modular and widely applicable. Building on this foundation, RQ2 explores the practical realization of these concepts by leveraging enterprise integration patterns and open-source technologies. Together, these questions aim to bridge the gap between theoretical design and real-world implementation, ensuring the resulting framework is both robust and capable of fostering seamless interoperability in complex enterprise environments.

1.4 Methodology

The methodology followed in this dissertation was designed to ensure both academic rigor and practical applicability. It combines systematic inquiry with an experimental validation strategy, thereby bridging theoretical research with the design and evaluation of a concrete framework for integrated information systems. The methodological pathway can be articulated in four interconnected stages.

The first stage comprised a systematic literature review, conducted according to the PRISMA guidelines. This process ensured the identification, screening, and selection of high-quality peer-reviewed studies directly relevant to system integration, business process modeling, and enterprise interoperability. Databases such as IEEE Xplore, ACM Digital Library, and Google Scholar were employed to retrieve publications, while inclusion and exclusion criteria guaranteed the reliability and relevance of the selected works. This step provided a solid

[1] The other concerns are being addressed on another master dissertation work running in parallel with this work. Despite that, both works are carried out independently of each other.

theoretical foundation for defining the research questions and framing the scope of the proposed solution.

Building upon this foundation, the second stage focused on the state of the art**.** Here, specific technologies and standards most pertinent to the construction of the framework were analyzed in depth. Emphasis was placed on Business Process Model and Notation (BPMN) as a standard for process modeling, on enterprise integration patterns as a means to operationalize interoperability, and on open-source tools such as Camunda. This examination allowed the identification of both the opportunities and the limitations of existing approaches, thereby situating the contribution of this dissertation within a broader technological and scientific landscape.

The third stage involved the design and development phase. In this stage, the conceptual framework was instantiated through the definition of its architecture, modular components, and integration logic. Special attention was devoted to ensuring modularity, scalability, and transparency. A constructive research methodology was used throughout the development process, continuously translating theoretical ideas into tangible architectural artifacts and implementation prototypes. The iterative nature of this phase allowed the framework to be gradually improved in response to both theoretical revelations and real-world limitations.

The evaluation phase, which came after development, was the focus of the fourth stage. Its goal was to verify the framework in relation to the previously established goals and research questions. The effectiveness of the suggested architecture in terms of interoperability, scalability, and process transparency was evaluated using controlled experiments and real-world situations. To ensure comparability and methodological soundness, evaluation metrics were taken from the literature as well as industry standards.

Finally, the outcomes of these stages converged in the result analysis and discussion. This step entailed a critical reflection on the experimental findings, juxtaposed with the theoretical expectations established in the literature review. The analysis considered both strengths and limitations, offering an informed discussion of how the proposed framework addresses the challenges of modern system integration and where further refinements might be necessary. Complementary to these stages, Appendix A (Skills Management) and Appendix B (Project Planning) provide additional context regarding the competencies developed and the structured planning process that supported the research execution.

1.5 Contributes

Implementing the proposed framework offers numerous benefits. By facilitating smooth communication between heterogeneous systems, BPMN and enterprise integration patterns improve interoperability, decrease fragmentation, and boost workflow efficiency. The modular and open-source nature of the framework ensures improved scalability, enabling it to adapt to organizational growth and evolving technological landscapes while minimizing the need for extensive re-engineering efforts. The use of standardized modeling and monitoring tools enhances operational transparency, allowing organizations to identify bottlenecks, optimize processes, and respond proactively to changes in their operating environments. By prioritizing open-source technologies and reducing reliance on proprietary solutions, the framework minimizes integration costs while maintaining high performance and reliability, as required and highlighted by Frantz et al. in 2016 [15].

Leveraging BPMN and Camunda, the framework automates workflows and enhances data accessibility, ensuring more accurate and informed decision-making. This technical advancement supports the objective of adopting cutting-edge technologies for seamless data integration and process automation. BPMN's structured models facilitate clear communication between stakeholders, while Camunda's execution engine ensures that processes are not only designed but also deployed and monitored with reliability. Together, these capabilities enable the robust execution and adaptability outlined in the research goals.

Two of the framework's primary operational benefits are its real-time monitoring and process transparency. Standardized modeling tools and real-time analytics help organizations identify bottlenecks, dynamically optimize processes, and adapt to shifting market demands. This transparency drives proactive management, which enables companies to respond rapidly to opportunities and operational difficulties.

The framework strategically increases organizational agility by creating adaptable systems that respond swiftly to changes in the market and unforeseen challenges. By aligning IT systems with overarching business objectives, it ensures that technology acts as an enabler of long-term strategic objectives rather than an operational bottleneck. This alignment fortifies organizational resilience, fostering innovation and allowing enterprises to leverage emerging opportunities effectively. These technological capabilities with business objectives enhance decision-making, foster innovation, and improve organizational agility, as demonstrated by Bellman and Griesi in 2015 [16]. Furthermore, the framework incorporates enterprise integration best practices, creating workflows that are not only resilient but also flexible enough to accommodate future developments in technology and changing business environments.

Ultimately, this strategy gives businesses the resources they need to attain operational excellence. Through process simplification, improved teamwork, and actionable insights, it enables companies to make wise choices and preserve a long-term competitive edge. By addressing both immediate operational needs and long-term strategic objectives, this holistic approach helps organizations succeed in intricately linked environments.

1.6 Ethical Considerations

Ethical considerations are fundamental in research involving business process modeling and integrated information systems, shaping the integrity and societal impact of the work. This study was conducted in alignment with established ethical standards, ensuring transparency, fairness, and adherence to professional codes of conduct.

The research strictly complied with the ethical guidelines outlined by the *Code of Conduct of P. Porto* [17] and the *Declaration of Integrity* [18], as well as broader frameworks such as the IEEE Code of Ethics [19]. These principles guided every stage of the study, fostering accountability and ethical rigor.

Given that this study builds upon prior academic work, explicit permissions were obtained from all collaborators and supervising faculty members associated with the original project. This ensures proper acknowledgment of their contributions, while respecting intellectual property and promoting collaborative integrity.

Data security and privacy were prioritized throughout the entire research process. All data were anonymized and handled in full compliance with regulations such as the GDPR in order to safeguard sensitive information and prevent misuse. Additionally, the study only used open-source and publicly available tools and datasets. The academic and professional communities' values of openness and transparency are reinforced by this approach, which promotes inclusivity, accessibility, and teamwork.

By following these moral guidelines, the study not only satisfies professional and institutional requirements but also shows that it is dedicated to responsible, fair, and influential scholarship in the field of integrated information systems.

1.7 Document Structure

The dissertation is divided into nine chapters, each of which aims to develop the suggested framework's conceptual foundations, methodological approach, architectural design, and practical validation in turn. This progression ensures coherence between theoretical grounding and empirical demonstration.

- **Introduction**: Establishes the research context and problem description, formulates objectives and research questions, and outlines the adopted methodology. It also discusses the main contributions of the study and addresses ethical considerations.
- **Background Knowledge**: Presents the fundamental concepts of information systems, the challenges of fragmentation, and the role of standards such as BPMN and Enterprise Integration Patterns. This chapter establishes the theoretical underpinnings required for integration research.
- **Systematic Literature Review**: Provides a critical survey of existing academic work, identifying current methodologies, frameworks, and their limitations. The analysis

directly informs the design of the proposed framework and addresses the defined research questions.

- **Framework Proposal**: Details the requirements, architectural rationale, and technological components of the integration framework. This includes the definition of use cases, design principles, and the methodological alignment between BPMN workflows, integration patterns, and governance structures.
- **Architectural Design**: Uses the C4 model to decompose the framework into multiple levels of abstraction (context, container, component, and code). Each level illustrates responsibilities, interactions, and boundaries, supported by UML diagrams and architectural patterns.
- **Practical Scenario and Implementation**: Demonstrates the framework in operation by integrating heterogeneous services in a real-world scenario. The scenario validates the framework's applicability by highlighting issues with synchronization, workflow orchestration, human participation, and authorization policies.
- **Results and Evaluation**: Analyzes the outcomes of the practical scenario and assesses the framework against its objectives. The evaluation addresses robustness, scalability, adaptability, and compliance with enterprise governance.
- **Discussion**: Interprets the results in light of the research questions and theoretical background, drawing connections to existing literature and highlighting the framework’s contributions to both academic and practical domains.
- **Conclusions**: Summarizes the findings of the dissertation, outlines limitations, and proposes directions for future research.

2 Background Knowledge

This chapter establishes the conceptual foundations that underpin the development of the proposed framework. It begins by examining the evolution and current challenges of Information Systems (cf. Section 2.1), with particular emphasis on the problem of fragmentation that motivates the pursuit of integration. The discussion then turns to Business Process Model and Notation (BPMN) (cf. Section 2.2), which is presented as a standardized language for representing and governing processes in a way that is intelligible to both technical and non-technical stakeholders. Subsequently, the role of Enterprise Integration Patterns (EIPs) is analyzed (cf. Section 2.3), highlighting how these reusable design constructs operationalize BPMN models and enable the orchestration of heterogeneous systems. The chapter concludes with a synthesis (cf. Section 2.4) that reflects on the interplay between Information Systems, BPMN, and EIPs, thereby preparing the ground for the architectural proposal and implementation discussed in subsequent chapters.

2.1 Information Systems

Information systems (IS) have always been crucial to organizations, influencing how they gather, process, and manage data to enhance decision-making. In the beginning stages of computing, these systems were simple, focusing on data storage and basic reporting tasks. As highlighted by Pawlzak, Laudon and hohpe, information systems have evolved over time into comprehensive tools capable of integrating and automating complex business processes [20].

Organizations rely on a layered approach to information systems (IS) to support their diverse needs, from day-to-day operations to long-term strategic planning. At the ground level, transactional systems keep the wheels turning, managing essential tasks like processing orders, tracking inventory, and handling payroll. These systems ensure the smooth functioning of routine operations. Widianto and Subriadi, highlight the critical role of transactional systems in facilitating real-time data processing and providing the foundation for decision-making at higher levels. Their research highlights the need for operational-level IS to successfully coordinate with the functions and duties of the different management levels within the company [21]. Decision support systems (DSS) and management information systems (MIS) become more important as we advance to middle management. These tools enable managers to make well-informed decisions by converting unstructured data into thorough analyses and reports. In order to guarantee smooth data flow and boost managerial effectiveness in decision-making, Liu et al. address the significance of integrating DSS with other IS layers[22]. In order to meet the changing demands of middle management, their suggested Integrated Decision Support Environment (IDSE) provides a multi-tier, reconfigurable framework that promotes cooperation and information exchange. At the top of the hierarchy, executive support systems (ESS) come into play, distilling vast amounts of complex information into clear, actionable

insights for senior leaders. Liu et al. provide examples of how ESS integration with web-based technologies helps businesses better manage intricate processes by bridging the gap between the strategic and operational layers [23]. Their study emphasizes how important it is to create open software platforms that facilitate individualized decision-making and allow for uniform information sharing amongst organizational levels. However, in spite of their significance, these systems frequently function independently, forming silos that obstruct cooperation and communication. This fragmentation draws attention to an increasing challenge—and an opportunity—for organizations to implement integrated solutions that streamline decision-making at all levels and unify their information systems. According to Widianto and Subriadi, assessing the efficacy and efficiency of IS across management layers yields important information for improving these systems and making sure they provide quantifiable business value in line with organizational objectives [21].

Organizations started putting in place specialized systems made to meet the particular requirements of various departments as they grew and changed. The once cohesive and unified organizational structure gradually became fragmented as each department adopted its own customized solutions, creating a new problem known as fragmentation and causing communication gaps and inefficiencies. Take, for example, a business where the sales team uses a different platform to handle orders, the marketing team tracks leads using a CRM system, and the finance department uses spreadsheets to reconcile payments. Although each system performs well within its own domain, they are unable to effectively communicate with one another. Duplication, inconsistencies, and inefficiencies result from the frequent need to enter data more than once. Because managers are compelled to rely on conflicting or insufficient information, this lack of integration impairs both operational efficiency and strategic decision-making [24]. The demand for a cohesive approach to Information Systems integration has become more pressing than ever.

A concrete illustration of this challenge is further developed in the Practical Scenario described in Chapter 6 (cf. Section 6), where heterogeneous systems such as finance, stock, and logistics are examined in detail. There, the fragmentation of information flows across these systems is shown to hinder operational transparency and efficiency, providing a real-world backdrop against which the proposed framework is validated.

2.1.1 Addressing the Challenge of Fragmentation in Information Systems

The complexity and diversity of contemporary organizations are reflected in the fragmentation of IS landscapes, which is more than just a technical problem. As mentioned by Valacich & Schneider in 2009, Over the years, as businesses expanded and adopted new technologies, they often prioritized immediate solutions over long-term integration strategies. For instance, a growing company might invest in a sophisticated ERP system to streamline its supply chain but later find that this system does not align with its legacy accounting software [25]. Similarly, a marketing department might implement a cutting-edge analytics tool only to discover that it cannot pull data directly from the company's operational databases [26].

The consequences of such fragmentation are far-reaching. Duplication of data across systems creates inconsistencies, making it difficult to generate reliable reports. Employees waste valuable time manually reconciling information, while IT teams struggle to maintain a patchwork of custom integrations. Most importantly, as mentioned by Pitt in 1995, this fragmentation obscures the larger picture, preventing organizations from achieving the transparency and agility they need to thrive in competitive markets [27].

But what is the root cause of this problem. It lies in the way organizations approach their systems development. In many cases, systems are built to solve specific problems without considering how they will interact with other applications. As mentioned by Hohpe & Woolf in 2012, integrations, if they exist, are often hard-coded and rigid, making it challenging to adapt as business needs change [9]. This is further complicated by the rapid pace of technological innovation, which constantly introduces new platforms, tools, and standards [28].

2.1.2 The Complexity of Integration: A Multi-Dimensional Challenge

Addressing the problem of IS integration requires more than just technical expertise; it demands a holistic understanding of organizational dynamics, technological constraints, and stakeholder needs. According to Bidgoli in 2003 The technical complexity of integrating different systems is a challenge in itself. Some may use incompatible data models, communication protocols, or architectures, requiring middleware or custom-built solutions to bridge the gap [29]. However, even the best technical solutions can fail if they do not align with organizational goals.

Consider the role of stakeholders in this process. IT teams are often focused on ensuring system reliability and security, while business managers are primarily concerned with improving efficiency and reducing costs [3]. As stated by DeLone & McLean in 1992, End-users, on the other hand, care about usability and the speed with which they can complete their tasks. These differing priorities can lead to misalignment, where solutions that satisfy one group inadvertently create challenges for another [30].

For instance, an IT department might implement a highly secure system that requires multi-factor authentication for every transaction. While this enhances security, it can frustrate employees who are used to faster workflows, reducing overall productivity. Similarly, a business manager might push for a low-cost integration solution that fails to meet the technical requirements of the IT team, resulting in a system that is unstable or difficult to maintain [31].

Poor integration not only affects internal operations but also has strategic implications. Inconsistent data can lead to flawed analyses, which in turn result in poor decision-making. As emphasized by Avison & Fitzgerald in 1988, fragmented systems also make it harder to respond to external pressures, such as new regulatory requirements or shifts in market demand. Organizations that cannot adapt quickly risk losing their competitive edge [32].

2.1.3 Rethinking Integration: Toward a Unified Framework

To navigate these challenges, organizations need to rethink how they approach IS integration. Rather than treating it as an afterthought or a purely technical exercise, integration should be seen as a strategic priority that requires careful planning, stakeholder collaboration, and a commitment to flexibility.

Imagine the perfect situation where all of an organization's systems work together harmoniously. Data moves between apps with ease, removing duplication and guaranteeing consistency. Business processes are transparent and well-documented, allowing managers to quickly identify bottlenecks and inefficiencies. IT teams can easily update or replace individual components without disrupting the entire ecosystem. Most importantly, employees across the organization feel empowered to use the tools at their disposal, confident that they are working with accurate, up-to-date information [33].

Achieving this vision requires a framework that addresses both technical and organizational challenges. Such a framework should be modular, allowing organizations to adapt to changing needs without overhauling their entire infrastructure as stated previously by Hohpe & Woolf in 2012 [9]. It should prioritize transparency, ensuring that all stakeholders—whether they are IT professionals, business leaders, or end-users—have a clear understanding of how systems interact and support organizational goals [26]. Finally, it should leverage existing technologies and standards where possible, reducing costs and accelerating implementation, As suggested by Stair & Reynolds in 2001 [28].

The development of this framework is not without its challenges. It must reconcile the diverse needs of stakeholders, balance short-term demands with long-term goals, and remain flexible enough to accommodate future innovations. However, by taking a systematic and collaborative approach, according to Valacich & Schneider, organizations can build a foundation for sustainable growth and success [25].

2.2 Business Process Model and Notation (BPMN)

In modern organizations, representing and managing business processes effectively is crucial for operational clarity and efficiency. Business Process Model and Notation (BPMN), a standard developed and maintained by the *Object Management Group (OMG)[34]*, offers a graphical approach to business process modeling. This standard provides a formalized framework to depict workflows, enabling clear communication and analysis of processes across various stakeholders. The following content introduces BPMN by outlining its structure, key components, and its role in the context of business process management.

2.2.1 The Power of Visualization: Uniting Complexity and Clarity

Imagine a bustling organization where managers struggle to communicate intricate processes to their teams. Departments operate in silos, and inefficiencies pile up due to fragmented workflows. Enter BPMN, a standard designed to restructure such confusion into clarity. BPMN bridges the gap between technical and non-technical stakeholders by providing a graphical representation of business processes. As stated by White and Miers, It enables teams to align on workflows, ensuring that everyone understands the sequence of activities, decisions, and interactions required to achieve organizational goals [35].

BPMN's visual models offer more than just aesthetic appeal. They provide an intuitive roadmap for process improvement. As Silver aptly describes, these diagrams serve as the foundation for fostering operational efficiency and streamlining communication across teams [36]. They reveal bottlenecks and inefficiencies, helping organizations optimize their operations while fostering a culture of continuous improvement.

2.2.2 Core Components of BPMN

To comprehend the BPMN (Business Process Model and Notation) concept, envision it as an elaborate toolkit designed for process mapping. Each component within this toolkit serves a distinct purpose. Collectively, these components construct a detailed and comprehensive map of a process, illustrating the path for every decision made and meticulously documenting each step involved in the process. Figure 1 shows a purchase order process, providing a clear illustration of these concepts in action.

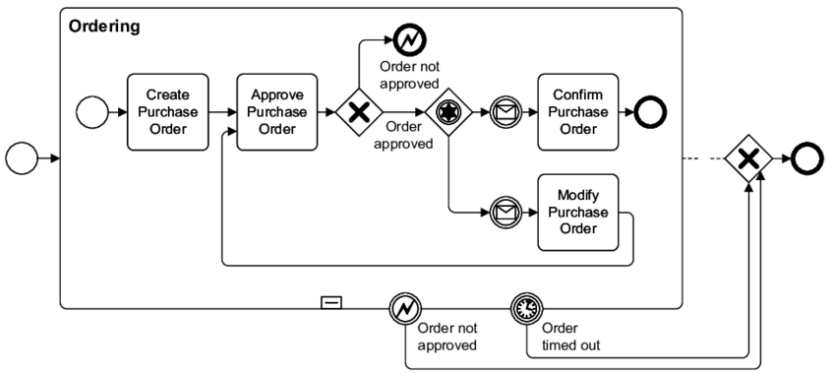


Figure 1 BPMN Order Process Example [37]

At the core of BPMN are flow objects, which drive the process forward. These include events, which signify critical moments such as the start, pause, or end of a process; activities, which represent tasks or actions that move the process along; and gateways, which serve as decision points, directing the flow based on specific conditions. For instance, in the purchase order process example, an event initiates the process when a purchase order is created. A gateway follows to decide whether the order is approved or requires modification, illustrating the decision-making element of BPMN.

Connecting objects link these flow elements together, ensuring a logical and seamless progression. Sequence flows dictate the order in which tasks occur, as seen in the sequence of "Create Purchase Order" leading to "Approve Purchase Order." Message flows record information exchanges, such as departmental communication during order confirmation. Associations further enhance understanding by linking tasks to relevant details, such as annotations explaining why an order may not be approved.

To enhance organization and role clarity within a process, BPMN allows the use of swimlanes, which visually divide tasks among participants or systems. Pools represent broader entities, such as entire organizations, while lanes within those pools delineate the responsibilities of specific roles or teams. In the purchase order example, swimlanes might separate tasks performed by the procurement team from those handled by management. This clear division aids in tracking responsibilities and improving process transparency, as emphasized by Allweyer [38].

Artifacts provide additional layers of detail, enriching BPMN diagrams by including data objects and annotations. Data objects represent the information utilized or generated throughout the process, such as transaction records or customer profiles. Annotations act as explanatory notes, offering clarity or justifications for certain steps. These elements make diagrams more accessible and informative, a point highlighted by White and Miers in their discussion on BPMN's utility as a modeling language [35].

By integrating these elements illustrated in Figure 2, BPMN diagrams construct a detailed yet accessible framework that transforms abstract workflows into clear, actionable steps. This structured methodology proves invaluable across various domains, particularly in complex projects and organizational processes, where ensuring clarity and precision is critical for successful execution and collaboration.

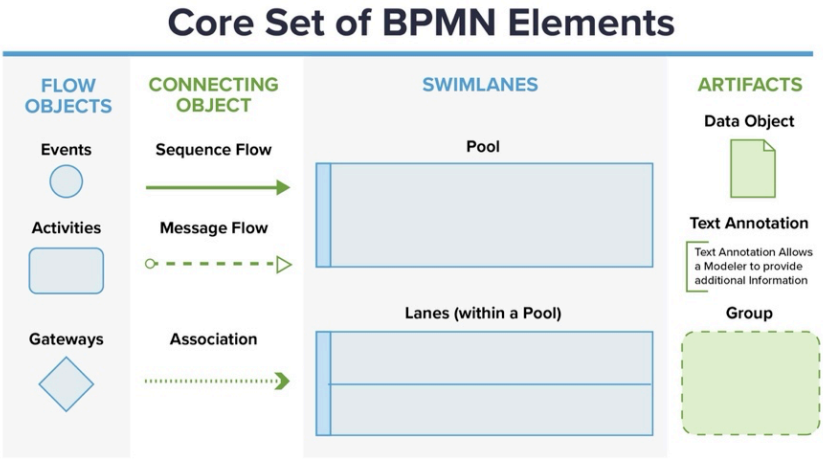


Figure 2 Core Set of BPMN Elements [39]

2.2.3 From Diagrams to Action: The Role of BPMN Engines

A meticulously crafted BPMN diagram serves as the foundation, but transforming these static blueprints into executable processes requires a combination of technical implementation and strategic planning. Organizations turn to BPMN engines such as Camunda [40] , BonitaSoft BPM [41], and Flowable [42]—powerful tools that interpret BPMN diagrams and operationalize them into dynamic, automated workflows. These engines form the foundation of process automation, carrying out the models through task management, progress tracking, and smooth integration with third-party systems like ERP and CRM platforms [43]. They ensure that the procedures shown in diagrams are successfully translated into actual operations by bridging the gap between theory and practice.

A number of organizational and technical factors are involved in achieving this transition. Technically speaking, it is crucial to make sure that the BPMN diagram is enhanced with executable details like service tasks, decision rules, and data mappings in addition to being syntactically correct. These components are necessary for BPMN engines to automate decision-making, initiate workflows, and establish connections with databases or APIs. The ability of the system to handle business cases or instances—distinct instances of a process—is another factor that affects operational flexibility. Due to the dynamic orchestration of these instances, workflows can be executed in response to real-time events and business requirements.

Addressing the rigidity of hard-coded workflows—which are used to define processes in many traditional applications—is another crucial component. Although hard-coded flows are effective for fixed operations, they lack scalability and flexibility. BPMN engines get around this limitation by offering programmable mechanisms that simulate a generic engine's adaptability. These mechanisms enable organizations to adjust workflows, reconfigure tasks, and incorporate changes without extensive programming, thereby fostering greater responsiveness to evolving business demands.

Operationalizing BPMN also enhances accountability and consistency. By embedding automation, organizations reduce human error, ensure data integrity, and streamline operations. Moreover, the integration of BPMN engines with enterprise systems unifies departmental workflows, eliminates redundancies, and provides a holistic view of business processes [44]. This seamless synergy allows businesses to stay agile in the face of changing market dynamics while maintaining efficiency and control. By transforming "beautifully crafted diagrams" into "executable models," organizations unlock the full potential of process automation, paving the way for innovation and continuous improvement.

2.2.4 Challenges in Using BPMN

Although BPMN (Business Process Model and Notation) is widely recognized as a robust and valuable tool for process modeling, organizations need to be mindful of the specific challenges associated with its implementation before committing to its extensive adoption. These

challenges can range from the complexity of the notation itself to the potential need for additional training and resources, which may impact the overall effectiveness of the transition.

2.2.4.1 Complexity and Learning Curve

Because of its comprehensiveness and flexibility, BPMN has a steep learning curve, particularly for beginners. The notation includes a broad array of symbols and elements, each with its own specific semantics, which can be overwhelming for modelers without prior experience. As Leopold et al. point out, the complexity of BPMN often results in challenges for practitioners, particularly when choosing between multiple representational options for the same semantics [45]. This emphasizes how effective notation mastery requires systematic instruction and consistent practice.

According to Recker, the growing demand for education and training has been fueled by the growing popularity of BPMN, reflecting the challenges users encounter in mastering the system [46]. While this complexity can initially deter adoption, organizations that invest in training programs often find that the long-term benefits, such as improved communication and operational clarity, outweigh the initial learning hurdles.

2.2.4.2 Balancing Detail and Simplicity

A significant strength of BPMN is its ability to model intricate processes in detail. But when diagrams are too detailed, they can also become a liability, creating cluttered representations that are hard for stakeholders to understand. According to Pufahl, this difficulty is especially noticeable in intricate fields like healthcare, where it is crucial to record intricate procedures but runs the risk of overcrowding the diagram with data [47].

To address this, Pufahl suggests using best practices to address this, like employing standardized process fragments to guarantee clarity while preserving adequate detail [47]. The key is striking a balance that allows the diagrams to convey necessary insights without overwhelming the audience, ensuring accessibility and comprehensibility.

2.2.4.3 Ensuring Consistency Across Diagrams

In large organizations, inconsistencies in the interpretation and implementation of BPMN across different teams or departments can lead to confusion and inefficiency. Recker in 2010 highlighted how variability in BPMN usage reflects a broader issue of over-engineering, where the abundance of features can result in divergent modeling practices [46].

To mitigate this, Leopold et al. in 2016 recommend establishing comprehensive guidelines and standards for BPMN usage. By enforcing uniform modeling practices, organizations can ensure consistency across departments, enhance collaboration, and reduce the risk of misinterpretation. This approach is particularly vital in industries where processes span multiple teams or geographic locations [45].

2.2.4.4 Domain-Specific Challenges

In specialized sectors like healthcare, additional challenges arise from the complexity of domain-specific processes. Pufahl in 2022 identified some difficulties in representing the intricate workflows inherent to healthcare and suggest that the modest adoption of BPMN in such fields may stem from these complexities. Their study reveals that providing tailored solutions, such as domain-specific process fragments, can help modelers capture nuanced aspects of healthcare processes while maintaining alignment with BPMN standards [47].

2.3 BPMN in the Context of Enterprise Integration

This section discusses the pivotal role of Enterprise Integration Patterns (EIPs) in managing system communication and ensuring the efficient execution of business processes, particularly when applied in the context of BPMN.

2.3.1 From Models to Action: Why Patterns Matter

Enterprise Integration Patterns (EIPs) provide essential mechanisms for managing the complexities of system communication and interaction within modern organizations. Hohpe and Woolf describe EIPs as reusable solutions for recurring challenges in system integration, offering a standardized way to streamline data exchange and workflow execution across diverse technological landscapes [48].

The increasing need for such patterns stems from the growing complexity of enterprise systems and the demand for flexibility in connecting them. Aier and Winter argue that integration services, which rely on EIPs, form a "middle layer" that bridges diverse artifacts, processes, and systems. These services ensure adaptability and alignment by offering modular fragments that can be tailored to specific project requirements. By doing so, they provide a robust framework for enterprise-wide integration [49]. The Figure 3 highlights some of the essential patterns.

Visual process modeling through BPMN is an essential step in understanding and communicating workflows. However, the true value of BPMN is realized only when its diagrams are operationalized. Curl and Fertalj emphasize that transforming BPMN models into executable workflows requires addressing the inherent challenges of integrating heterogeneous systems, applications, and platforms. EIPs enable this transition by establishing standardized protocols for communication and data transformation [50].

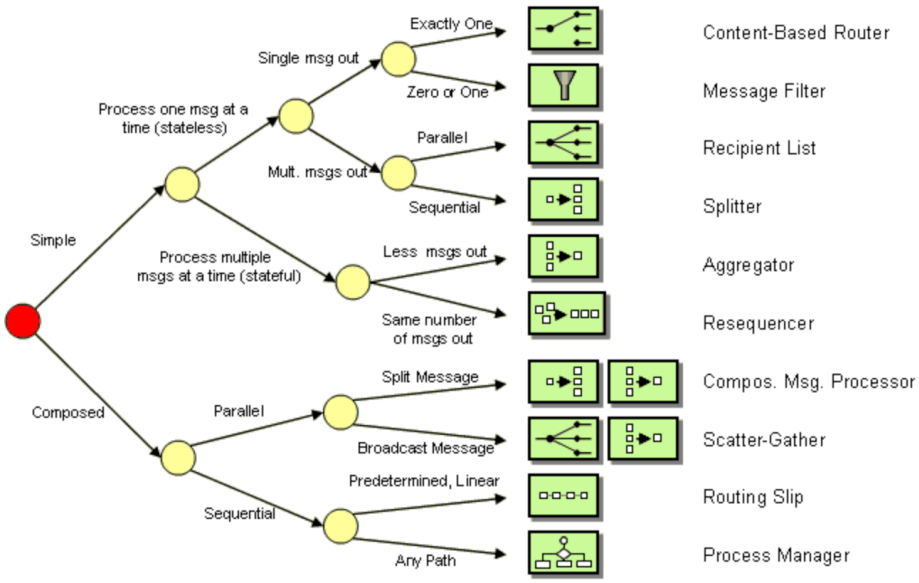


Figure 3 Enterprise Integration Patterns [51]

Ritter highlights the role of EIPs in formalizing interactions between systems, ensuring consistency in data handling and process execution. This structured approach supports the operationalization of BPMN workflows, turning abstract designs into actionable integrations that align with enterprise requirements [52].

2.3.2 Designing Integrated and Strategic Solutions with BPMN and EIPs

The integration of BPMN and EIPs represents a powerful symbiosis for managing enterprise workflows and system interactions. BPMN provides a visual blueprint for processes, ensuring clarity and consistency in design. EIPs, on the other hand, offer the operational tools necessary to execute these workflows seamlessly across interconnected systems. According to Aier and Winter, this partnership promotes scalability and adaptability, allowing companies to match technical solutions with changing organizational objectives [49].

Organizations can handle common integration issues like message routing, data transformation, and error handling by leveraging EIPs in BPMN-modeled workflows. According to Ritter, this strategy guarantees that workflows stay stable, flexible, and responsive to shifting business needs, laying the groundwork for long-term expansion and innovation [52]. Thullner shows how open-source frameworks can facilitate this integration by providing adaptable and affordable solutions that complement the modular ideas of EIPs and BPMN [53].

The strategic integration of BPMN and EIPs reduces complexity and enhances transparency, providing organizations with a comprehensive framework for operational excellence. Hohpe and Woolf emphasize that EIPs simplify system interactions, enabling businesses to focus on strategic priorities rather than technical details [48]. Meanwhile, Curl and Fertalj highlight how

this approach bridges the gap between technical and business domains, fostering agility and competitiveness in an ever-evolving market landscape [50].

2.3.3 The Unified Advantage: BPMN and EIPs

The union of BPMN and EIPs delivers significant advantages, offering a cohesive solution for both design and execution of enterprise processes. This integration ensures that workflows are not only visually comprehensible but also practically executable, bridging the gap between abstract process design and tangible system performance. Technical complexity abstraction gives organizations the flexibility to adjust to shifting demands, improve stakeholder collaboration, and stay in line with strategic objectives.

Th Ultimately, this synergy offers a way to achieve operational efficiency, scalability, and transparency. In the face of constant change, it enables companies to keep a competitive edge in dynamic markets, optimize their workflows, and effectively handle complexity.

2.4 Summary

In conclusion, the fragmentation of information systems poses serious problems for contemporary organizations, impeding strategic alignment, decision-making, and operational efficiency. The development of IS from simple data storage solutions to complex multi-layered frameworks, including executive support systems (ESS), decision support systems (DSS), and transactional systems, has greatly improved organizational capabilities, as explained in the Information Systems section. However, silos, inefficiencies, and inconsistent data have resulted from the spread of specialized departmental systems, making it more difficult to respond effectively to shifting business demands and complicating processes.

Important strategies to deal with these problems are provided in the sections on Business Process Model Notation (BPMN) and Enterprise Integration Patterns (EIPs). BPMN offers a standardized, visual framework for mapping and analyzing workflows to help technical and non-technical stakeholders communicate and align. With this systematic approach, organizations can identify bottlenecks, optimize operations, and create flexible processes. EIPs improve BPMN by offering reusable patterns for managing data exchanges and system integration. Important technical problems like message routing, error handling, and real-time system-to-system communication are addressed by these patterns. Together, BPMN and EIPs develop integrated, actionable workflows that promote scalability, consistency, and long-term operational efficiency while enhancing departmental collaboration.

Additionally, the chapter emphasizes how the use of open-source technologies increases the potential of these frameworks. In addition to middleware solutions, workflow automation tools like Camunda offer strong platforms for putting BPMN and EIPs into practice in practical settings. These tools provide cost-effective substitutes for proprietary solutions, improve adaptability through modularity and flexibility, and guarantee scalability by adjusting to changing workloads.

In addition to facilitating smooth data flow, middleware solutions simplify the integration of disparate systems, guaranteeing accurate and consistent data throughout the company. Organizations are better equipped to handle current issues and proactively adjust to new opportunities in a rapidly changing technological landscape thanks to this integrated approach, which also improves operational excellence and strategic agility.

3 Systematic Literature Review

With an emphasis on tackling important issues like system integration, process transparency, and organizational adaptability, this systematic literature review looks at the body of research on developing frameworks for integrated information systems. It aims to evaluate how current methodologies and frameworks contribute to these areas, identifying theoretical insights that can guide the development of the proposed framework in this dissertation. For that, the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) [54] approach is taken, guided by critical research questions (cf. Section 1.3) to explore foundational principles, tools, and methodologies related to integrated information systems.

The selection criteria for pertinent studies are described in this chapter, along with particular keywords, eligibility requirements, and data sources. It also describes the procedures for gathering and combining data, guaranteeing an open and thorough research process.

3.1 Setting up

In order to move from the methodological orientation of the review to its practical execution, it was necessary to establish the conditions under which the systematic search would take place. This preparatory stage ensured that the process remained rigorous, transparent, and directly aligned with the research questions defined earlier. The setup was structured around three essential elements:

i. the identification of suitable data sources, with a focus on scientific databases that guarantee both breadth of coverage and reliability of peer-reviewed research in computer science and information systems (cf. Section 3.1.1);
ii. the definition of search terms, carefully refined through an iterative process so as to capture the multifaceted nature of integration frameworks and business process-centric architectures while filtering out irrelevant material (cf. Section 3.1.2); and
iii. the specification of inclusion and exclusion criteria, which established clear boundaries for the screening process and ensured that only studies of sufficient quality, accessibility, and thematic relevance were retained (cf. Section 3.1.3)

Taken together, these steps provided a coherent foundation for the subsequent stages of the PRISMA workflow, allowing the review to progress in a systematic and academically sound manner.

3.1.1 Data Sources

Identifying the appropriate data sources is a vital step in gathering relevant studies for this research. Table 1 outlines the primary data sources utilized in this study.

Table 1 Data Sources

Identifier	Database	URL
DS1	Google Scholar	https://scholar.google.com/
DS2	ACM Digital Library	https://dl.acm.org/
DS3	IEEE Xplore	https://ieeexplore.ieee.org/Xplore/home.jsp

These data sources encompass a diverse array of publications, including online books, conference proceedings, newspapers, articles, and academic studies. They distinguish themselves by offering a substantial collection of peer-reviewed articles that have been rigorously curated for their relevance to the field of information technology, ensuring the inclusion of high-quality, authoritative research.

3.1.2 Search Terms

A search query is developed to locate studies most relevant to the topic, focusing on frameworks and architectural aspects in enterprise systems. The keywords utilized include "*business process-centric framework*", "*process-centric framework*", "*integration framework*", "*enterprise framework*", "*information systems architecture*" and "*enterprise architecture*". Additional focus areas include "*key characteristics*", "*components*", "*integration patterns*" and terms related to seamless or faultless communication, adaptability, and flexibility.

Given the nuanced purposes and researchable content of each question, the query is designed to capture studies that explore the components and characteristics of enterprise frameworks, and integration patterns in open-source systems. To address this complexity, multiple search parameters and logical groupings were employed to ensure comprehensive and relevant results.

The Trial-and-Error Search method involves starting with an initial search query and refining it iteratively based on the results obtained [55]. This approach was utilized, and the final search query is illustrated in Figure 4.

3.1.3 Inclusion and Exclusion Criteria

The inclusion criteria decided for the articles selected are:

- **IC1** – Studies discussing integration frameworks, enterprise integration patterns, and their application in information systems.

- **IC2** – Studies focusing on information system architectures, including their adaptability and usability in diverse environments.
- **IC3** – Studies exploring the role of open-source technologies in developing frameworks for seamless communication in complex system environments.

The exclusion criteria for the articles are:

- **EC1** – Studies lacking an abstract.
- **EC2** – Studies without full-text access.
- **EC3** – Studies not available in English.
- **EC4** – Studies published before the year 1999.
- **EC5** – Studies primarily focused on unrelated domains, such as data mining.
- **EC6** – Studies that do not address software development, system integration, or architecture.
- **EC7** – Studies solely centered on the development of information systems.

```
(
("business process-centric framework"  OR "integration framework" OR "enterprise framework")
AND(
    (("information systems architecture" OR "enterprise architecture")
AND
      ("key characteristics" OR "components" OR "component" OR "parts") )
    OR
    ( ("integration patterns" AND "open-source"
AND
            ("seamless communication" OR "faultless communication" OR "integration" OR "integrated" OR "adaptability" OR "flexibility")
        ))
   )
)
```

Figure 4 Search Query.

3.2 Research Methodology

According to the adopted methodology (PRISMA [54]), the studies selection comprises applying three key stages (as represented in Figure 5):

- **Identification**: This stage involves collecting all potentially relevant studies that align with the research questions based on predefined eligibility criteria. A total of 787 records were identified across three databases: ACM Digital Library (96), IEEE Xplore (281), and Google Scholar (410). During this phase, 19 duplicated records were

removed, along with 108 records deemed irrelevant for other reasons, leaving 679 records for screening.

- **Screening:** The studies identified in the previous step were reviewed to determine their suitability according to the inclusion criteria. Initially, articles were categorized as either "Relevant" or "Irrelevant." Out of the 679 records screened, 617 were excluded as they did not meet the criteria. The remaining 63 reports were sought for retrieval; however, 38 were not retrieved. This left 25 reports, which were analyzed in greater detail. After this assessment, all 25 reports met the inclusion criteria and were confirmed relevant to the research.
- **Included:** In this final stage, the 25 selected studies that met all criteria were included for use in answering the research questions. These studies helped to address the main goals of the research and served as the basis for the systematic review.

By following these steps and applying specific search queries along with inclusion and exclusion criteria, relevant data were systematically gathered to address the research questions effectively. This structured process ensures a transparent and rigorous approach, culminating in a focused selection of high-quality studies that align with the research goals.

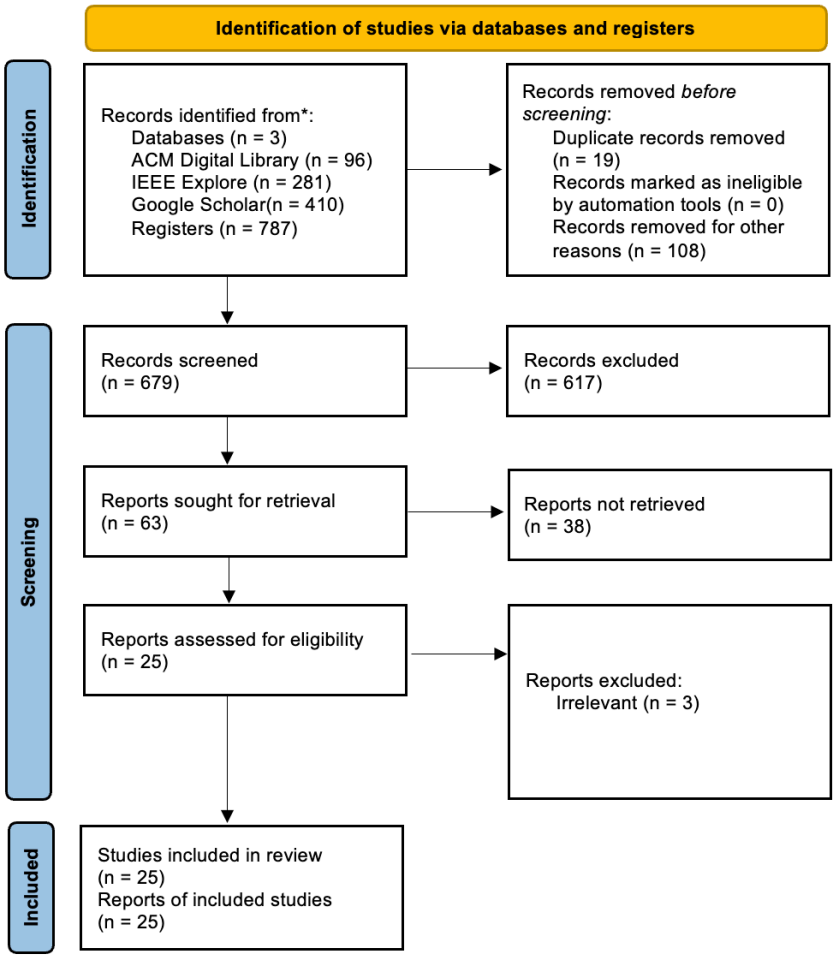


Figure 5 PRISMA systematic

The selected studies are identified and related to the corresponding research question in Table 2 and Table 3.

3.3 Results – Studies Analysis

The following section presents the results of the systematic literature review, building on the methodological setup defined earlier and guided by the research questions introduced in Chapter 1. Having established the inclusion criteria and applied the PRISMA methodology, the selected corpus of studies is now examined with the aim of deriving insights directly relevant to the objectives of this dissertation. Rather than offering a fragmented list of works, the analysis seeks to position the studies within a coherent narrative that shows how the literature has progressively evolved to address the integration of information systems from both theoretical and technological perspectives. Particular emphasis is placed on identifying common characteristics, recurring patterns, and the use of open-source approaches, as these dimensions provide the conceptual foundation for the framework developed later in this work. The analysis proceeds in a structured manner: first presenting an overview of the studies retrieved, then mapping them against the research questions, and finally discussing the findings that emerge from this mapping.

3.3.1 Overview

A comprehensive summary of the studies that are part of this review is helpful before delving into each of the research questions separately. Tables 2 and 3 provide a summary of the corpus, separating contributions that support Research Question 2 (RQ2) from those that inform RQ1 (RQ1). There are two purposes for this summary. In one sense, it creates a chronological and thematic arrangement of the literature, demonstrating the evolution of the discussion of integration frameworks from early enterprise architecture conceptualizations to more recent works emphasizing sustainability, adaptability, and interoperability. However, it also draws attention to the wide range of fields in which these frameworks have been used, from government and healthcare information systems to process-centric architectures and inter-firm collaborations. The overview lays the foundation for a more thorough examination by placing the works in this manner, elucidating not only the abstract contributions made by the literature but also how these contributions come together to address the particular research questions of this dissertation.

Thirteen articles, with an average publication year of 2015, covering the years 1999–2022, were found to answer Research Question 1. The ongoing development of enterprise architecture and integration framework research is reflected in this extensive timeline. The groundwork for later advancements in enterprise information system architecture was laid by early research, including Zachman. These seminal works, which make up 15% of the identified articles, are still relevant today.

In the mid-2000s, research shifted towards creating frameworks that address specific integration challenges. For example, Gang & Quan introduced a synergetic theory-based framework, while Van Nuffel & De Backer explored multi-abstraction business process modeling. Together, these works account for 31% of the articles and emphasize a growing focus on operationalizing enterprise architecture in practical, multidisciplinary contexts.

Table 2 Articles used to answer research question 1

REF	Title	Year	Research Question	Author(s)
[56]	Application Integration Patterns and their Compositions: Foundations, Formalizations, Solutions	2019	RQ1	Ritter (2019)
[57]	A framework for information systems architecture	1999	RQ1	Zachman (1999)
[58]	Aggregated framework of enterprise information system based on synergic theory	2006	RQ1	Gang & Quan (2006)
[59]	Supporting Enterprise Integration with a Multidimensional Interoperability Framework	2015	RQ1	Delgado (2015)
[60]	Common quality measures for Enterprise Architecture	2022	RQ1	Karande & Joshi (2022)
[61]	On Integrated Governance of e-Government: Technology, Institution, Market, and Government	2010	RQ1	Gui-sheng (2010)
[62]	Multi-abstraction layered business process modeling	2012	RQ1	Van Nuffel & De Backer (2012)
[63]	Architecture Frameworks	2022	RQ1	Weilkiens et al. (2022)
[64]	Adaptive Enterprise Architecture: Initiatives and Criteria	2020	RQ1	Wissal et al. (2020)
[65]	Service Oriented Enterprise Architecture Framework	2010	RQ1	Haki & Forte (2010)
[66]	Combining BPM and EA in Complex IT Projects: (A Business Architecture Discipline)	2011	RQ1	von Rosing et al. (2011)
[16]	Enterprise architecture advances in technical communication	2015	RQ1	Bellman & Griesi (2015)
[67]	A Strategy Framework For Incorporating Sustainability Into Enterprise Architecture	2020	RQ1	Perdana et al. (2020)
[68]	Integrating Business Models and Enterprise Architecture	2014	RQ1	Petrikina et al. (2014)

Recent studies, comprising 54% of the articles (from 2020 onwards), highlight contemporary priorities such as adaptability and sustainability. Perdana et al., who concentrate on incorporating sustainability into enterprise frameworks, and Wissal et al. , who study adaptive enterprise architecture, are two examples. These results show that tackling contemporary issues like environmental responsibility and dynamic system requirements is becoming more and more important. The study's themes include business-IT alignment (23%), enterprise integration (38%), and the creation of sophisticated interoperability frameworks (15%). This development demonstrates a distinct path from theoretical underpinnings to real-world, industry-specific applications.

Eleven articles covering the years 2002–2021 with an average of 2012 were found for Research Question 2. With 64% of the articles published between 2002 and 2013, the distribution of

studies reveals a greater concentration in earlier years, suggesting a strong early focus on foundational and domain-specific integration challenges.

Table 3 Articles used to answer research question 2

REF	Title	Year	Research Question	Author(s)
[69]	Tutorial: open source enterprise application integration - introducing the event processing capabilities of Apache Camel	2013	RQ2	Emmersberger & Springer (2013)
[70]	Patterns for emerging application integration scenarios	2017	RQ2	Ritter et al. (2017)
[71]	GLUEMan: a WBEM-based Framework for Information Providers in Grid Services	2008	RQ2	Andreozzi et al. (2008)
[72]	Theoretical frameworks for information systems integration in inter-firm collaborations	2021	RQ2	Bakhaev (2021)
[73]	Architecture for a large healthcare information system	2002	RQ2	Mukherji et al. (2002)
[74]	Developing a hospital information system ecosystem for creating new clinical collaboration methodologies	2012	RQ2	Schramm et al. (2012)
[75]	Applying EA Perspective to CRM: Developing a Competency Framework	2008	RQ2	Caldeira et al. (2008)
[76]	Improving the composition and assembly of services in process-oriented environments	2008	RQ2	Ramljak (2008)
[77]	Alignment of Business and IT Architectures in the German Federal Government: A Systematic Method to Identify Services from Business Processes	2013	RQ2	Birkmeier et al. (2013)
[78]	Enterprise Architecture Analysis and Network Thinking: A Literature Review	2016	RQ2	Santana et al. (2016)
[79]	Verifying data integration agents with deduction-based models	2013	RQ2	Klimek et al. (2013)

Early research explored practical challenges in specialized domains. For example, Mukherji et al. analyzed healthcare information systems, while Andreozzi et al. introduced a framework for information providers in grid services. These studies reflect the initial push to address technical interoperability and sector-specific IT challenges, accounting for 45% of the articles.

Research expanded until the 2010s to incorporate more diverse viewpoints and approaches. Birkmeier et al. emphasized IT architecture alignment in government settings, while Santana et al. examined network thinking in enterprise architecture. These mid-decade works, which make up 36% of the articles, concentrate on enhancing interorganizational collaboration and coordinating IT systems with institutional requirements.

More recent studies, such as Bakhave, delve into theoretical frameworks for inter-firm collaborations, reflecting a shift toward fostering collaboration in dynamic and increasingly interconnected environments. These newer studies (published after 2016) account for 18% of the identified articles.

Thematically, RQ2 articles center on application integration frameworks (40%), healthcare and public-sector IT systems (30%), and collaborative ecosystems (30%). This focus highlights a

consistent trajectory from solving technical challenges to addressing broader systemic and collaborative issues.

This study explores two critical dimensions of this domain. First, it identifies the essential characteristics and components required for a business process-centric framework to function effectively across a wide range of architectural environments. Second, it investigates how enterprise integration patterns (EIPs) and open-source technologies contribute to the conception and construction of frameworks that promote seamless communication and adaptability in complex system ecosystems.

On the next sections these studies are exploited in order to find / provide answers to the underlying research questions.

3.3.2 Answering Research Question 1

> *RQ1: What key characteristics and components should a business process-centric framework have to be usable in the wide range of possible information systems architectures?*

A comprehensive strategy that incorporates flexibility, scalability, and alignment with organizational objectives is needed to create a business process-centric framework that can operate across a variety of information systems architectures. While promoting operational and strategic coherence, such a framework must be flexible enough to accommodate the intricacies of various technological environments.

The ability of these frameworks to modularize their constituent parts is one of their distinguishing features. This framework's modularity guarantees that various components function separately while preserving overall integration. Ritter discusses how modular integration patterns facilitate the management of large systems and enable companies to swap out or modify individual parts without compromising the structure as a whole [56]. Similar to this, Zachman highlights the importance of structured architectures in handling the increasing complexity of modern information systems by utilizing modular design principles [57]. Gang and Quan further emphasize this modularity by demonstrating how synergistic integration frameworks can enhance communication and information transfer within and between organizational units. According to their research, system agility is increased when internal synergy mechanisms are isolated from external procedures like e-commerce platforms [58].

Equally significant is the ability of the framework to scale effectively. Organizations frequently deal with unpredictable workloads and growing data volumes, requiring for a framework that can adapt to these alterations without compromising effectiveness. In order to handle these scalability issues, Delgado suggests an interoperability framework that emphasizes the interaction of lifecycle stages, structural compliance, and quality of service [59]. Karande and Joshi build on this by introducing quality metrics such as response time, resource utilization

efficiency, throughput and horizontal scalability as a core aspect of enterprise architecture, ensuring that systems can adapt to varying demands [60]. Gui-sheng reinforces this idea in the context of e-governance, proposing an integrated framework that accommodates technological, institutional, and market dimensions to handle cross-departmental coordination effectively [61].

To bridge the diverse layers of an enterprise, frameworks must incorporate abstraction mechanisms that unify strategic objectives with detailed operational processes. Van Nuffel and De Backer illustrate this concept through a multi-layered business process modeling framework that harmonizes high-level enterprise perspectives with task-specific workflows [62]. This layered approach ensures coherence across all organizational dimensions, enabling decision-makers to align overarching goals with the day-to-day realities of their operations. This concept is further developed by Weilkiens et al., who advocate for abstraction as a means to manage the complexity inherent in multi-faceted enterprise systems [63].

The dynamic nature of business environments also necessitates frameworks that are inherently adaptable. Frameworks need to be flexible enough to adapt to the quickly shifting technological and market environments that organizations must contend with. The concept of adaptability in enterprise architectures is examined by Wissal et al., who highlights the necessity of frameworks that proactively react to changes in the outside world [64]. Haki and Forte support this flexibility by showing how incorporating service-oriented principles into enterprise frameworks promotes alignment with changing organizational strategies [65]. Similarly, von Rosing et al. highlight the integration of business process management (BPM) with enterprise architecture (EA), showing how this combination ensures frameworks remain flexible and aligned with strategic priorities, even in the most complex IT projects [66].

Standardization plays a vital role in ensuring that frameworks maintain consistency and interoperability across heterogeneous systems. Widely accepted tools like BPMN and UML facilitate the modeling of business processes in ways that are compatible with various architectural designs. Bellman and Griesi explore the role of standardized frameworks in aligning business strategies with technological infrastructures [16], while Perdana et al. emphasize the importance of compliance and sustainability as integral aspects of modern enterprise systems [67].

Beyond technical considerations, the integration of business models into enterprise frameworks further enhances their practical utility. Petrikina in 2014 analyzed the potential of incorporating business models within enterprise architecture processes, demonstrating how such integration ensures alignment between operational practices and strategic objectives[68]. This creates a cohesive environment where technological frameworks support not only the operational efficiency of an organization but also its broader ambitions.

In conclusion, a business process-centric framework that can operate effectively across diverse information systems architectures must integrate a blend of interconnected characteristics that work together to ensure its usability and relevance. The framework's design should prioritize modularity to allow seamless integration and independent operation of its components,

enabling organizations to adapt parts of the system without causing disruptions. Scalability becomes essential as businesses encounter fluctuating demands and expanding data volumes, necessitating a structure capable of maintaining efficiency under varying conditions. The framework can create a cohesive system that synchronizes overarching goals with daily operations by utilizing abstraction mechanisms to balance strategic objectives with operational realities. Lastly, standardization offers a basis for consistency and interoperability among various systems, while adaptability guarantees that the framework stays robust and responsive in the face of changing market dynamics and technological breakthroughs. These components work together to create a strong and all-encompassing framework that supports operational effectiveness and is in line with an organization's long-term strategic vision across a variety of information systems.

3.3.3 Answering Research Question 2

> *RQ2: How can enterprise integration patterns and open-source technologies contribute to the conception and construction of a business process-centric framework that promotes seamless communication and adaptability in complex system environments?*

Enterprise integration patterns (EIPs) [48] and open-source technologies play a key role in designing frameworks focused on business processes that promote smooth communication and flexibility in challenging situations. By providing both theoretical foundations and practical tools, they address the multifaceted challenges of integrating diverse systems while supporting continuous improvement and responsiveness.

EIPs provide a structured approach to solving common integration challenges by offering reusable solutions that streamline interactions between disparate systems. The seminal work of Hohpe and Woolf, as explored in practical contexts by Emmersberger and Springer, underscores the role of EIPs in building cohesive integration strategies [69]. Frameworks like Apache Camel [80] operationalize these patterns, implementing features such as message brokers, adapters, and routing mechanisms to ensure seamless data flow between heterogeneous systems. The value of such patterns in bringing complex technological ecosystems together is demonstrated by their efficient deployment.

The flexibility and scalability of integration frameworks are enhanced by open-source technologies. According to Ritter et al., tools like Jenkins are now essential to pipelines for continuous integration and delivery because they allow for iterative improvements and guarantee that frameworks stay in line with evolving organizational requirements [70]. Another essential component of open-source infrastructure, Kubernetes, offers reliable container orchestration that makes managing and deploying distributed systems easier [71]. These tools enable businesses to establish in place scalable solutions that may evolve to accommodate shifting workloads and new technology.

In complex multi-enterprise contexts, integration frameworks must also support inter-firm collaborations. Bakhaev examines theoretical frameworks for integrating information systems across organizational boundaries, addressing challenges such as system heterogeneity, workflow alignment, and governance structures [72]. Because they offer standardized methods to guarantee seamless communication and collaboration between businesses, EIPs and open-source tools become essential in these kinds of settings. For instance, middleware tools like Kafka and event-driven architectures [81] and RabbitMQ [82]enable real-time responsiveness, ensuring seamless interactions between disparate systems within a collaborative ecosystem.

Open-source technologies meet the requirements for real-time responsiveness in complex environments in addition to scalability. Event-driven architectures guarantee effective message processing and delivery, assisted by middleware technologies such as Kafka and RabbitMQ. Mukherji et al. highlight the importance of dependable messaging systems in extensive operations, demonstrating how middleware facilitates smooth communication [73]. Schramm et al. further explore the integration of monitoring tools like Prometheus [83], which provide real-time insights into system performance and enable proactive adjustments to prevent disruptions [74].These capabilities are essential in inter-firm contexts, where coordinated workflows require high levels of reliability and transparency.

Security and compliance are indispensable in frameworks that rely on EIPs and open-source tools. Vault [84], an open-source security solution, provides mechanisms for managing sensitive data, ensuring compliance with regulatory standards [75]. Furthermore, standardized communication protocols, such as REST APIs [85] and SOAP [86], enhance interoperability while maintaining data integrity [76]. These considerations reinforce the reliability of frameworks in environments where data privacy and protection are paramount.

These frameworks' flexibility and modularity are directly related to their adaptability. Wissal et al. highlight the necessity of frameworks that can anticipate and react to change in their criteria for assessing adaptive enterprise architectures. According to Birkmeier et al., firms can maintain their agility in the face of changing requirements by facilitating this adaptability through the alignment of business and IT architectures [77]. By leveraging the modularity of EIPs and the versatility of open-source technologies, organizations can construct frameworks that accommodate both current operational needs and future challenges.

Innovation is another significant advantage provided by the integration of EIPs and open-source technologies. Tools like Grafana [87] offer real-time visualizations, enabling organizations to analyze data and make informed decisions efficiently [78]. Rule-based approaches, as demonstrated by Klimek et al., further enhance the framework’s capacity to integrate diverse data sources and automate decision-making processes [79]. Beyond contributing to increasing operational effectiveness, these skills put businesses in a position that can take advantage of new opportunities.

Innovation is another significant advantage provided by the integration of EIPs and open-source technologies. Tools like Grafana [87] offer real-time visualizations, enabling organizations to analyze data and make informed decisions efficiently [78]. Rule-based approaches, as

demonstrated by Klimek et al., further enhance the framework's capacity to integrate diverse data sources and automate decision-making processes [79]. These capabilities not only improve operational efficiency but also position organizations to capitalize on emerging opportunities. In summary, open-source technologies and enterprise integration patterns are essential to the development of business process-centric frameworks because they offer a mutually beneficial combination of theoretical ideas and useful tools that tackle the difficulties of adaptability and smooth communication in complex settings. Enterprise integration patterns, as operationalized by frameworks such as Apache Camel, provide structured solutions to common integration challenges. These solutions facilitate cohesive strategies that use technologies like message brokers, adapters, and routing techniques to unify disparate systems, ensuring consistent data flow and operational harmony. The flexibility and scalability offered by open-source technologies, such as Jenkins for continuous integration and Kubernetes for distributed system orchestration, reinforce this foundation and allow for iterative improvements and responsiveness to shifting organizational demands. Through middleware solutions like Kafka and RabbitMQ, standardized protocols like REST APIs, and security frameworks like Vault, these technologies also address crucial elements of real-time responsiveness, security, and compliance, promoting data integrity and interoperability across various and dynamic enterprise ecosystems. These frameworks align business and IT architectures by combining modularity, adaptability, and innovation. In addition to using tools like Grafana for automated decision-making and real-time insights, this allows organizations the flexibility to respond to changing needs. Open-source technologies and enterprise integration patterns work together to ensure that frameworks are safe, scalable, and future-ready, allowing businesses to successfully and robustly navigate the complexities of modern enterprise environments.

3.4 Conclusion

This literature review examined the theoretical underpinnings and practical methods of developing frameworks for integrated information systems by focusing on topics like organizational adaptability, process transparency, and system integration. It was guided by the preliminary question RQ0: *"How can a modular, business process–centric framework respond to the challenges of the software development life cycle in new information systems, while also ensuring their construction, integration, and interoperability with those already in place within an organization?"* In addressing this guiding question, the chapter laid out a theoretical and practical pathway for the design of modular, process-centric integration frameworks that reflect the complexity of modern enterprise environments. This pathway was subsequently refined and expanded through the two complementary research questions, RQ1 and RQ2, which provided a structured approach for evaluating both architectural principles and viable implementation strategies, were used to develop and further expand this path.

The review emphasized the significance of a well-rounded framework design that skillfully incorporates several fundamental principles in order to answer RQ1. Effective frameworks are able to function in a modular format, enabling independent updates and maintenance without

affecting the system as a whole. They also show that they can scale well to accommodate a range of shifting operational requirements. Furthermore, a coherent organizational structure is promoted when strategic goals and operational procedures are aligned through suitable degrees of abstraction. Such frameworks' relevance is further increased by their flexibility in responding to changing market and technological conditions, and their consistent standardization guarantees interoperability across various platforms and systems. Together, these integrated principles create a strong design foundation that allows frameworks to be adaptable and resilient in a variety of enterprise contexts.

Building on these observations, RQ2 emphasized how important enterprise integration patterns (EIPs) and open-source technologies are to the functioning of such frameworks. By facilitating message routing and orchestration across dispersed components, EIPs—best represented by products such as Apache Camel—offer reusable solutions for persistent integration problems. Scalability, continuous improvement, and responsiveness to changing organizational demands are further made possible by the use of open-source platforms like Jenkins and Kubernetes. In situations involving multiple enterprises, middleware technologies like Kafka and RabbitMQ are crucial for maintaining real-time data flows. Simultaneously, interoperability is strengthened, data integrity is protected, and regulatory compliance is guaranteed by security measures like Vault and standardized communication protocols like REST. When taken as a whole, these components show how using open-source tools and integration styles directly improves the framework's practicality.

The integration of RQ1's theoretical foundations with the useful tools investigated in RQ2 highlights the development from conceptual design to practical implementation, highlighting the significance of combining enabling technologies with fundamental design principles to create flexible, scalable frameworks that can meet the complex requirements of various system environments. The integration and interoperability issues of complex enterprise systems can be addressed comprehensively and practically through to this complementary relationship. With this alignment, this review offers a pathway to developing modular, business process-centric frameworks that enhance organizational efficiency and adaptability

.

4 Framework Proposal

This chapter advances the proposal of the integration framework by clarifying its purpose, scope, and architectural rationale. The design is structured according to the C4 model [88], a recognised methodology in software architecture that represents systems through four successive layers of abstraction: Context, Container, Component, and Code. By following this model, the framework can be articulated in a systematic and coherent manner, progressing from a global view of the system's external interactions to the detailed specification of its internal elements.

In the context of this dissertation, the C4 model is especially suitable because the proposed framework operates in a multi-system integration environment**,** where it is essential to describe not just the internal mechanisms but also how it positions itself as a mediator between heterogeneous external systems, engines, and user interfaces. By employing this model, the chapter provides a clear narrative that progressively zooms in: first outlining what the system is and who interacts with it (Level 1), then identifying the main structural building blocks (Level 2)**,** and finally unpacking the critical components (Level 3) that give the framework its functional depth.

4.1 Requirements

The specification of requirements constitutes a critical stage in the architectural design of the framework, as it determines both the scope of its functionality and the qualities that must govern its behaviour. In this dissertation, requirements are structured according to the FURPS+ model [89], a classical methodology in software engineering proposed by Robert Grady at Hewlett-Packard in the 1990s. This model—comprising Functional, Usability, Reliability, Performance, and Supportability—has been widely adopted because it provides a balanced view that unites functional expectations with the non-functional attributes that determine long-term viability. By adopting FURPS+, the requirements specification of the framework gains both clarity and depth: it specifies what the framework must do and how it must behave under the pressures of real-world operation.

Central to this specification are the actors who interact with the framework, since their goals determine the nature of the requirements. Three categories of actors are identified: Administrators, Developers, and End Users. Each actor within the framework assumes a distinct function and advances particular interests, all of which must be explicitly acknowledged to ensure that requirements remain coherent with organizational priorities. The Administrator oversees system configurations, policies, and governance mechanisms, aligning operational behaviour with strategic objectives. The Developer focuses on modelling and deploying processes and integrations, using the orchestration layer to transform technical specifications into executable workflows. The End User interacts through dedicated interfaces to perform

human-oriented tasks, thereby ensuring that workflows integrate both automated routines and decision points requiring contextual judgment. These roles, however, are not fixed boundaries: developers may occasionally assume administrative responsibilities in complex scenarios, while end users may contribute to process definitions or small-scale adaptations typically associated with development. Such flexibility illustrates the framework's adaptability to varied organizational contexts, where responsibilities can be specialised or shared depending on team structure and expertise.

- **Administrators**
  - Configure and maintain the operational environment.
  - Manage authentication, authorization, and governance policies.
  - Monitor system health and audit logs to ensure compliance and stability.
  - Their primary interest lies in security, governance, and operational reliability.
- **Developers**
  - Design, upload, and deploy integrations.
  - Generate client modules from external specifications and develop orchestration logic.
  - Validate, publish, and monitor integration services.
  - Their primary interest lies in productivity, flexibility, and engine-agnostic interoperability.
- **End Users**
  - Interact with business processes instantiated through the Tasklist.
  - Claim and complete user tasks as part of human-in-the-loop workflows.
  - Provide operational data that is routed into integrations.
  - Their primary interest lies in usability, accuracy, and the seamless execution of processes.

4.1.1 Functional Requirements

At the centre of this specification lie the Functional Requirements (FRs), which (most of it) are directly derived from the actor–goal interactions represented in the Use Case Diagram (Figure 6). This diagram illustrates the roles of Administrators, Developers, and End Users, and the ways in which they interact with the framework's containers. By establishing explicit traceability between use cases and requirements, the framework guarantees that every interaction has been translated into a verifiable system capability, thereby aligning design choices with user needs and organizational objectives.

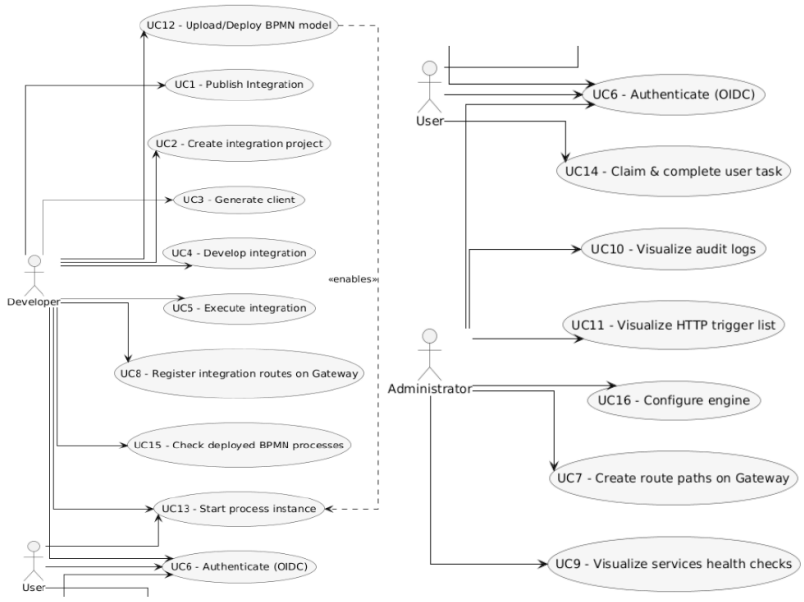


Figure 6 Use Case View situating Administrator, Developer, End User across Framework

At the foundation of this ecosystem lies the design and development of integrations. The process begins with UC02 – Create Integration Project, which provides developers with a structured environment where new integrations can be scaffolded. Building on this foundation, UC03 – Generate Client enables the transformation of external system specifications into reusable client modules, ensuring that each integration begins with a formalized and contract-driven artifact. Following client generation, UC04 - Develop Integration integrates external APIs with the framework's process flows to enrich these projects with orchestration logic. This lifecycle culminates with UC01 - Publish Integration, where the developed module is added to the framework's catalog and made available for use. Formally, this invocation is stated in UC05-Execute Integration, where integrations—which have now been operationalized—extend the orchestration of the framework into diverse enterprise systems.

Governance and access control form a second, equally critical dimension. UC06 – Authenticate (OIDC) represents the initial checkpoint, ensuring that every request entering the system is tied to a verified identity. Although authentication is typically defined as a functional requirement rather than a dedicated use case, in this dissertation it is deliberately modelled as UC06. The rationale is to capture its behaviour in sequence diagrams and to demonstrate, at a deeper level, how identity verification integrates with subsequent orchestration flows. Once authenticated, the logical routes that services are made available via are established by administrators using UC07 - Create Route Paths on the gateway. These routes acquire operational meaning in UC08 – Register Integration Routes on gateway, binding integrations to specific endpoints and guaranteeing that they are exposed only through centrally controlled, auditable channels. Together, these use cases embody the principle that security and governance are not embedded deep within services but enforced consistently at the edge of the framework.

The orchestration of processes is governed by the Process Execution Manager, where several use cases ensure that BPMN models evolve from static diagrams into living workflows. UC12 – Upload/Deploy BPMN Model initiates this cycle, transforming abstract process definitions into deployable entities within the system. These processes are brought to life in UC13 – Start Process Instance, allowing workflows to transition from design to execution. As processes unfold, UC14 – Claim & Complete User Task ensures that human actors remain integral to the orchestration, capturing the indispensable contributions of end users in decision-making and validation steps. Oversight of process states is expressed in UC15 – Check Deployed BPMN Processes, which enables administrators and developers to confirm the availability and status of models. Meanwhile, UC16 – Configure Engine provides the administrative capacity to adapt execution environments, ensuring that orchestration remains engine-agnostic and responsive to organizational needs.

Finally, observability guarantees that the framework operates with transparency and accountability. UC09 – Visualize Services Health Checks provides administrators with a rapid assessment of component availability, ensuring the infrastructure remains operational. UC10 – Visualize Audit Logs goes further, exposing detailed records of actions and requests, essential for governance and compliance. Complementing these, UC11 – Visualize HTTP Trigger List provides insight into deployed serverless functions, offering administrators visibility into the endpoints that underpin integration execution. Together, these use cases transform monitoring from a peripheral activity into an integral aspect of the framework's governance model.

Taken as a whole, the sixteen use cases articulate the framework's functional scope in a way that balances creativity, governance, orchestration, and assurance. Developers are empowered to create integrations through a contract-first methodology. Administrators enforce policies and configure execution environments, while end users contribute through human tasks integrated seamlessly into workflows. The framework, therefore, emerges not as a set of isolated features but as a unified environment in which integration and orchestration are designed, governed, executed, and monitored with equal rigor.

4.1.2 Non-Functional Requirements

4.1.2.1 Usability (U)

The usability of the framework should derive from its ability to provide role-specific access points that abstract orchestration complexity while presenting users with clear, structured, and contextually appropriate actions. Instead of generic dashboards, the framework should establish differentiated environments aligned with the responsibilities and authority of each actor, thereby upholding the principle of separation of concerns. Such a design approach ought to partition governance, integration design, and task execution into distinct operational spheres, with the purpose of reducing cognitive load, facilitating faster user proficiency, and systematically minimizing the risk of errors arising from cross-domain interactions

- NFR-U01 — Actor-specific UIs. The system shall provide dedicated UIs, each one constrained to a single actor and its responsibilities.
- NFR-U02 — Consistent interaction and feedback. Critical actions (deploy BPMN, publish integration, create/update route) shall provide immediate, unambiguous status and error messages, including operation IDs for later audit.
- NFR-U03 — Visual validation for BPMN. BPMN uploads shall render a visual preview before deployment, enabling users to confirm the model aligns with intent.
- NFR-U04 — Guided completion of human tasks. User tasks shall be presented through structured and validated forms, with field constraints and client-side validation to reduce errors during submission.

4.1.2.2 Reliability (R)

Defensive validation, idempotent operations, and fault isolation across containers secure reliability. BPMN artifacts are validated before deployment; deploy/redeploy actions do not corrupt state. Failures are captured centrally and contained to the smallest runtime unit (a function or request), thereby preserving system stability under adverse conditions.

- NFR-R01 — BPMN validation gates. Only syntactically and semantically valid BPMN shall be deployable; invalid models shall be rejected with structured diagnostics.
- NFR-R02 — Idempotent lifecycle operations. Repeated deployments/publishes of the same artifact shall not create duplicates; version and status remain consistent.
- NFR-R03 — Fault isolation at runtime. An integration failure shall not crash the Core Backend or PEM; the impact is confined to the failing function/request with clear error propagation.
- NFR-R04 — Durable metadata & auditability. Raw BPMN XML and deployment metadata (status, version, timestamp, tags) shall be persisted to support rollback, re-deploy, and incident analysis.
- NFR-R05 — Centralised exception handling. The Core Backend shall capture, classify, and log errors with correlation IDs, exposing hooks for retry/compensation when applicable.

4.1.2.3 Performance (P)

Performance follows from lean ingress and elastic execution. The Gateway facilitates fast authentication and authorization, as well as routing; the serverless plane enables low-latency invocation with horizontal scaling. Performance targets are pragmatic, oriented to enterprise responsiveness rather than synthetic benchmarks.

- NFR-P01 — Low-latency ingress. Under nominal load, Gateway authentication + routing shall add only a small, bounded overhead to end-to-end orchestration.
- NFR-P02 — Elastic function execution. Integration functions shall scale horizontally based on demand, sustaining throughput without breaching latency budgets defined for critical paths.

- NFR-P03 — Predictable cold-start impact. The platform shall minimise and amortise cold-start penalties (e.g., via warm pools or proactive scale-to-ready strategies) for frequently invoked integrations.
- NFR-P04 — Non-blocking orchestration. Orchestration calls (deploy/start) shall complete within operator-acceptable SLAs, with progress states surfaced to the UI to avoid perceived timeouts.

4.1.2.4 Supportability (S)

Supportability is achieved through modular decomposition, portability, and a clear path to observability. Each container is independently deployable; integrations follow a uniform, containerised project template; and while the prototype offers only lightweight checks, the architecture explicitly anticipates enterprise monitoring.

- NFR-S01 — Modular, independently deployable containers. The containers shall be independently versioned and deployable.
- NFR-S02 — Portable integrations. All integrations shall be packaged as container images built from a standard template, enabling environment-agnostic deployment.
- NFR-S03 — Configuration as code. Engine bindings, routes, and policies shall be expressed declaratively (files or manifests) to enable repeatable, reviewable changes.
- NFR-S04 — Extensible engine adapters. Adding a new workflow engine shall require only a new adapter, without modifications to Gateway, UIs, or existing adapters.
- NFR-S05 — Path to observability. Beyond basic reachability checks, the system shall expose metrics/logs in a way that can be integrated with enterprise stacks (e.g., Prometheus/Grafana) without architectural changes.
- NFR-S06 — Backwards-compatible evolution. Minor/patch upgrades of containers or adapters shall not break public contracts (routes, payloads) without documented migration notes.

4.2 Boundaries and responsibilities

At the highest level, the architecture of the proposed framework is structured around a set of boundaries that define both its scope of control and the distribution of responsibilities across the ecosystem in which it operates. The context boundary delineates the wider environment, encompassing the external systems, cross-cutting services, and human actors with which the framework interacts. The system boundary, by contrast, encloses the components and functions over which the framework assumes direct ownership and accountability. The articulation of these two perimeters, illustrated in Figure 7, provides not only a visual abstraction of the framework's position but also a conceptual foundation for understanding its role as an orchestrator and integrator of heterogeneous systems.

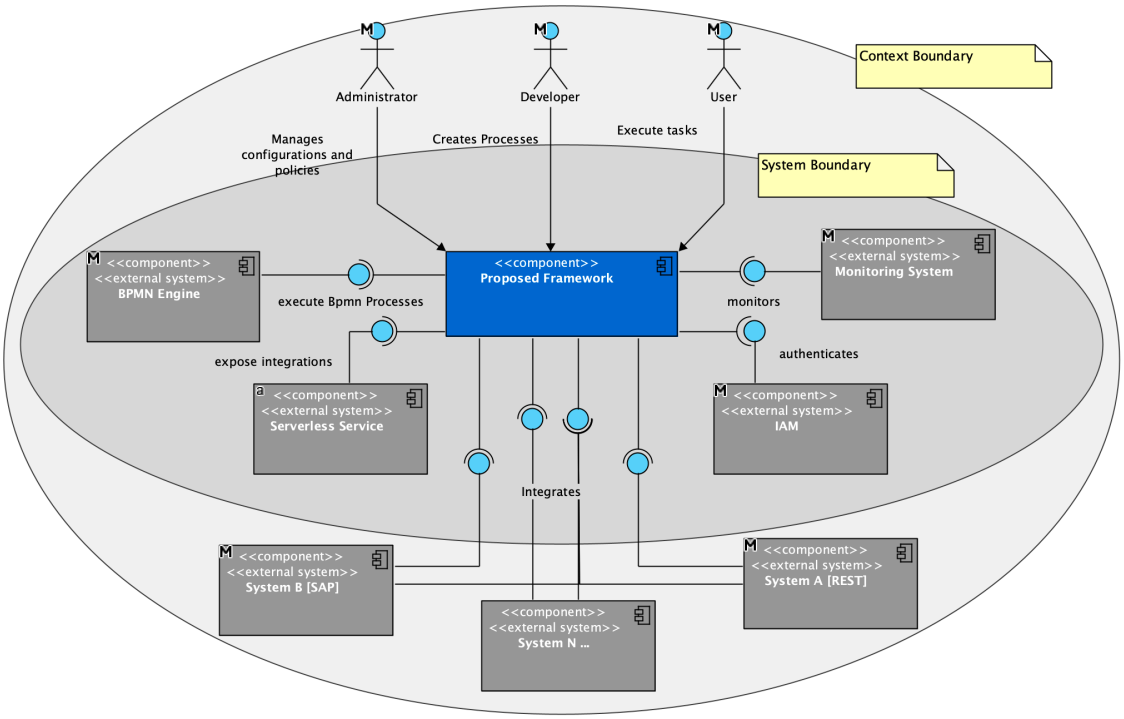


Figure 7 System Boundaries

At the center of the system boundary lies the Framework itself, represented as a cohesive unit whose primary responsibility is to orchestrate business processes modeled in BPMN and to mediate integrations across disparate systems. It assumes the task of coordinating process flows, managing the execution of integrations, and ensuring that business-level intentions are consistently translated into executable actions and properly authorized. A distinctive aspect of the framework lies in its deliberate commitment to technological agnosticism. Rather than embedding its own BPMN engine, identity provider, or monitoring infrastructure, it establishes clear contractual interfaces with such external services. This architectural choice preserves orchestration and mediation as its core responsibilities, while delegating execution, identity management, and observability to specialised and interchangeable components. In doing so, the framework avoids redundancy, fosters modularity, and ensures that each function is handled by tools best suited to its domain.

Surrounding the framework, in a hybrid zone overlapping both the system and context boundaries, are the Cross-Concerns. These include the Identity and Access Management (IAM) service, the BPMN engine, and the infrastructures for logging and monitoring. Their positioning reflects their dual nature: they can be replaced with enterprise solutions that are already established in a particular organizational context, or they can be instantiated internally by the framework. The IAM service ensures centralized and secure handling of identity and access, the BPMN engine is trusted with dependable workflow execution, and monitoring and logging offer insight into the behavior of the system. Since they can be provided internally or externally without changing the core orchestration logic of the framework, their designation as cross-concerns highlights that they are necessary but not inherent to it. In this manner, the

framework is prepared and promotes a more seamless integration in the overall IT architecture of a business that intends to use it.

Beyond these enabling services reside the External Systems, such as ERP suites, REST-based applications, or SAP backends (represented abstractly in Figure 7 as System A, System B, … N). These systems retain their architectural independence and are not reshaped by the framework. Instead, they remain governed by their own contracts and business logic, while the framework ensures that integration with them occurs in a standardized, policy-compliant, and transparent manner. Their autonomy is maintained by this division, but they also become a part of a well-coordinated ecosystem where their roles support more extensive business procedures without compromissing personal integrity.

The framework guarantees robustness, extensibility, and clarity of purpose by arranging responsibilities and boundaries in such a way. The actors supply the human input and governance that direct its operation; the external systems preserve autonomy while taking part in integrated flows; the framework itself orchestrates and mediates; and the cross-concerns allow execution, security, and visibility. This layered distribution prevents fragmentation, avoids duplication, and guarantees that integration is not incidental but a governed, explicit responsibility. It also underscores the extensibility of the architecture: new systems, engines, or even new actor roles can be introduced without altering the framework's core, reinforcing its suitability for dynamic and heterogeneous enterprise environments.

4.3 Framework Containers

While the context view establishes the outer boundary of the framework, the container view represents the first step into its inner workings, offering a structural perspective on how the system is built. In the C4 model, the container level is where the architecture begins to take tangible form, revealing the major (internal) building blocks that exist within the system boundary and showing how they interact to deliver the framework's core capabilities.

Each of these containers is more than an abstract concept; it is a deployable and independently scalable unit of functionality that serves a precise role in the broader design, as shown in Figure 8. Through assigning responsibilities in a way that keeps the system modular, maintainable, and growth-ready, they collectively establish the framework's operational skeleton.

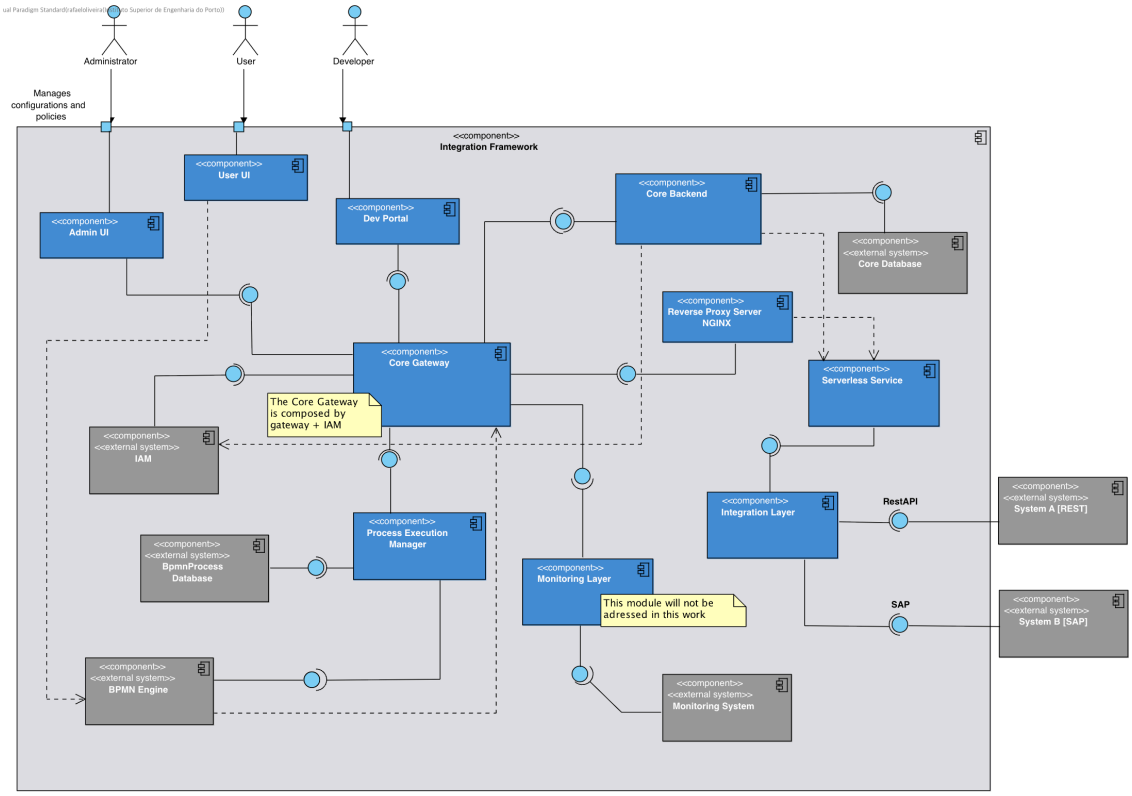


Figure 8 Container View

At the container level, the framework's design embodies both the consolidation of long-standing software engineering principles and the particular requirements of process-centric integration. Modern architectures are commonly structured around a clear separation between front-end and back-end layers. The front-end mediates interaction with human actors, rendering organizational tasks as accessible workflows and ensuring that engagement with the system remains intuitive and responsive. The back-end, in turn, concentrates the logic, persistence, and coordination functions that translate those interactions into consistent and verifiable outcomes, thereby upholding reliability, scalability, and conformity with governance rules. This separation of responsibilities, validated through decades of practice, reinforces modularity and facilitates the autonomous evolution of the different architectural layers.

Acknowledging the diversity of actors that interact with the framework—administrators, developers, and end users—it was considered essential to design dedicated user interfaces tailored to each profile. In accordance with the Non-Functional Requirements defined in (cf. Section 4.1.2), and specifically NFR-U01 on role-specific usability, three interfaces were developed to ensure actor-aligned interaction. The Admin UI was conceived to support administrators in governance tasks, offering dashboards for configuring engines, assigning policies, and inspecting service health. In parallel, the Dev Portal was introduced as the workspace for developers, enabling them to upload BPMN models, manage deployed processes, monitor logs, and oversee integration flows. Finally, the Tasklist UI was reserved for end users, who require a streamlined environment to claim and complete tasks generated by process

models. This tripartite distribution ensures that each actor engages with the framework through an interface proportionate to their responsibilities, thereby minimizing cognitive overhead and reinforcing the principle of separation of concerns.

Under the user-facing layers resides the Core Gateway, which combines an API gateway with an identity management system to act as the single authoritative entry point into the framework. Its function extends well beyond routing, as it enforces authentication and authorisation, standardises incoming requests, and guarantees that every interaction adheres to the identity and policy constraints defined upstream. By concentrating enforcement in a single location, designating the gateway as the sole entry point enhances security and unifies governance. Requests are sent to the Core Backend, the orchestration centre of the framework, after being verifie. Whereas the gateway remains confined to mediation and access control, the backend assumes responsibility for interpreting business-level instructions, coordinating dependencies across containers, and determining the execution order required to implement a process or integration. This intentional division follows best practices in distributed architectures, where separating orchestration logic from traffic mediation improves fault tolerance, scalability, and maintainability.

The Process Execution Manager (PEM) represents a pivotal container within the framework, functioning as the steward of BPMN process definitions. It is in charge of managing the storage, version control, deployment, and execution of process models throughout their whole lifecycle. The PEM offers a structured connection between the practical realm of workflow execution and the conceptual layer of business process modeling by preserving the raw XML specifications along with additional metadata like tags, timestamps, and operational status. It guarantees accurate instantiation, dependable execution, and efficient monitoring of process models by coordinating with the BPMN engine. While execution is purposefully left to the external engine, orchestration and lifecycle governance are still internalized, ensuring a clear division of duties and bolstering architectural coherence. This design exemplifies the framework's emphasis on modularity.

The Serverless Services module expands the framework into the domain of elastic, on-demand computation, complementing the previous containers. This module allows integrations to be implemented as distinct, event-driven functions that run independently and can be dynamically orchestrated when needed, as opposed to being encapsulated within monolithic applications. These functions are modular building blocks that are kept in a registry to make it easier for other processes to find and use them. The strategy prioritizes scalability, effective resource use, and fine-grained modularity, reflecting the broader trend in enterprise systems towards serverless paradigms. The Core Backend can minimize operational overhead and maximize responsiveness and adaptability to workload fluctuations by interacting with this container to provision only the functions that are strictly required for a particular workflow instance.

Equally essential is the Integration Layer**,** whose role is to insulate the framework from the heterogeneity of external systems. Whether the target is a REST API, a SOAP service, or an enterprise-grade platform such as SAP, the integration layer provides the abstraction necessary

to handle different protocols, message formats, and data semantics. In doing so, it ensures that the logic expressed in BPMN remains independent from technological contingencies, strengthening the framework's claim to interoperability and adaptability.

Finally, the architecture anticipates—but does not fully implement within this thesis—the Monitoring Layer[2], conceived as a transversal capability for observability. Its purpose is to provide administrators with visibility into the execution of processes and integrations by collecting logs, metrics, and traces. While its conceptual importance is acknowledged, the monitoring layer is treated in this work as an external dependency, to be satisfied either by integrating with existing enterprise monitoring stacks or by future expansion of the framework itself. This explicit delimitation respects the scope of the prototype, while leaving a clear path for extending operational maturity.

Figure 8 encapsulates the result of these design decisions. The diagram portrays the framework as an ecosystem of interacting containers, each carrying a defined responsibility and positioned to interact coherently with actors, cross-concern services, and external systems. What emerges is not an arbitrary collection of modules but a structured architecture that progresses logically from general principles—front-end/back-end separation, modularization of concerns—to specific components aligned with the objectives of business process–centric integration.

4.4 Core Gateway

Within the architectural composition of the framework, the Core Gateway emerges as a pivotal boundary component, entrusted with the responsibility of regulating and mediating all interactions between the external environment and the internal containers of the system. Its existence responds to a fundamental architectural principle: that complex, distributed systems cannot rely on ad hoc entry points or unregulated traffic flows, but must instead enforce a unified ingress through which all requests are channeled, authenticated, and normalized. The gateway is therefore not a peripheral utility, but a structural mechanism of governance, operating as the controlled access point that guarantees consistency, traceability, and security at the system's perimeter. Importantly, this component is not conceived as a gateway in isolation, but as a hybrid unit that integrates ingress mediation with identity and access

[2] The Monitoring Layer, although architecturally relevant, is not further addressed/detailed through this work since it is considered out of scope.

management (IAM), ensuring that traffic control and identity enforcement are inseparable at the architectural boundary.

From a conceptual standpoint, it serves as the first line of defence and control, ensuring that no interaction can traverse into the framework without being subject to inspection, validation, and alignment with the overarching governance policies. This strict separation of concerns ensures clarity in architectural roles: orchestration belongs to the backend and execution managers. By embedding IAM responsibilities directly into the gateway, the framework achieves a unified enforcement model, preventing the dispersion of authentication and authorisation across downstream containers and consolidating policy management at the perimeter.

Another defining feature of the Core Gateway at the architectural level is its lightweight and stateless design philosophy. It is conceived not as a locus of computation but as a structural checkpoint. Every request—whether originating from an administrator invoking configuration tasks, a developer deploying a new process, or an end user completing a human task—is required to enter the framework through this singular point. By doing so, the system achieves not only consistency of ingress but also centralised observability. All external interactions are recorded at the same boundary, enabling the establishment of uniform audit trails, usage metrics, and compliance verifications.

The architectural rationale for centralising ingress in the Core Gateway also lies in the need for scalability and reliability. Because the gateway deliberately avoids embedding orchestration logic, it remains comparatively simple to scale horizontally in response to fluctuations in request volumes. Its operational complexity is intentionally restrained, thereby reducing the risk of cascading failures and ensuring that the enforcement boundary remains dependable under variable workloads. This approach transforms the gateway into a predictable, stable anchor point for the system: it is the part of the framework least subject to functional evolution but most critical for sustaining its operational resilience.

Figure 9 depicts the responsibility flow of the Core Gateway within the framework. All incoming requests traverse a single ingress, where they are authenticated, authorised, and subjected to centralised policy enforcement before being routed to internal services. The diagram shows how minimal identity is propagated downstream and how observability hooks—such as logging and tracing—are applied at the boundary. By visually mapping these responsibilities, the figure reinforces the gateway's synergy with IAM, as the unified enforcement point of the architecture, ensuring that governance, security, and traffic control are consistently applied across all interactions.

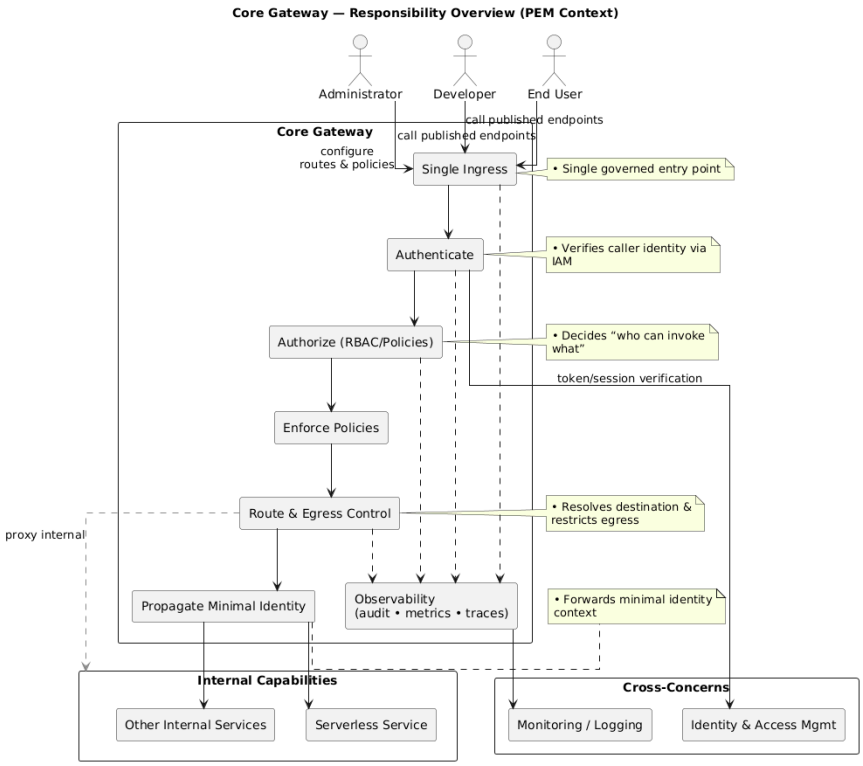


Figure 9 Core Gateway Responsabilities Overview

Ultimately, the Core Gateway represents the architectural manifestation of the principle that no request should enter the framework unmediated or unchecked. It is intentionally kept thin, stable, and scalable, precisely so that the orchestration logic and business intelligence can remain concentrated in the containers designed for those purposes. What remains is the essence of a gateway tightly coupled with IAM: a structural safeguard that validates and regulates ingress, normalises interactions, and guarantees that the framework's core is shielded from ungoverned access. The specific mechanisms by which these responsibilities are implemented—such as route-level policies, identity propagation, or the technical integration with an IAM provider—are not considered in this section, since they belong to the prototyping dimension of this dissertation and will be discussed in detail in Chapter 5 (cf. Section 5.1).

4.5 Core Backend

Guided by this last paradigm, the Core Backend serves as the central orchestration brain of the proposed framework. It embodies the business logic that coordinates interactions across heterogeneous systems and containers. Its primary responsibility lies in interpreting high-level orchestration commands and distributing them to the appropriate components, such as delegating BPMN lifecycle management to the Process Execution Manager or routing integration calls toward the Serverless Service layer. The Core Backend not only makes decisions but also integrates functions such as centralized error handling, trigger coordination, configuration management, and authorization enforcement. The architecture of this container

is monolithic, containing controllers, managers, and auxiliary services in a single deployable unit. Coherence and ease of deployment are given precedence over distribution in such a structure, which reflects both a technological choice and an architectural intent. The runtime perspective is still that of a single service, but the codebase internally maintains a modular structure that divides duties. In this balance, orchestration logic is concentrated in a single locus, while specialized execution tasks are routed to specific containers, such as the Serverless Service layer for integration execution or the Process Execution Manager for BPMN lifecycle management.

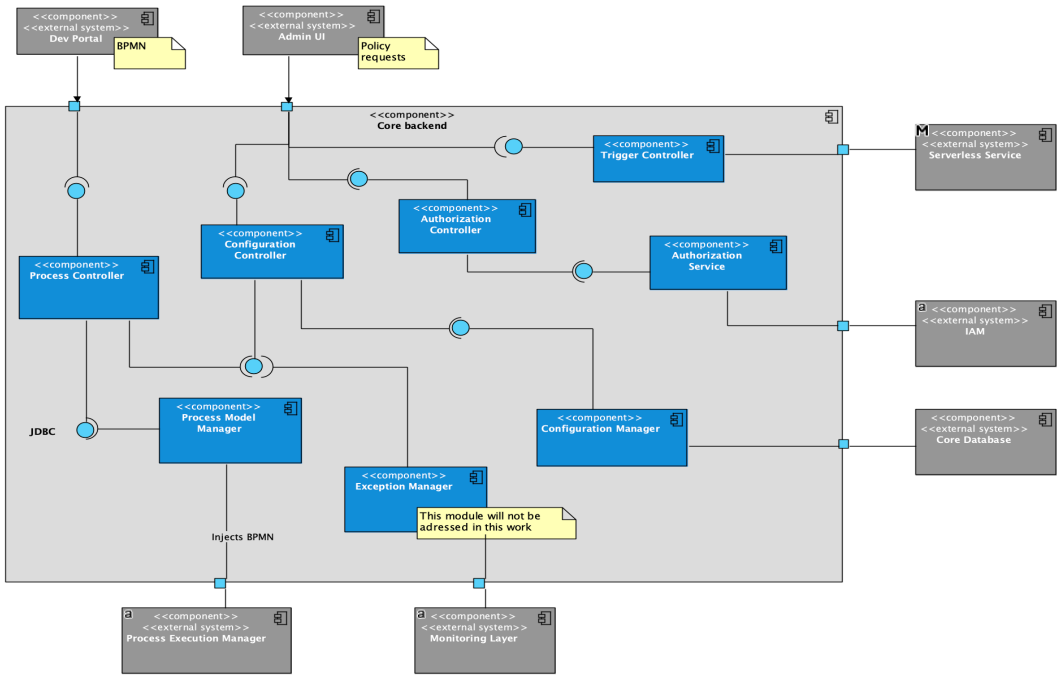


Figure 10 Core Backend - Component View of Orchestration Brain

At the core of this orchestration lies the Process Controller, which acts as the primary entry point for BPMN-related operations arriving from the Dev Portal or the Admin UI. When a process model is uploaded, a new instance is started, or the state of an ongoing process is queried, the Process Controller interprets the request and initiates the necessary actions. This controller delegates model-specific responsibilities to the Process Model Manager, which has full ownership of the lifecycle of BPMN definitions. It handles their validation, injection, and internal registration before passing them to the Process Execution Manager (an external system) for execution. This separation ensures that orchestration logic is isolated from the technicalities of BPMN runtime handling.

Configuration management follows a parallel path. Incoming configuration requests are directed to the Configuration Controller, which manages administrative commands such as engine selection, integration endpoint registration, and global system settings. This component works closely with the Configuration Manager, which is responsible for persisting configurations and ensuring their consistency across system restarts. All configuration changes are guaranteed to be durable and auditable because the persistence itself is assigned to the Core Database.

This careful layering—controller, manager, persistence—reflects a purposeful design to improve maintainability by separating decision logic from data storage.

An additional layer of accountability is carried by the authorization infrastructure of the backend. In this instance, the Authorization Controller acts as the entry point for requests that require explicit permission checks, and the Authorization Service manages the logic that evaluates those requests. These checks could entail looking up policy rules established in the higher governance layer or submitting a query to the external IAM system. Because of this dual structure, the framework can maintain security and modularity by enforcing fine-grained access rules without directly integrating identity logic into the backend's core.

The Trigger Controller assumes a narrower but relevant responsibility within the Core Backend. Rather than actively invoking serverless functions, its primary task is to query and monitor the set of available triggers associated with deployed integrations. In reality, these triggers are produced automatically when service tasks are bound to HTTP endpoints that the serverless runtime exposes during the BPMN design and deployment stage. In order to facilitate orchestration and governance, the Trigger Controller gathers metadata about these endpoints, including their availability, assigned paths, and related integration modules.The component guarantees that orchestration logic stays transparent by limiting its function to discovery and monitoring, thereby avoiding the duplication of execution tasks that are the responsibility of the serverless runtime or the BPMN Engine. The idea that the backend should control and monitor while workflow definitions and runtime infrastructure handle actually calling functions is strengthened by this division.

Resilience is ensured by the Exception Manager, which centralizes error handling within the backend. If a BPMN deployment fails validation, an integration call produces unexpected results, or a configuration change cannot be applied, the Exception Manager ensures that such events are captured, logged, and, whenever possible, recovered from gracefully. Its presence reduces the risk of silent failures, thereby safeguarding the predictability and reliability of the backend. Although the Exception Manager is structurally present in the architecture, the detailed implementation of advanced recovery strategies remains outside the scope of this dissertation.

Finally, the Monitoring Layer is also shown as an external system linked to the backend. Although the backend produces logs, metrics, and status events for monitoring, the detailed implementation of this monitoring infrastructure and its visualization dashboards is beyond the scope of the present thesis. It is, however, acknowledged as an essential cross-concern for any future industrial deployment of the framework, but, as mentioned before, it is not addressed in this dissertation.

All of these components work together in a carefully orchestrated chain of dependencies. Controllers (Process, Configuration, Authorization, Trigger) act as entry points, receiving requests from users or external systems. They delegate to managers (Process Model Manager, Configuration Manager) or services (Authorization Service) that contain the actual execution logic. These, in turn, interact with external systems, including the Process Execution Manager for BPMN workflows, the IAM for identity management, the Core Database for persistence, and

the Serverless Services module for integration functions. This arrangement prevents duplication of responsibilities while ensuring a coherent end-to-end flow of execution.

4.6 Process Execution Manager

The Process Execution Manager is the framework's dedicated BPMN orchestration and lifecycle management hub. While the Core Backend provides the logic that determines when and why a process should run, the Process Execution Manager is responsible for deploying, validating, and executing that process. It serves as the bridge between the framework's high-level orchestration and the workflow engines that perform the actual process execution. From a structural perspective, the Process Execution Manager is composed of a set of components that mirror the stages of the BPMN lifecycle — from receiving and validating models, to dispatching commands, to interacting with engines via adapters. Figure 11 illustrates his architecture.

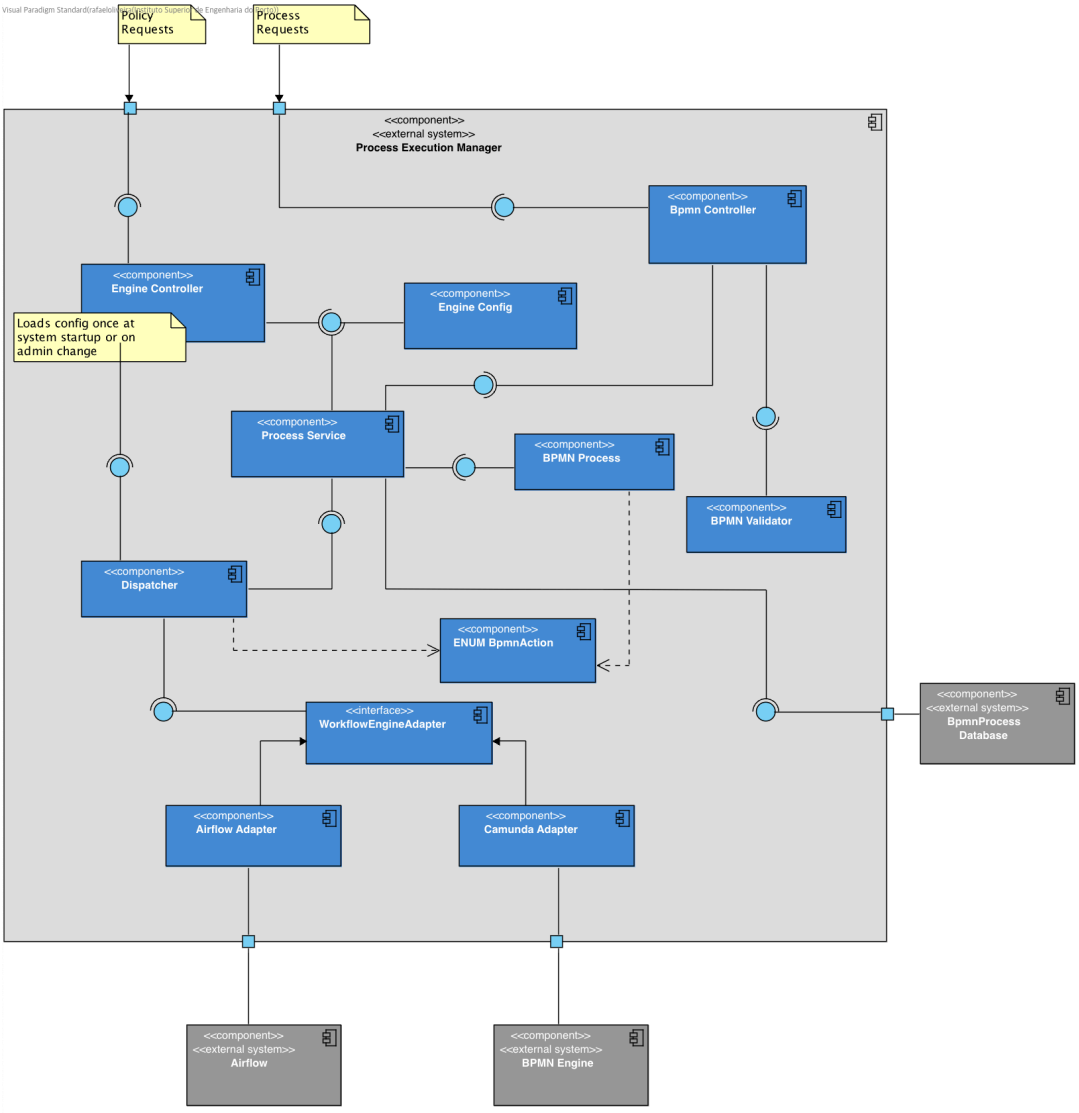


Figure 11 Process Execution Manager - Component View of BPMN Lifecycle Management

At the entry point is the Engine Controller, which plays a crucial role in system initialization and configuration. When the framework starts, or whenever an administrator changes engine settings, the Engine Controller loads engine configurations and ensures that the Process Execution Manager knows which engines are active, how they are reached, and what default behaviors should apply. This configuration is stored and maintained through the Engine Config component, which acts as a reference for all other modules whenever they need to interact with a BPMN engine.

The Bpmn Controller handles incoming BPMN-specific commands. Whether the request comes from the Core Backend, this controller interprets the command and orchestrates the appropriate internal sequence — for instance, starting a new BPMN model deployment or initiating a process instance. It closely collaborates with the Process Service, which manages the core set of operations that can be performed on a BPMN model or process.

Every BPMN model undergoes validation by the BPMN Validator before it can be deployed. This component ensures both syntactic validity (that the XML structure is correct) and semantic validity (that the process definition is logically consistent and executable). Only after validation does the model transition into the BPMN Process component, where it is formally registered into the system and prepared for execution.

The Dispatcher is the Process Execution Manager's internal command router. Once a process action (deploy, start, suspend, or other) is confirmed, the Dispatcher determines the correct engine path and sends the instruction to the corresponding workflow engine. To ensure standardization across different engines, every command is expressed through the BpmnAction enumeration (ENUM BpmnAction), which normalizes operations into a fixed set of actions (for example, DEPLOY, START, STOP, DELETE). This prevents the system from being locked into engine-specific verbs and guarantees that all engine adapters speak a common internal language.

A central component that makes this flexibility possible is the WorkflowEngineAdapter. Instead of the Process Execution Manager being tightly coupled to one workflow engine, this adapter serves as an abstraction layer, delegating commands to the correct concrete implementation. Below it, engine-specific adapters, such as the Camunda Adapter and the Airflow Adapter, translate generic BPMN actions into the exact API calls and protocols required by those engines. If another engine were to be added in the future, it would simply require the development of a

new adapter, without touching the rest of the framework. This interface approach is represented in the Figure 12.

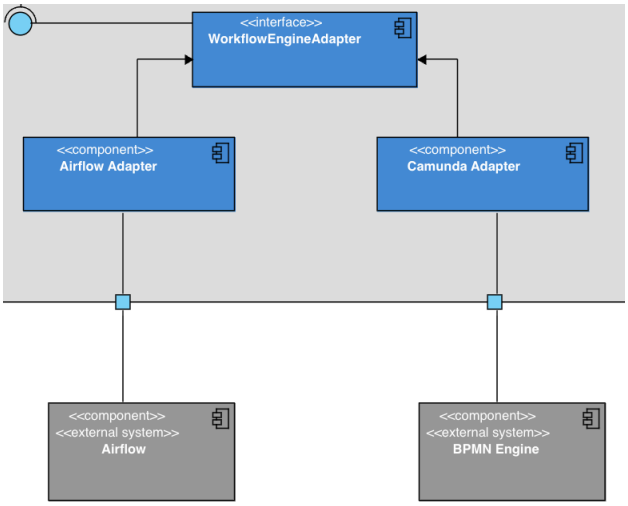


Figure 12 Workflow Engine Adapter Interface

Within the architectural role design, the Process Execution Manager (PEM) is positioned as the guardian of process identity and governance, not merely the dispatcher of lifecycle commands. Beyond receiving, validating, and deploying models, the PEM maintains a canonical record for each process that includes user-defined metadata—such as tags, categories, and descriptive attributes—captured at registration time and mutable under controlled policies. This metadata supports portfolio-level classification (e.g., “finance”, “customer-onboarding”, “experimental”) and facilitates semantic discovery (e.g., filtering and search across extensive catalogues).The framework allows stakeholders to curate and reason about the process landscape without tying those concerns into engine-specific constructs by elevating such annotations to first-class architectural elements, which separate descriptive concerns from execution mechanics.

In summary, the PEM ensures the integrity of the BPMN lifecycle by combining extensive validation and deployment with process metadata enrichment. Therefore, it turns process definitions from temporary technical artifacts into assets that can be controlled and discovered. The PEM offers a logical way for administrators to enforce policies, developers to organize deployments, and end users to engage with processes in a semantically meaningful manner by combining descriptive attributes and execution responsibilities into a single locus. This dual role—operational executor on one side, and architectural steward of process identity on the other—highlights its centrality to the framework. The subsequent chapter (Chapter 5) examines in detail the components that fulfill these responsibilities, from controllers and validators to

dispatchers and adapters, thereby breaking down the architectural design into its constituent mechanisms.

4.7 Serverless Service

The Serverless Service container provides the execution environment in which integrations are deployed as independent, elastic, and event-driven functions. While integrations are designed and prepared within the Integration Layer, they require a runtime that can manage their lifecycle at scale, ensuring that each integration can be activated on demand, scaled according to workload, and isolated from other concurrent executions. In this way, integration modules are converted into callable services that function reliably and under control by the Serverless Service, which serves as a specialized execution substrate that separates integration design from operational realization. In the context of this architectural design, the description focuses solely on the responsibilities and requirements attributed to this container, as the intention is to implement them through the adoption of an existing tool capable of fulfilling these roles.

This container's compatibility with a proxy component is an essential feature that ensures different functions can operate together within a single, manageable entry point. A fragmented and operationally precarious landscape would result if each integration exposed itself directly through its own listener and port, in the absence of such mediation. Functions are hidden behind a unified routing layer that routes incoming requests to the relevant integration by combining the serverless runtime with a proxy. This design routes all invocations through a uniform endpoint structure, doing away with the requirement that each function maintain its own public interface. As a result, the system avoids inconsistent service exposure, reduces the complexity of port management, and offers a central mechanism that the Gateway can use to control requests in the future.

This configuration improves resilience and elasticity in addition to simplicity. As demand changes, the serverless runtime ensures that operations scale horizontally, and the proxy controls request multiplexing across multiple instances to maintain availability and balance the load. Because each integration is a separate function, it operates independently, lowering the possibility that malfunctions will spread to other unrelated services. Additionally, this architectural layering enables the addition of more governance features at the proxy boundary, such as logging, request normalization, and health checking, which improves the system's overall observability.

From an architectural perspective, the Serverless Service cannot be regarded as an isolated component but as an integral element in the broader flow that links integration design, execution, and governance. Within this sequence, the Integration Layer generates standardized integrations that are subsequently transformed into deployable functions by the Serverless Service. These functions are then presented through a uniform interface managed by the proxy, which forwards them to the Core Gateway. At this boundary, authentication and authorisation mechanisms are applied, ensuring that every invocation of a function is subject to consistent

enforcement and cannot occur without proper validation.In this way, the container ensures not only execution elasticity but also the seamless integration of serverless functions into the framework's governance model.

It should also be noted that this section refrains from detailing specific serverless platforms or proxy technologies, as these are addressed in the prototyping stage of this dissertation, Chapter 5 (cf. Section 5.4). Likewise, a comparative study of serverless platforms—covering, for example, Nuclio, Fission, and other alternatives—is provided in Appendix E, situating the architectural choice within a broader technological context.

4.8 Integration Layer

The Integration Layer constitutes the architectural container responsible for preparing, managing, and governing the environment in which enterprise integrations are conceived and executed. Its ability to offer a formal, regulated environment where integrations are converted from abstract requirements into cohesive architectural artifacts is what makes it fundamental, not the low-level mechanics of implementation. In order to guarantee that integrations are consistently designed, deployed, and managed under explicit governance rules rather than developing as ad hoc connectors, this container connects the framework's orchestration logic with the diverse systems that make up the enterprise ecosystem.

The Integration Layer ensures stability and consistency across heterogeneous technologies by creating a controlled environment where interactions with external systems are defined in a validated and standardized way, as shown in Figure 13. Integrations can be versioned, maintained, and reused without compromise architectural coherence because they are modeled as governed modules that adhere to the framework's lifecycle rather than being handled as ad hoc scripts.

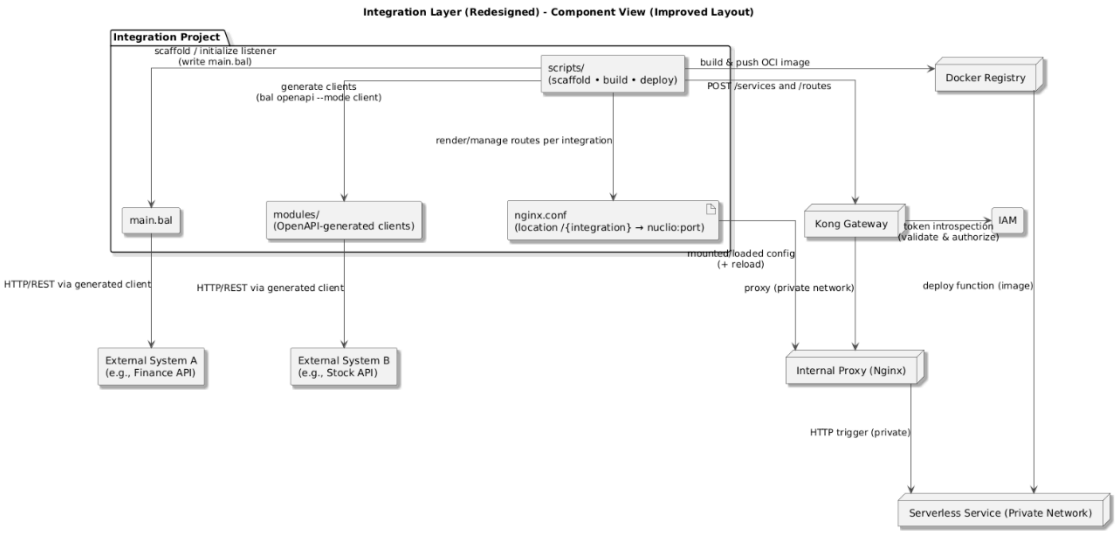


Figure 13 Integration Layer - Architecture

Equally central to the architectural role of this container is its responsibility for managing the path from design to deployment. The Integration Layer is tasked with defining how integrations are structured as self-contained modules, prepared for execution in the Serverless Services container, and registered through standardized routing mechanisms. A crucial architectural responsibility lies in its ability to inject integration routes into the gateway. This action should not be mistaken for authorization, which remains outside the scope of this container; rather, it is a mechanism for registering integrations within the routing infrastructure, guaranteeing that deployed services are discoverable and callable through controlled, auditable entry points. By doing so, the framework ensures that orchestration logic interacts with integrations through standardized endpoints, while the enforcement of access policies is delegated to other containers more directly concerned with identity and authorization.

The Integration Layer assumes a dual mandate within the framework. On the one hand, it provides an organized setting where integration development is raised to the level of architectural design, guaranteeing that procedures stay consistent and repeatable. On the other hand, it ensures that integrations follow a standardized and controlled route to deployment and routing after they are conceived. In performing this function, the container serves as a unifying force across the enterprise landscape, incorporating external platforms, modern APIs, and legacy applications into a single architectural model. By avoiding direct dependence on the idiosyncrasies of individual systems, this abstraction allows the orchestration layer to function through consistent and reliable interfaces.

By emphasizing architectural discipline over implementation detail, the Integration Layer establishes itself as a foundational element of the framework. It assures stakeholders that integrations are not improvised but systematically designed, registered, and managed under governance structures that promote modularity, maintainability, and extensibility. The technical mechanisms by which these responsibilities are realized—such as code scaffolding, containerization, or deployment pipelines—are deliberately deferred to Chapter 5 (cf. Section 5.6), where the prototyping of the framework provides a concrete account of the implementation process.

4.9 BPMN Engine

The BPMN Engine represents the second execution environment within the proposed framework, complementing the Serverless Service container by addressing orchestration rather than discrete integration logic. While serverless functions ensure the execution of fine-grained integration tasks, the BPMN Engine governs the end-to-end flow of business processes, enforcing BPMN semantics and coordinating both human and automated activities across heterogeneous systems. This duality provides balance: one environment specializes in elasticity and event-driven execution, while the other ensures process integrity and lifecycle continuity.

From an architectural perspective, the BPMN Engine is purposefully isolated from the framework's orchestration logic, functioning as a modular and interchangeable container.

Instead of any vendor-specific implementation, the focus is on abstraction. The Process Execution Manager (PEM), which serves as the mediation layer, converts orchestration directives— such as deploying, initiating, or suspending workflows —into standard commands before sending them to the engine. By doing this, the framework maintains a contract-based relationship where process models are validated and controlled regardless of the runtime chosen, avoiding reliance on a single execution technology. Further detail on this mediation is provided in Chapter 5 (cf. Section 5.2), where the prototype demonstrates how these principles are applied to configure and operate processes across different engines.

Another architectural responsibility of the BPMN Engine is its ability to integrate human participation seamlessly into process execution. Whereas service tasks invoke automated calls to integrations, human tasks require suspension, allocation, and subsequent resumption once input has been provided. The BPMN Engine, in this sense, guarantees that workflows can capture user interaction as a native part of their lifecycle, preserving traceability and synchronization between automated and human-driven steps. The implementation of these interaction mechanisms is documented in the prototyping phase, where the Tasklist interface illustrates the binding between process definitions and user-facing forms.

Although the design of the BPMN Engine is intentionally agnostic, the framework acknowledges the importance of providing a reference implementation to validate its abstractions. From an architectural perspective, however, the present discussion deliberately refrains from addressing implementation details and focuses exclusively on the role of the engine as an execution container responsible for coordinating processes. The practical configurations through which this execution logic is realized are examined in Chapter 5 (cf. Section 5.2**)**, where the prototyping stage illustrates how an engine can be operationalized within the framework. Complementarily, Appendix D offers a detailed exposition of the internal architecture of a representative engine. At the same time, Appendix C extends this perspective by situating the chosen reference technology within the broader state of the art.

By structuring the exposition in this way, the BPMN Engine is not portrayed as a static dependency but as an architectural contract within the framework. It encapsulates the responsibilities of process lifecycle governance, mediates between orchestration logic and execution runtimes, and guarantees that workflows—whether automated or human-driven—unfold in a manner consistent with enterprise requirements. Its modular and replaceable nature aligns with the overall principle of openness that guides the framework, enabling future engines to be incorporated with minimal disruption while preserving the same guarantees of consistency and traceability.

4.10 User Interfaces

While the deeper layers of the framework are concerned with orchestration, integration, and execution, their effectiveness is only realized through the user interfaces that channel these capabilities to human actors. The architectural design of the UIs reflects not a mere graphical veneer, but an explicit separation of responsibilities aligned with the three principal roles identified in the framework: administrators, developers, and end users. Each group is provided with a dedicated interface—Admin UI, Developer Portal, and User UI (Tasklist)—whose structure and modularity embody the principle of role-driven design. Figure 14 and Figure 15 present the internal organization of the Admin UI and Developer Portal, respectively, while the Tasklist, being a native component of the engine, is considered an integrated external UI.

4.10.1 Admin UI

The Admin UI is conceived as the architectural cockpit of governance, providing administrators with a unified environment through which they can oversee and configure the framework. Its internal structure, illustrated in Figure 14, is organized around a central entry point (App.js), which is initialized by index.js and from which the various modules are orchestrated. Rather than functioning as isolated pages, the views are arranged as complementary governance instruments.

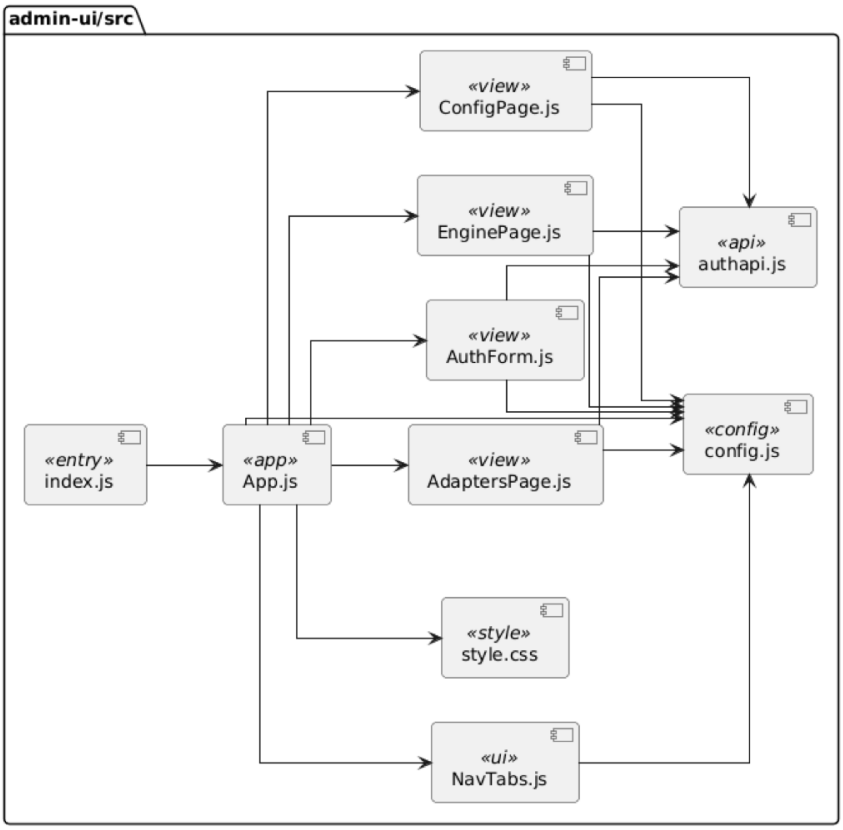


Figure 14 Admin UI Architecture

The configuration view (ConfigPage.js) allows system-wide parameters to be defined and adapted, while the engine view (EnginePage.js) enables the selection and fine-tuning of execution environments. Authentication is handled through a dedicated interface (AuthForm.js), which ensures that identity-related interactions remain consistent with the framework's security principles. The management of adaptors connecting the framework to external services is concentrated in a further view (AdaptersPage.js), thereby separating integration governance from other administrative tasks.

These views rely on supporting modules such as authapi.js and config.js, which centralize calls to backend services and abstract configuration details, ensuring that sensitive operations are consistently mediated. Visual coherence and navigability are reinforced by NavTabs.js, which provides structured access to the different views, and style.css, which guarantees a uniform presentation layer across the interface. Taken together, this modular arrangement illustrates the principle of separation of concerns at the level of user interaction: governance activities are distributed across specialized modules but integrated into a single cockpit, allowing administrators to intervene quickly in one area without destabilizing the broader system. Such a design is not only conducive to maintainability but also ensures that the framework can accommodate governance requirements that evolve over time while preserving structural stability.

4.10.2 Developer Portal

The Developer Portal functions as the creative and operational workspace of the framework, providing an environment where developers can design, deploy, and manage processes in a structured yet flexible manner. Its architecture, illustrated in Figure 15, mirrors the design pattern established in the Admin UI: initialization begins with index.js, which delegates to App.js, thereby creating a consistent entry structure across all interfaces. From this core, a set of specialized views branch out, each serving a distinct role within the lifecycle of process development. The process list view (ProcessList.js) provides developers with a catalog of available processes, offering not merely a static listing but an interactive environment where processes can be inspected, selected, and organized. Closely tied to this functionality, the visualization and submission of process models are handled by BpmnViewer.js and BpmnUploader.js, which together ensure that models can be both validated visually and uploaded seamlessly into the system. The management of operational triggers and runtime feedback is achieved through HttpTriggerList.js and MonitorLogs.js, enabling developers to link processes to event-driven mechanisms while simultaneously maintaining oversight of system behavior through accessible logs. Supporting these views are service modules such as processService.js and triggerService.js, which encapsulate the details of backend communication, thereby separating the complexity of API interaction from the responsibilities of the user interface. Configuration concerns are centralized in config.js, ensuring uniformity across modules, while setupProxy.js provides a means of isolating proxy rules during development, allowing network dependencies to be managed without entangling them in application logic. This layered design illustrates a deliberate architectural intent: to provide a

developer-facing environment that is at once self-contained and extensible, mapping each responsibility to a distinct module while ensuring that interconnections remain strictly defined. In this way, the Developer Portal transcends the role of a mere collection of tools, functioning instead as a coherent architectural environment that aligns with the full lifecycle of integration development.

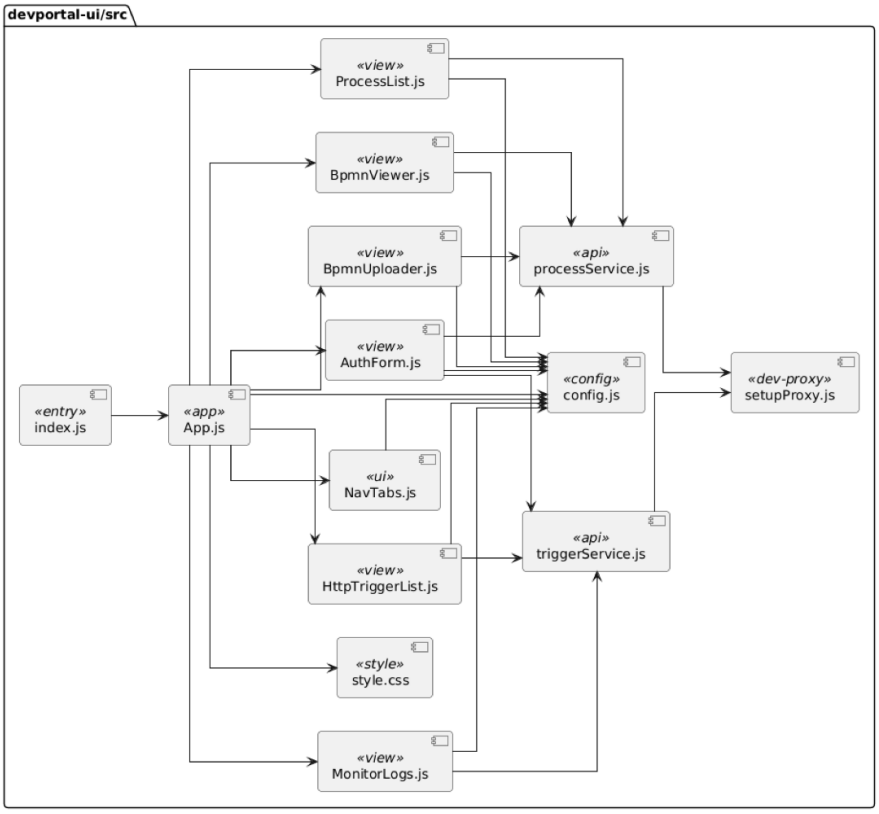


Figure 15 Devportal UI Architecture

4.10.3 User UI (Tasklist)

The User UI provides the dedicated interface through which end users interact with workflows by performing human tasks. Whenever a process reaches an activity that cannot be automated, execution is suspended, and a corresponding task is made visible to the designated user group. Through this interface, tasks can be claimed, completed, and submitted in a structured manner, ensuring that user input becomes an integral part of the orchestration flow. The User UI thus embodies the human-in-the-loop principle, enabling business processes to combine automated logic with human judgment, while maintaining traceability and governance across the execution lifecycle.

Taken together, the three interfaces illustrate the framework's role-driven approach to interaction. The Admin UI enables governance and policy control, the Developer Portal offers an environment for design and deployment, and the User UI integrates human participation into automated workflows. Their separation of concerns ensures that each actor—

administrator, developer, or end-user—engages with the framework through a tailored environment aligned with their respective responsibilities. This deliberate structuring reinforces modularity, reduces cognitive overhead, and upholds extensibility. It must also be emphasized that this section is limited to the architectural design of the interfaces, while their concrete implementation and prototyping, including their integration with backend services and identity management, will be examined in subsequent chapters of this dissertation.

4.11 Specialization: Agnostic Integration Approach

A distinguishing characteristic of the proposed framework is its deliberate agnosticism toward the technologies used for process execution, integration development, and serverless deployment. While many integration platforms couple themselves tightly to a single workflow engine or a single programming model, this framework is explicitly designed to remain open and extensible, allowing multiple alternative technologies to coexist under the same architectural boundary.

This specialization manifests in three principal areas. First, at the workflow execution level, the framework does not prescribe a single BPMN engine. Engines such as Camunda, Airflow, or Zeebe can be connected and orchestrated interchangeably through the Process Execution Manager, which normalizes operations into a standard action model (deploy, start, stop). This ensures that organizations are not constrained by one vendor or runtime, but can choose the execution engine that best aligns with their existing environment or future requirements.

Second, within the integration layer, the framework supports the development of integration services in Ballerina but does not restrict integrations to this language alone. Ballerina is used in this dissertation as the reference implementation due to its strong native support for network protocols and data transformations. However, the framework permits the deployment of integration modules developed in other languages or toolchains, provided they can be packaged and registered as callable services. This language-agnostic philosophy reflects the heterogeneous reality of enterprise systems, where developers often need to leverage different stacks to connect with diverse external platforms.

Third, at the serverless execution layer, the framework avoids dependency on a single function platform. While Nuclio is employed in this work to deploy and run integration functions due to its advanced observability and high-performance runtime, alternative platforms such as Fission or Knative can also be integrated. The framework treats these platforms as interchangeable function runtimes: each must provide an HTTP-accessible execution environment, but the choice of runtime remains flexible.

By adopting this agnostic stance, the framework gains several advantages. It avoids vendor lock-in, ensuring that future migrations or technology substitutions do not require a redesign of the framework itself. It offers adaptability to diverse organizational contexts, where different engines or platforms may already be in place. Finally, it future-proofs the architecture, as new

engines, languages, or serverless systems can be adopted incrementally by implementing adapters at the appropriate boundary.

This architectural perspective is not confined to conceptual abstraction alone. As will be demonstrated in Chapter 5, the prototype embodies this agnostic philosophy in practice, showing how the framework can be instantiated with concrete technologies while preserving the capacity for substitution and extensibility.

5 Prototype

The architectural design presented in Chapter 4 established the conceptual foundations of the integration framework, defining its boundaries, responsibilities, and internal structure across containers and components. However, an architectural blueprint, while necessary, does not yet constitute a functioning system. The purpose of this chapter is therefore to move from design to instantiation, showing how the abstract responsibilities previously described are materialized in concrete technological choices and integrated into a working prototype.

Moving from design to prototype entails assembling the individual components into a coherent whole. What was previously described in terms of roles and responsibilities is now translated into executable modules, deployed services, and working interfaces. In this process, architecture and implementation are brought into alignment: the abstractions introduced earlier are materialized in software artifacts that can be configured, executed, and evaluated.

This chapter also discusses the difficulties and challenges faced during implementation and the methodological decisions that were required to address them. Certain internal details of third-party systems are deferred to appendices, while the focus here is placed on how the framework integrates these systems into a functioning prototype. In this way, this chapter marks the transition from design to realization, preparing the ground for the following chapter, where the framework's applicability is validated in a real-world scenario.

5.1 Core Gateway: Centric Access Governance

Within the prototype, the Core Gateway assumes the role of the controlled entry point through which all interactions with the framework must pass. It represents the boundary at which requests originating from administrators, developers, or end users are admitted, inspected, and prepared before being forwarded to the orchestration layers that reside internally. Whereas in the architectural design of Chapter 4 (cf. Section 4.4) the gateway was described in abstract terms as a governance mechanism responsible for mediating external access, in the prototype this role was concretely instantiated through Kong, a technology chosen for its maturity, extensibility, and ability to enforce policies consistently at the system's perimeter. Here, emphasis is placed on how authentication and authorization mechanisms were operationalized in practice, demonstrating how the conceptual design was transformed into an enforceable security perimeter within the prototype.

At its essence, The Core Gateway is conceived not as a simple reverse proxy or infrastructural convenience but as a boundary layer that unifies gateway services with identity and access management (IAM). All traffic—whether from administrators, developers, or end users—is funneled through this point, where policies, tokens, and identities are consolidated and verified before requests proceed to the orchestration logic of the Core Backend or to the lifecycle

operations of the Process Execution Manager. This arrangement was realised by coupling the gateway with the IAM provider, and the technology chosen for this purpose was Keycloak, which ensures that identity remains authoritative while the gateway enforces fine-grained runtime policies. The concrete implementation of this design, including the development of a custom introspection plugin, is discussed in the following subsections (cf. Section 5.1.2). At this stage, the prototype confirms the architectural decision to keep the gateway lightweight in orchestration responsibilities while strategically positioning it as the enforcement point for all identity-driven constraints.

The relevance of this integration becomes most evident in the practical scenario developed in Chapter 6, where the Core Gateway was exercised as the single enforcement perimeter of the framework. Within this setting, its role extended beyond abstract governance to the concrete regulation of requests, authentication, and authorisation in real workflows. The scenario also surfaced subtle but critical runtime challenges, such as token issuer alignment across components, which demanded diagnostic testing and corrective measures. These operational findings confirmed the architectural rationale for concentrating identity enforcement at the gateway, establishing it as both the point of trust and the guarantor of consistency throughout the framework.

5.1.1 Authentication and Authorization via Introspection Plugin

In the context of integrating heterogeneous systems within a unified framework, it became imperative to implement a centralized, flexible, and secure mechanism for managing authentication and authorization. Although Kong provides a native introspection plugin, this feature is restricted to the enterprise edition and, therefore, incompatible with the open-source foundation of this project. To address this constraint, the integration of Keycloak with Kong Gateway was established manually, enabling fine-grained access control at the level of individual API routes. By delegating token validation and role-based authorization to the gateway, this approach eliminates the need to embed access control logic within backend applications—such as those developed in Spring Boot—thereby promoting a clean separation of concerns, improved maintainability, and stronger governance over system access policies.

The implementation began with the configuration of Keycloak as the identity provider, within which a dedicated realm was created to manage authentication across services. Within this realm—named `core-gateway-realm`—both clients and roles were defined in alignment with the required access policies. The roles were designed to reflect specific permissions associated with various system functionalities or services, laying the groundwork for granular access control.

To enforce these permissions at the gateway level, the custom Kong plugin was developed using Lua. This plugin is executed during the access phase of each request and is responsible for validating the bearer token supplied by the client. Upon intercepting the request, the plugin extracts the token from the `Authorization` header and performs an introspection call to

Keycloak, using a client ID and secret for authentication. The introspection endpoint validates the token and returns a JSON payload containing the token's status and any associated roles.

If the token is deemed active, the plugin evaluates whether the roles included in the token match those permitted for the route in question. This authorization logic ensures that access is granted only to users with appropriate permissions, and that it is enforced before any request reaches the underlying service. If the token lacks any of the authorized roles for that route, the request is denied with a 401 Unauthorized response. This design makes it possible to control access on a per-route basis, without modifying the services behind the gateway.

To streamline the administration of these permissions, an auxiliary interface was developed. This interface represented in Figure 16, enables an administrator to log into Keycloak using a valid user account, retrieve an access token, and decode it to extract the available roles assigned to the client. The application then fetches the list of routes exposed by Kong for a given service and displays them in a user-friendly interface. For each route, the administrator can assign or revoke roles by selecting from the decoded list, effectively updating the associated plugin configuration dynamically.

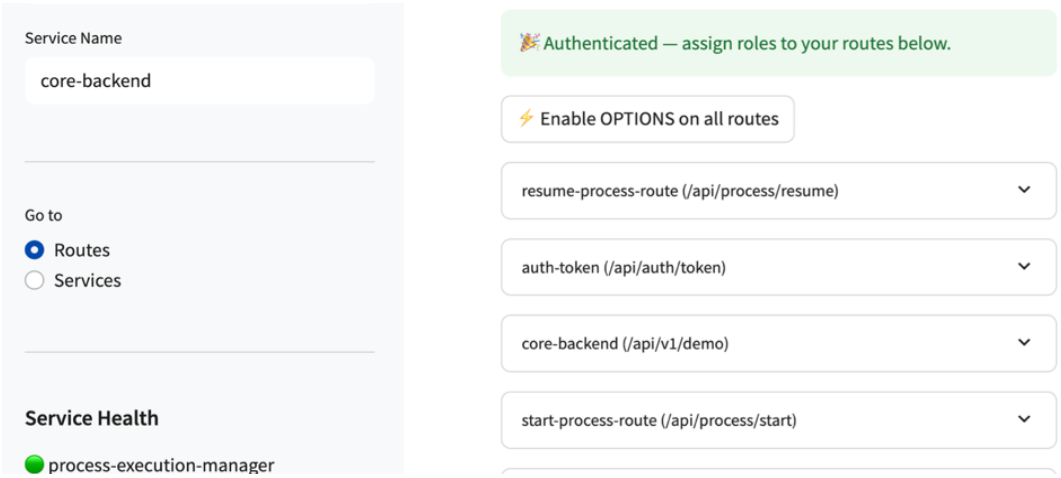


Figure 16 Authorization Manager User Interface

This integration not only centralizes the management of authentication and authorization but also introduces a scalable model for securing services across the platform. By decoupling authorization from the services themselves and encapsulating it at the gateway level, the system achieves greater modularity and adherence to the principles of least privilege. Additionally, the ability to modify route-level access control via a dedicated UI enhances operational efficiency and reduces the risk of misconfiguration.

5.1.2 Authorization Methodology: Introspection

The chosen methodology for authorization in this framework is based on token introspection as mentioned in the previous subsection, an approach where the gateway does not rely solely on the integrity of the presented bearer token but instead consults the identity provider directly. In practice, this means that every protected request arriving at Kong Gateway undergoes validation against Keycloak before being forwarded to any upstream service. This guarantees that access decisions always reflect Keycloak's current state, ensuring that revocations, expirations, or role modifications take effect without delay.The sequence diagram (Figure 17) illustrates the runtime execution of this approach.

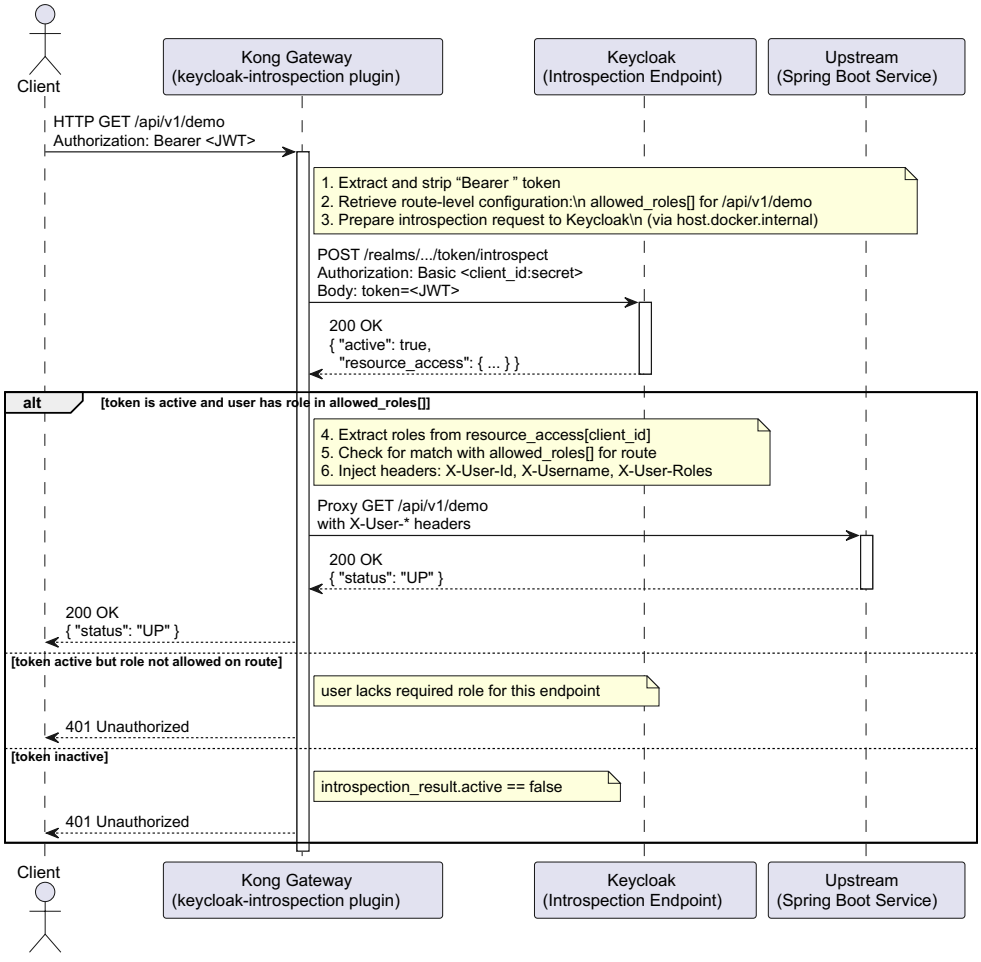


Figure 17 Introspection Plugin Workflow

When a request reaches Kong, it carries an `Authorization: Bearer …` header containing the opaque token issued by Keycloak. At this point, the custom introspection plugin intercepts the request during the access phase. The plugin first extracts the raw token from the header and retrieves the route-level policy defined for the requested endpoint. This policy specifies the list of roles permitted to access that resource. With both elements at hand, the plugin prepares an HTTP call to Keycloak's introspection endpoint, located within the protected network.

Keycloak receives the opaque token and validates its state. If the token is active, it returns a JSON payload containing metadata, including role assignments. The plugin then resolves the relevant roles from the claims, either from `realm_access.roles` (global roles) or from `resource_access[client_id].roles` (client-specific roles). It compares these values against the list of roles defined at the route level. If a match is found, authorization succeeds. At this stage the plugin enriches the request with contextual headers such as `X-User-Id`, `X-Username`, and `X-User-Roles`, which are injected transparently before proxying the call to the upstream service.

The diagram also depicts the two possible failure paths, represented as alternative branches. If the token is valid but the caller lacks the required role, the plugin terminates the request with a 403 Forbidden, preventing the call from propagating to the upstream. If the token is inactive, expired, or revoked, the introspection endpoint responds accordingly, and the plugin returns a 401 Unauthorized. In both cases, the enforcement is performed at the gateway level, ensuring that upstream services are never exposed to unauthenticated or unauthorized requests. This separation of concerns embodies the architectural goal: the gateway enforces security, Keycloak remains the single source of truth, and upstream services are kept agnostic of identity protocols.

5.1.2.1 Introspection Plugin: Role, Configuration, and Impact

The plugin is a small Kong component written for the access phase. Two files are central: a schema describing configuration per route, and the handler that executes the checks. The schema declares what the administrator can define per route (client credentials for introspection, where to read roles from, and which roles are allowed). This is what gives the system fine-grained, route-specific authorization without touching the upstream code.

The first component is the plugin schema (Figure 18). This schema defines the configuration parameters that administrators can set on a per-route basis.

```lua
-- schema.lua (excerpt)
return {
  name = "keycloak-introspection",
  fields = {
    { config = {
        type = "record",
        fields = {
          { introspection_endpoint = { type = "string", required = true } },
          { client_id              = { type = "string", required = true } },
          { client_secret          = { type = "string", required = true } },
          -- where to read roles (realm or per-client)
          { roles_claim            = { type = "string", default = "realm_access.roles" } },
          -- route-level policy
          { allowed_roles          = { type = "array", elements = { type = "string" }, required = true }
},
          -- optional small TTL to amortize round-trips
          { cache_ttl_seconds      = { type = "number", default = 30 } },
        }
    }}
}
```

Figure 18 Configuration Parameters Schema

It includes the Keycloak introspection endpoint, the client credentials required to query it, the claim from which roles will be resolved, and the list of roles that are permitted for that particular route. Notably, the schema also introduces a small TTL cache to amortize introspection calls, balancing strict validation with efficiency.

The handler file operationalizes these policies (Figure 19). In its early stages, the plugin attaches to Kong's access phase, intercepting every incoming request. It retrieves the Authorization header, strips the Bearer token, and prepares the introspection call to Keycloak. The code excerpt shows how the plugin encodes the token, client ID, and secret into a POST request against the configured introspection endpoint. This is the technical realization of the first branch of the diagram, where Kong validates whether a token is active. Failure to provide a header, a malformed token, or a negative introspection response results immediately in a 401 Unauthorized, exactly as shown in the diagram's Figure 17 alternative paths.

```lua
-- handler.lua (excerpt)
local http  = require "resty.http"
local json  = require "cjson.safe"

local function introspect(conf, token)
  local httpc = http.new()
  return httpc:request_uri(conf.introspection_endpoint, {
    method  = "POST",
    body    = ngx.encode_args({
                token         = token,
                client_id     = conf.client_id,
                client_secret = conf.client_secret }),
    headers = { ["Content-Type"] = "application/x-www-form-urlencoded" },
    ssl_verify = false
  })
end

function plugin:access(conf)
  local auth = kong.request.get_header("authorization")
  if not auth then
    return kong.response.exit(401, { message = "missing Authorization header" })
  end

  local token = auth:match("[Bb]earer%s+(.+)")
  if not token then
    return kong.response.exit(401, { message = "malformed bearer token" })
  end

  -- Note 1-3: ask Keycloak if the token is active
  local cache_key = "kc:introspect:" .. ngx.crc32_short(token)
  local res_body = kong.cache:get(cache_key, { ttl = conf.cache_ttl_seconds }, function()
    local res, err = introspect(conf, token)
    if not res or res.status ~= 200 then
      kong.log.err("introspection error: ", err or res and res.status)
      return nil
    end
    return res.body
  end)

  if not res_body then
    return kong.response.exit(500, { message = "introspection failure" })
  end

  local payload = json.decode(res_body)
  if not (payload and payload.active) then
    -- alt branch: inactive token
    return kong.response.exit(401, { message = "invalid or expired token" })
  end
```

Figure 19 Handler.lua Code Snippet

If the token is valid, the handler moves to role resolution (Figure 20). Here, the plugin checks whether roles should be extracted from the realm scope or from the resource access tied to the specific client. This corresponds to code snippet note 4. The logic is deliberately flexible: different deployments may require role checks at different levels of granularity, and the plugin supports both modes without modification to the upstream application. This separation is critical, as it allows the same architectural pattern to support organization-wide roles or fine-tuned, per-service authorizations.

```
-- Note 4: resolve roles from configured claim
local roles = {}
if conf.roles_claim == "realm_access.roles" then
  roles = (payload.realm_access and payload.realm_access.roles) or {}
else
  local client = conf.client_id
  roles = payload.resource_access and
          payload.resource_access[client] and
          payload.resource_access[client].roles or {}
end
```

Figure 20 Role Resolution Code Snippet

Following role extraction, the plugin enforces the route-level policy (Figure 21). The loop over the declared allowed_roles matches the resolved roles from the token. If the set intersection is empty, Kong terminates the request with a 403 Forbidden. This is note 5 in the code snippet below, where the failure branch represents a user who is authenticated but not authorized for the endpoint. Again, the upstream remains untouched; authorization is enforced entirely at the gateway layer.

```
  --Note  5: enforce route-level policy
  local authorized = false
  for _, r in ipairs(roles) do
    for __, allowed in ipairs(conf.allowed_roles) do
      if r == allowed then authorized = true; break end
    end
    if authorized then break end
  end
  if not authorized then
    -- alt branch: lacks role
    return kong.response.exit(403, { message = "insufficient role for this endpoint" })
  end

  -- Note 6 : propagate identity to the upstream for auditing
  kong.service.request.set_header("X-User-Id",          payload.sub or "")
  kong.service.request.set_header("X-Username",         payload.preferred_username or "")
  kong.service.request.set_header("X-User-Roles",       table.concat(roles, ","))
end
```

Figure 21 Route-Level Policy Code Snippet

When authorization succeeds, the final step is identity propagation. Before proxying the request to the upstream service, the plugin injects contextual headers such as X-User-Id, X-Username, and X-User-Roles. This corresponds to Note 6 of the code snippet. By doing so, the upstream application receives just enough identity context for auditing and domain-specific

decisions, but it does not need to parse JWTs or implement Keycloak client libraries. This cleanly embodies the architectural principle of separating concerns: Kong enforces authentication and authorization, while the upstream remains focused on business logic.

These fragments illustrate why the plugin is central to the design: (i) the introspection call guarantees that acceptance of a token always reflects Keycloak's current view (revocations and expirations take effect immediately); (ii) The role resolution step implements truly route-scoped authorization; two routes on the same service can demand different roles without any modification to the upstream application; (iii) The identity headers give the upstream service enough context for auditing and domain-level decisions while keeping it free from token parsing or library coupling.

Finally, the plugin is attached to the service or to individual routes through Kong's Admin API. This binds the configuration shown above to a concrete entry point and is what enables the per-endpoint policy as reflected in the image below (Figure 22).

```
# attach the plugin to the internal proxy service that fronts Nuclio
curl -sX POST http://localhost:8001/services/nuclio-proxy-service/plugins \
  --data name=keycloak-introspection \
  --data config.introspection_endpoint=http://keycloak:8080/realms/core-gateway-realm/protocol/openid-
connect/token/introspect \
  --data config.client_id=kong \
  --data config.client_secret=$KC_SECRET \
  --data config.roles_claim=realm_access.roles \
  --data config.allowed_roles=integration_admin \
  --data config.allowed_roles=integration_user
```

Figure 22 Plugin Attachment to the Route

With this in place, the behavior depicted in Figure 23 is not merely conceptual; it is the runtime contract of every protected integration route. Kong becomes the single enforcement point, Keycloak remains the single source of truth, and upstream services only the requests that already satisfy the organization's authorization policy. Each route should have a similar setup to this:


```
            ],
            "client_secret": "mr4bUBXMKdlULJSMYINnBA5oesBiXbUI",
            "client_id": "core-gateway",
            "keycloak_introspection_url": "http://host.docker.internal:8080/realms/core-gateway-
        realm/protocol/openid-connect/token/introspect"
        },
        "created_at": 1747765766,
        "instance_name": null,
        "route": {
            "id": "7ecf8553-859d-4b39-b490-5db832fa0be4"
        },
        "ordering": null,
        "id": "abd00c54-50df-45ca-897a-1fef241d20d2",
        "name": "keycloak-introspection"
    }
  ]
}
```


Figure 23 Route JSON Setup

5.1.3 Issuer Misalignment through Gateway Mediation

During the development and testing phases, the authentication mechanism was based on direct communication with the Keycloak server, which was hosted locally and exposed via the default port `8080`. In this configuration, access tokens issued by Keycloak included an `iss` (issuer) claim that correctly reflected the server's canonical address:

```
"iss": "http://localhost:8080/realms/core-gateway-realm"
```

This alignment ensured that resource servers receiving such tokens were able to validate them successfully, as the issuer information matched the expected origin of authentication.

However, as the architecture matured, the system introduced an API gateway—Kong—to serve as a unified entry point for all client requests. This reverse proxy was configured to route requests to the Keycloak instance, now accessible indirectly through the gateway at:

```
http://localhost:8000/realms/core-gateway-realm/protocol/openid-
connect/token
```

Although this approach offered benefits in terms of centralized routing, observability, and plugin-based authorization (e.g., introspection), it introduced an unintended side effect. By default, Kong preserved the original `Host` header (`localhost:8000`) when forwarding requests upstream to Keycloak. Consequently, Keycloak interpreted the incoming request as having been addressed under port `8000`, and generated access tokens with an issuer claim reflecting this perceived address:

```
"iss": "http://localhost:8000/realms/core-gateway-realm"
```

This subtle discrepancy—between the actual service host (localhost:8080) and the host inferred from the gateway—proved problematic. Any component or plugin responsible for token validation and introspection, including the Kong keycloak-introspection plugin, compared the token's issuer against a static, expected value (in this case, still `http://localhost:8080/...`). Because the issuer claim no longer matched the origin of the Keycloak service, the tokens were deemed invalid or inactive, and access was subsequently denied.

This misalignment highlighted a nuanced but critical aspect of token-based authentication within service-oriented architectures: the perceived origin of token issuance can be unintentionally altered by intermediary layers such as gateways and proxies. Unless the gateway explicitly rewrites or masks the `Host` header—or unless the identity provider is configured to tolerate multiple issuer origins—authentication failures may occur, even when token structure and signature remain technically correct.

5.1.4 Diagnostic Validation of Issuer Alignment

This architectural issue, while subtle in configuration, had highly visible consequences at runtime. To investigate its impact systematically, a series of authentication tests was conducted under different client invocation contexts. Each test traced a distinct path through the system, as shown in Figure 24, allowing us to isolate the issuer misalignment and observe how it manifested across service boundaries.

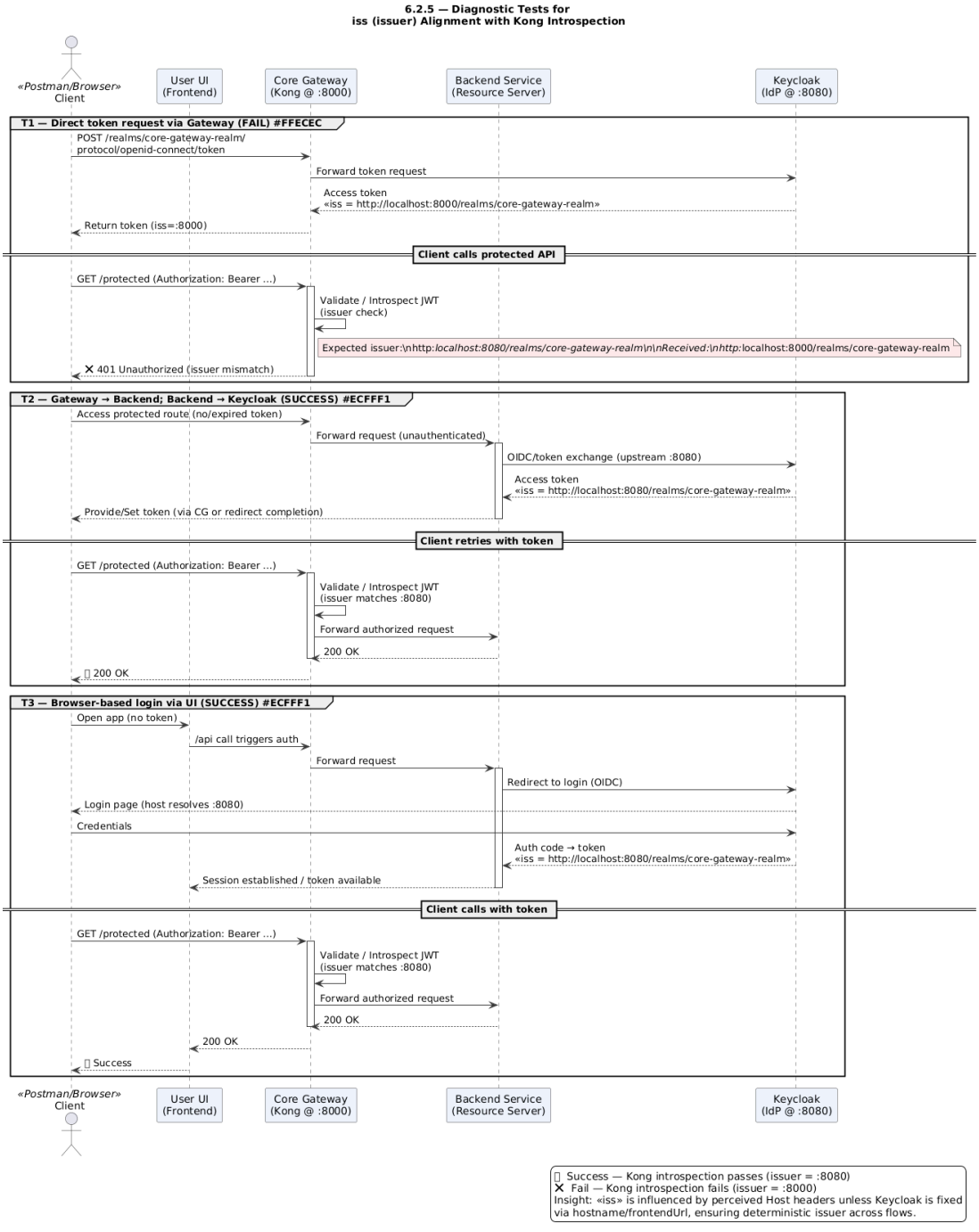


Figure 24 Diagnostic Tests for Issuer Alignment

In the first scenario—executed as T1, simulating a direct token request from Postman to Keycloak through the Kong gateway—observed a failure in token introspection. The token, although structurally valid and correctly signed by Keycloak, bore an `iss` (issuer) claim of `http://localhost:8000/realms/core-gateway-realm`, derived from the gateway's Host header. However, the introspection plugins configured within Kong, had been statically instructed to trust tokens only if their issuer was `http://localhost:8080/realms/core-gateway-realm`. The mismatch led to rejection of the token at the authorization layer, despite it being freshly minted and cryptographically intact.

In a contrasting test labeled T2**,** was introduced a change in the flow by having Postman contact the gateway, which then routed the request to the backend service, where token issuance occurred indirectly. In this flow, the backend delegated the authentication to Keycloak via its original upstream endpoint, bypassing the influence of Kong's forwarded headers. The token returned in this case carried the expected issuer—`http://localhost:8080/...`—and was successfully accepted by the introspection plugin. This result confirmed that the problem was not intrinsic to Keycloak itself, nor to the token validation logic in the plugin, but rather stemmed from the way in which the API gateway altered the perception of the request origin.

A third case, T3**,** sought to simulate a realistic user-facing scenario. Here, a simulated browser (represented by Postman) triggered the login process via the frontend, which then routed through the Core Gateway and backend service. The backend, in turn, initiated authentication by redirecting the request to Keycloak. This multi-layered flow introduced more complexity but nonetheless passed successfully—provided that the backend itself, rather than the gateway, initiated the token request. The resulting token once again bore the issuer tied to port 8080, consistent with the Keycloak internal service definition.

These tests served not merely as technical validations but as a diagnostic window into the broader issue: the token issuer is not an absolute property of the identity provider—it is contextually constructed based on the Host header and internal network resolution. This means that any component sitting between the client and Keycloak (such as a reverse proxy or gateway) must either sanitize headers or be accounted for explicitly in Keycloak's configuration.

Furthermore, these tests revealed an interesting asymmetry. When the token is issued through a UI-based flow—where redirection causes the browser to resolve `localhost:8080`—everything works as expected. But when tokens are issued programmatically from the gateway context, the issuer becomes contaminated by the gateway's external-facing port, resulting in silent but devastating incompatibilities with resource servers that rely on strict issuer checks.

In the designed architecture, the Kong gateway itself enforced these checks during introspection, requiring exact issuer alignment. This led to a puzzling paradox: while the gateway was responsible for routing and security enforcement, it also became the very cause of failed authentications by modifying the observable address space seen by the identity provider. This tension, while easy to overlook, is common in service-mesh designs and highlights the fragility of distributed authentication when issuer semantics are misaligned.

Ultimately, two remedies were available to make the issuer deterministic: (i) statically configuring Keycloak's hostname at deployment time (e.g., via Docker settings such as a fixed host and realm Frontend URL) so that tokens always carry a canonical `iss`, which decouples token semantics from proxy headers and container DNS; and (ii) preserving or rewriting forwarding headers at the gateway so that Keycloak perceives the intended public host and constructs the matching issuer. The first option provides stronger determinism and centralizes correctness in the identity provider, but introduces environment specificity (distinct hostnames per stage), requires careful alignment of redirects/cookies/CORS—particularly when TLS is terminated at the gateway—and demands disciplined governance of client registrations and redirect URIs, which can be cumbersome in multi-domain scenarios. The second option can be operationally convenient when multiple public entry points must coexist, but it reintroduces coupling to proxy correctness, increases the risk of regressions when routes or plugins change, and enlarges the security and maintenance surface area by relying on precise header management. In practice, the project adopted the first option to enforce a single canonical issuer and avoid proxy-induced drift, with the gateway-based approach retained as a secondary, topology-dependent alternative.

After evaluating both remedies—(i) pinning Keycloak's issuer via a fixed hostname and realm Frontend URL, and (ii) rewriting/preserving forwarding headers at the gateway—deliberately was chosen not to apply either, because the current architecture delegates authentication to Keycloak from the backend (backend-initiated OIDC against the upstream `localhost:8080`), rather than issuing tokens directly through Kong; in this configuration the resulting tokens naturally carry the expected issuer and pass Kong's introspection without additional changes. This decision minimizes configuration churn, avoids introducing environment-specific hostname management or fragile header rewriting, and preserves a stable, working runtime path. It is recognised that if gateway-mediated token issuance becomes a requirement in future iterations, one of the two remedies will be adopted to guarantee issuer determinism. Until then, the backend-initiated flow provides the necessary interoperability with the least operational risk.

Through this case, the experiment exposed one of the more insidious classes of authentication failures: situations in which tokens appear valid and credentials are correct, yet the semantics of trust collapse due to configuration drift within the network topology. This observation reaffirmed the necessity of tracing authentication not only at the conceptual level but also in its practical unfolding—across proxies, containers, ports, and header rewrites.

5.2 BPMN Engine: Adapter-Driven Execution

In the prototype, the BPMN Engine is the execution environment where models dispatched by the Process Execution Manager become running workflows. The architectural treatment in Chapter 4 positioned this container as replaceable and adapter-driven; the prototype turns that abstraction into a working setup by wiring a concrete engine into the framework's orchestration

path while preserving portability. Camunda 8 is adopted here as the reference engine, providing the runtime against which deployments, instance creation, and task handling are validated.

This choice reflects not only its maturity and distributed architecture but also its ability to integrate seamlessly with enterprise identity providers. Yet the framework deliberately avoids binding itself to a single vendor: by relying on adapters within the Process Execution Manager, BPMN actions are normalized through the BpmnAction enumeration and routed to whichever engine is configured. The orchestration pipeline will continue to be extensible and pluggable as a result, supporting alternatives like Apache Airflow, which can still be aligned with the framework through its adapter interface despite being DAG-based rather than BPMN-based.

A decisive advancement demonstrated in the prototype lies in the integration of the BPMN Engine with the centralized IAM. In contrast to earlier workflow platform generations that frequently depended on separate user stores, this configuration links the engine to the enterprise Keycloak realm directly. In reality, the required clients (Operate, Tasklist, Optimize, and Identity) were provisioned centrally, the bundled Keycloak container was turned off, and the Camunda Identity Service was redirected to the external realm. By centralizing authentication in a single trusted provider and coordinating role mappings with the framework's governance model, this configuration removes redundancy. While Appendix D provides a thorough examination of Camunda's internal architecture and its larger runtime ecosystem, the practical changes necessary to accomplish this were made at the level of Docker configuration files.

What emerges in the prototype is a unified authentication and authorization flow: actors authenticate through the enterprise IAM, tokens are validated by Camunda’s Identity Service, and fine-grained permissions are enforced consistently across all engine applications. This pattern is not limited to Camunda; through adapters, other engines such as Airflow can participate in the same federated identity scheme, confirming the broader engine-agnostic vision of the framework.

From this foundation, the next subsections provide greater detail. Section 6.3.1 explores the integration of workflow engines with IAM, illustrating how centralized identity governs authentication and authorization across heterogeneous execution platforms. Section 6.3.2 then examines how engines communicate and execute tasks, distinguishing the semantics of human tasks—requiring suspension and user involvement—from service tasks that automate system calls. Finally, Section 6.3.3 addresses the challenge of integrating heterogeneous systems into the IAM-centric framework, demonstrating how different authorization contexts (external providers, embedded logic, or unprotected endpoints) can all be reconciled within a uniform model. Together, these subsections show not only how the BPMN Engine operates in isolation but how it anchors the orchestration pipeline within the broader governance of the prototype.

5.2.1 Federating Workflow Engines with IAM

The integration of workflow engines with enterprise identity and access management (IAM) solutions represents one of the most important architectural evolutions in process automation. Earlier generations of engines relied on internal identity modules that managed users, roles, and groups in isolation. While sufficient for small-scale deployments, this approach created silos: each engine maintained its own user database, its own role model, and its own synchronization logic. In enterprise scenarios, this fragmentation generated inconsistencies, duplication of effort, and unnecessary security risks.

Modern engines have decisively abandoned this model. Platforms such as Flowable, Activiti Cloud, jBPM, and Zeebe (Camunda 8) now externalize identity management and delegate authentication and authorization to upstream IAM providers through standardized protocols such as OAuth 2.0, OpenID Connect, or SAML. This change is more than a technical convenience: it reflects a recognition that identity governance is a central enterprise concern that demands consistency and centralization. Instead of embedding user management within each engine, identity is now federated to specialized providers such as Keycloak, ensuring that access is controlled uniformly across applications and systems.

Camunda provides a particularly clear example of this architectural realignment. Its current distribution introduces an Identity Service and a distinct Authorization Service, both of which act as intermediaries between Camunda's applications (Operate, Tasklist, Optimize) and an external IAM. As shown in Figure 25, the Identity Service validates tokens against upstream providers, while the Authorization Service enforces fine-grained rules. Authentication filters in front of the REST API and web applications ensure that only validated tokens can access engine resources. In other words, authentication and authorization are no longer inseparable from the engine but are brokered services that can be coupled with an enterprise IAM.

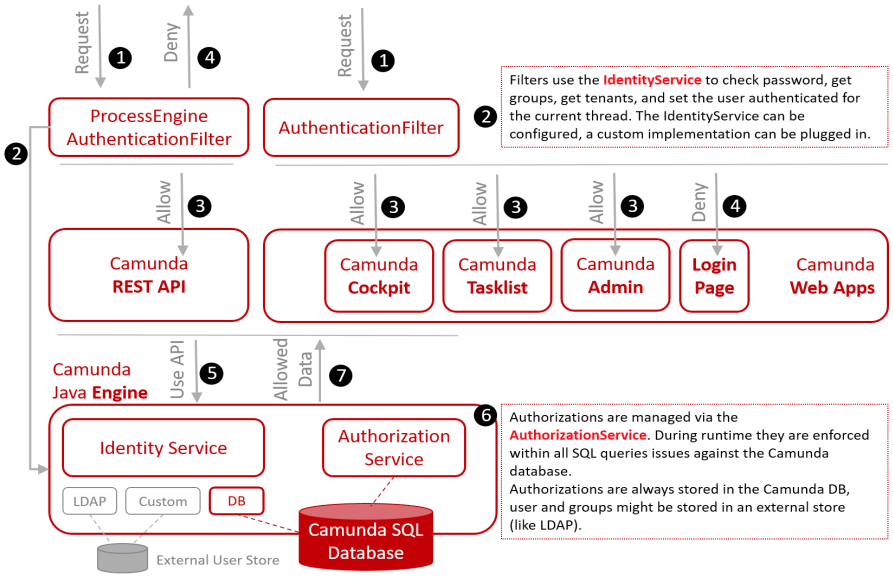


Figure 25 Camunda Authentication Architecture

In the framework, this decoupling allowed us to eliminate the bundled Keycloak instance provided by Camunda and instead connect the engine directly to the existing enterprise Keycloak realm. The process required reconfiguring the docker-compose file deployment so that the Identity Service pointed to the realm rather than the packaged container. The redundant Keycloak was disabled to prevent conflicts, and on the enterprise side we provisioned the required clients (operate, tasklist, optimize, and identity) while aligning their role mappings with the organizational model. Once configured, user authentication took place exclusively through the enterprise Keycloak, and Camunda's Identity Service became a broker that validated and propagated tokens downstream to the engine and its applications.

The result is a single, coherent authentication flow: actors sign in via the enterprise Keycloak, tokens are validated by the Identity Service, and the Authorization Service enforces permissions consistently across all Camunda components. This integration not only eliminates credential duplication but also enhances governance by ensuring that policies are defined, audited, and updated in a single, central location. Architecturally, it demonstrates the strength of separating responsibilities: engines focus on executing workflows, while IAM systems govern who is allowed to participate. This specialization ensures consistency, improves scalability, and guarantees that the framework remains compatible with ongoing developments in the BPMN ecosystem.

5.2.2 Task Execution Semantics in Workflow Engines

When using workflow/orchestration engines like Camunda or Apache Airflow in an integration architecture, it's essential to understand how each engine manages tasks, assigns work, and communicates between components. Below, we break down how these engines "talk" – i.e., how they execute tasks and interact with external systems or workers – focusing first on Camunda and then on Airflow. To ground this discussion in a concrete setting, the Product Name Flow Process has been selected as the illustrative example, a simple one that asks the users the product's previous name and the new one to replace. Figure 26 represents the process diagram.

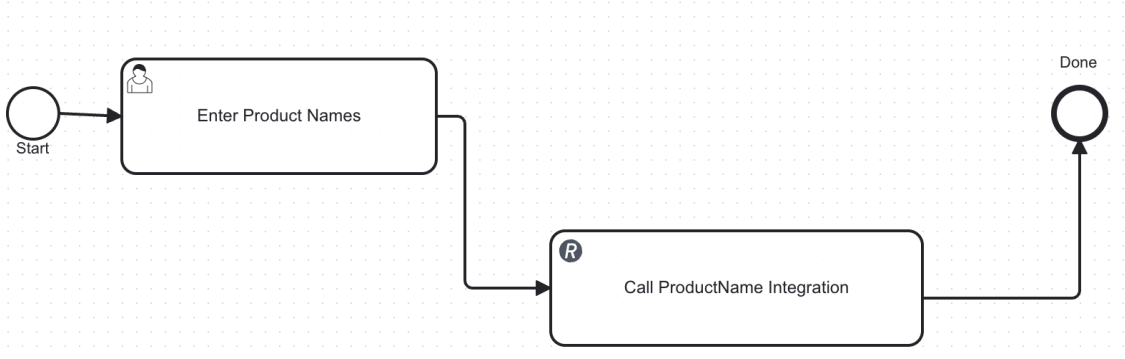


Figure 26 Product Name Flow Process Diagram

The execution of the *prodcutnameflow* process illustrates in concrete terms how workflow engines orchestrate the collaboration between human actors and automated services. The process begins with a human task, represented in the Tasklist interface as *Enter Product Names*. At this point, the engine suspends execution and creates a task entry that remains visible to all eligible users. Initially, the task is marked as unassigned, signaling that it awaits human intervention. Once a user claims it, the interface dynamically renders a structured form that prompts for the necessary inputs: the previous product name and the new product name. Only after these values are provided and the task is completed does the engine resume execution, seamlessly passing the collected data to the next activity in the sequence.

This behavior exemplifies the essential semantics of human tasks in workflow engines. Suspension of the process flow, persistence of state, and explicit user involvement are characteristics of human tasks. The engine's responsibility is to make sure that the work is appropriately assigned, visible, and finished by a designated actor, not to carry out the task itself. The Tasklist UI and its REST APIs, which expose operations for claiming, assigning, and completing tasks, are how platforms like Camunda accomplish this. Other engines achieve the same effect with different user-facing interfaces or APIs, but the principle remains constant: execution pauses until human action occurs, ensuring that critical decision-making or validation steps can be integrated directly into automated workflows.

The subsequent step in *prodcutnameflow*, labeled *Call Productname Integration*, demonstrates the complementary semantics of a service task. Unlike human tasks, service tasks represent work that can be executed entirely by a machine. Here, the engine no longer waits for human intervention but instead triggers the execution of a system-level activity. In Camunda, this may involve invoking a Java class through a delegate, leveraging a built-in REST connector, or creating an external task that is polled and executed by a worker application. In Apache Airflow, by contrast, the same logic is captured in an operator within a DAG, where the scheduler determines when the task is ready and dispatches it to a worker process for execution. Despite these implementation differences, the semantics remain identical: a service task is a unit of automated work, triggered by the engine, performed by software, and reported back without human interaction.

The diagram provided below (Figure 27) highlights this interplay in the framework context. The human task *Enter Product Names* is first created, assigned, and completed through user interaction. Once finished, the engine immediately transitions to the service task *Call Productname Integration*, invoking the deployed HTTP-based integration. This integration, routed securely through Kong and Nuclio, executes the update logic and propagates the changes across the finance and stock services. The diagram makes visible what the execution logs confirm: human and machine activities alternate within the same workflow, each executed according to their respective semantics, but coordinated into one seamless process by the engine.

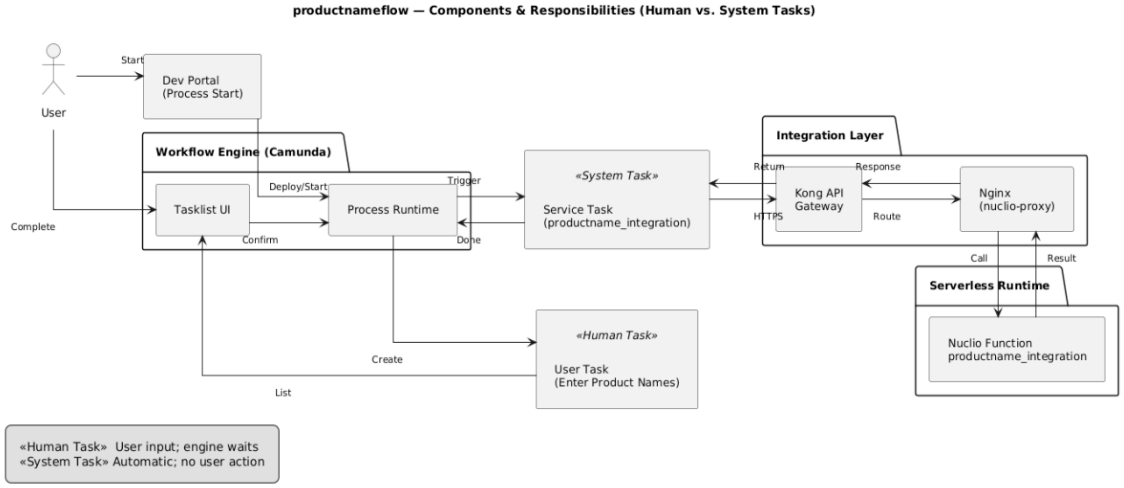


Figure 27 Product Name Flow Process in the Framework Context

These engine-agnostic properties are central to the architecture. By ensuring that integrations are always deployed as callable HTTP services, the framework guarantees that they can be consumed uniformly across heterogeneous workflow technologies. In Camunda, a service task may simply configure a REST connector pointing to the integration endpoint; in Airflow, the same endpoint may be called from a Python operator. The underlying integration does not change, nor does its deployment model. Similarly, human tasks are always surfaced through some form of tasklist interface, be it Camunda's UI or another platform's equivalent, reinforcing the principle that human–system collaboration is an invariant feature of process orchestration.

5.2.3 Integrating Heterogeneous Systems into the Keycloak-Based Framework

When integrating heterogeneous systems into a unified architecture, one of the most relevant challenges is the disparity of authorization mechanisms that each system may employ. In practice, three broad situations tend to emerge. Some systems operate with their own identity providers and full-fledged authorization services; others embed authorization directly in their application code, issuing and validating tokens without a separate service layer; and some, in their simplest form, expose endpoints with no protection at all. For the sake of clarity, these can be referred to as Option A, Option B, and Option C in the Table 4.

To describe the heterogeneity observed in practice, three baseline authorization contexts can be identified. The first, Option A**,** refers to external systems that already possess their own authorization service. This may be another Keycloak realm, an enterprise identity provider such as Okta or Azure AD, or a custom OAuth2 implementation. In this case, the framework does not attempt to override or replicate the fine-grained policies of that external system.

Table 4 Possible Systems Authorization Levels

Option	Authentication Source	Authorization Enforcement	User/Role Management
A	External system's identity provider (e.g., Keycloak realm, Okta, Azure AD)	Enforced internally by the external system	Roles and groups defined and maintained in the external IdP
B	Application itself (in-code JWT or local user database)	Enforced directly by application logic	Users stored locally, roles mapped in code or via lightweight converters
C	None (open access)	No enforcement; system accepts any request	No explicit user or role management

Instead, the strategy is to establish a trust relationship through Keycloak's brokering mechanism, allowing roles and claims from the external provider to be mapped into the Central-Realm. Once imported, these attributes appear in the central token, and Kong enforces only a minimal role check (for example, verifying that the user has the role *A_USER*). From that point onward, requests can flow into the external system under a centrally validated identity, while the external service remains free to enforce its own internal rules. The integration itself does not participate in any further checks: it simply invokes the system with a token that has already been authenticated and authorized at the gateway.

The second context, Option B, covers systems that do not rely on a dedicated identity provider but embed their authorization logic directly in the code. A typical example would be a Spring Boot microservice that enforces roles using annotations such as `@PreAuthorize` or through a custom JWT converter. In these cases, the integration strategy requires that the users and roles known to the external service be imported or federated into the Central-Realm. This ensures that Keycloak tokens can reproduce the same role claims that the system expects. Thus, when a request passes through Kong, the gateway can enforce precise role checks (for instance, ensuring that a caller has either *B_ADMIN* or *B_USER*). If the request passes, the integration invokes the service with a token already validated centrally. The application continues to enforce its own in-code logic without any modification, but crucially, the integration itself does not repeat any authorization. The system trusts the token because it already contains the expected roles, and the framework guarantees that only properly authorized users can trigger the integration in the first place.

The third context, Option C**,** corresponds to systems that expose open endpoints, with no authentication or authorization in place. At first glance, such systems present the highest risk, since any unauthenticated client could reach them. However, the integration framework neutralizes this risk by interposing Kong as the sole entry point. A coarse role is created in the Central-Realm (for example, *C_USER*) and assigned to all individuals who should have access. Kong enforces this role for every request directed to the system, ensuring that only

authenticated users, validated against Keycloak, can reach it. From the perspective of the open system itself, nothing changes—it continues to accept any request it receives. Yet the crucial difference is that only requests passing through Kong, carrying the correct centrally issued token, will ever reach it. The integration therefore executes its logic under the assurance that all traffic has already been authenticated and authorized upstream.

The integration strategy across Options A, B, and C reveals a common principle: the integration itself never performs authorization logic. Instead, every decision about whether a user may invoke an integration is resolved beforehand at the gateway, based on tokens issued by Keycloak. This has several advantages. It ensures consistency, since policies are written once in a central place and applied to all routes; it provides clarity of responsibility, since identity is managed by Keycloak, Kong performs enforcement, and integrations are dedicated exclusively to business logic; and it ensures scalability, since new systems—regardless of their initial authorization model—can be integrated without rewriting their internal code or duplicating checks inside the framework.

In practice, this means that when an integration such as *productname_integration* is triggered, the user initiating it has already been authenticated by Keycloak and validated by Kong. The integration then executes against its target systems with complete assurance that the call is legitimate. Whether those systems maintain their own fine-grained authorization (Option A), rely on in-code role checks (Option B), or expose open endpoints (Option C), the framework's responsibility is fulfilled at the gateway, and the integration itself remains a pure, role-agnostic execution unit.

These three options describe the authorization landscape that one can expect to encounter when connecting disparate systems. However, the framework developed in this project deliberately avoids tailoring its integration logic to each of these cases. Instead, it imposes a uniform approach to authorization that is anchored in Kong Gateway and Keycloak. Every request entering the framework must first pass through the gateway, where tokens are introspected and validated against the central identity provider. Only after this validation succeeds can an integration be invoked.

5.3 Core Backend: Operational Coordination Layer

The Core Backend, within the prototype, emerges as the principal coordination unit of the framework. Its architectural role is to mediate between user-facing portals and execution-oriented services, thereby consolidating all configurations, metadata, and administrative state into a single authoritative repository. In this capacity, it guarantees that every operation initiated through the interfaces—whether the deployment of a BPMN model, the registration of a new integration, or the adjustment of engine settings—is consistently registered, validated, and dispatched to the appropriate subsystem. By centralising this coordination, the Core Backend assumes the character of the system's "administrative memory," ensuring traceability and governance across the entire prototypical workflow.

From a technological standpoint, the prototype materialises this container in Spring Boot, a choice justified by its agility, its mature ecosystem, and the reliability of its integrated security framework. In its initial iterations, the backend tightly coupled itself to Keycloak through Spring Security, configured as an OAuth2 resource server, and complemented with a custom JwtAuthConverter. This converter extracted claims embedded in Keycloak-issued JWT tokens—particularly the resource_access field—and used them to enforce role-based restrictions directly at controller endpoints. Figure 28 illustrates this approach: requests arriving from user interfaces carried bearer tokens that were locally validated by Spring Security before any business operation could proceed. This arrangement provided fine-grained access control and adhered to best practices in microservice security, but it also introduced structural drawbacks. By embedding authorization logic within the backend itself, policies risked becoming entangled with application code, leading to configuration drift across deployments and undermining the principle of separation of concerns.

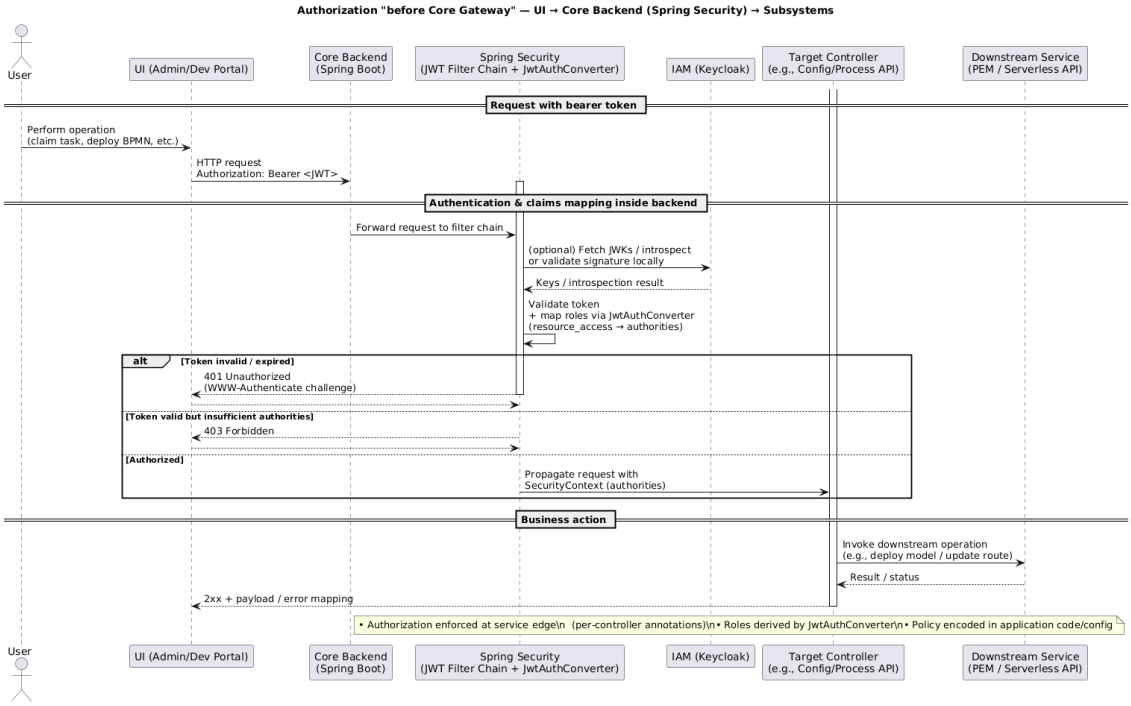


Figure 28 Token Validation in the Core Backend via Spring Security

These responsibilities were externalised to the Core Gateway, where a custom introspection mechanism now enforces role-based authorization consistently at the system's ingress. This transition, analysed in detail in the section (cf. Section 5.1.1), liberated the backend from direct security enforcement. Under the new model, authentication and authorization are performed upstream, and only sanctioned requests—enriched with identity context headers—arrive at the backend. The effect of this change is twofold: (i) the Core Backend becomes lighter and more resilient, focused exclusively on coordination and governance rather than enforcement; (ii)

security policies achieve a centralised and uniform application at the gateway level, strengthening the overall integrity of the framework.

In its refined role, the Core Backend functions as the central brain of the framework, concentrating the orchestration logic that governs how the system behaves as a whole. It is here that configurations are stored, administrative states are maintained, and coordination rules are enforced, ensuring that all operations progress in a coherent and policy-compliant manner. While the execution of processes and integrations is delegated to specialised containers such as the Process Execution Manager and the Serverless Service, the Core Backend provides the logical scaffolding that determines when and under what conditions those components are engaged. This makes it the authoritative point of governance within the prototype: not an executor of tasks, but the guarantor of consistency, accountability, and traceability across the framework. By consolidating these responsibilities, the Core Backend affirms its role as the orchestrator of the framework's logic, preserving modularity while sustaining the long-term adaptability of the architecture.

5.4 Serverless Service: Function-Oriented Runtime

Within the prototype, the Serverless Service assumes the role of executing integration logic in an elastic, event-driven environment, thus completing the chain of responsibilities that begins in the Core Backend and is mediated through the Process Execution Manager and the Integration Layer. The technology explicitly chosen for this container was Nuclio, reflecting its suitability for serverless execution in enterprise-grade contexts. Whereas in Chapter 4 this container was introduced in strictly architectural terms, highlighting its pluggability and production-grade maturity, in the present chapter, it is examined from a prototyping perspective. The goal is to demonstrate not only the conceptual role of Nuclio but also its operational presence within the prototype, where it functions as the runtime substrate for integration functions.

Nuclio was integrated into the framework because it offers a reliable execution model that bridges the integration artifacts produced in the Integration Layer with the orchestration logic defined through BPMN workflows. By packaging Ballerina-based integrations as Nuclio functions, each artifact becomes a self-contained and event-driven service, activated dynamically through HTTP triggers or other supported events. In this way, orchestration no longer depends on static or long-running services but can delegate execution to functions that are provisioned only when required, ensuring elasticity and reducing operational overhead. This design choice is illustrated in Figure 29, which outlines the responsibility overview of the Serverless Service. The diagram shows how integrations move from container images to deployable functions, how triggers initiate their execution, and how results are channelled back into the process flow. The figure therefore makes explicit the role of Nuclio as a mediating layer: it operationalises integration modules as on-demand services while maintaining consistency with governance and monitoring, thus ensuring that the framework can scale flexibly across heterogeneous enterprise environments.

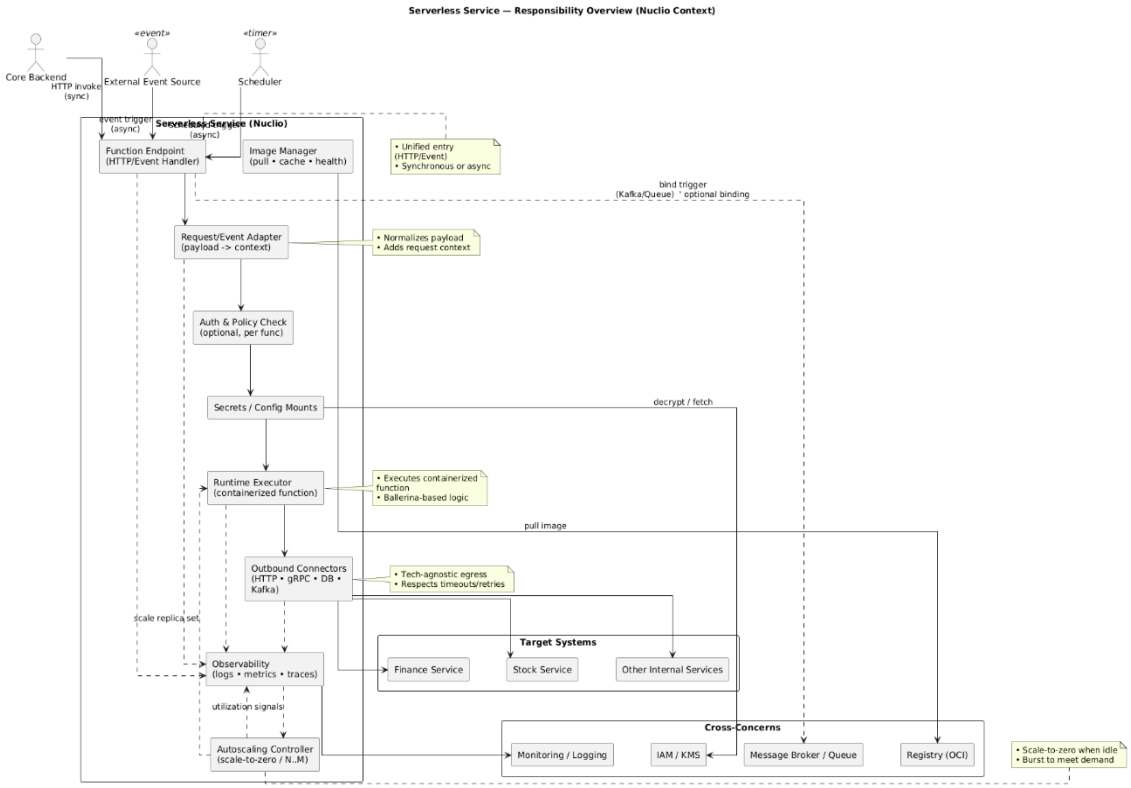


Figure 29 Serverless Service Responsibility Overview

In practice, the prototype validates two central aspects of Nuclio's role. The first concerns its communication model, which is deliberately reduced to standard HTTP interactions. As elaborated in (cf. Section 5.4.1), this decision enables any workflow engine—whether BPMN-based like Camunda or DAG-oriented like Apache Airflow—to invoke integrations without requiring specialized adapters or proprietary libraries. The second relates to its security model, formalized in (cf. Section 5.4.2), where Nuclio's inherently exposed HTTP triggers are encapsulated within private networks and made accessible exclusively through an internal proxy and the Core Gateway. This controlled pathway ensures that integrations remain fully elastic yet never bypass the governance enforced by the IAM system.

The internal complexity of Nuclio—its controllers, runtime managers, and autoscaling mechanisms—has not been expanded here, since such technical exposition is reserved for Appendix E, where Nuclio is compared in depth with other serverless platforms such as Fission. What matters at the prototyping stage is to underline that Nuclio embodies the operational proof of concept for the serverless model envisioned in Chapter 4. By confirming that integration projects can be scaffolded, containerized, deployed, and securely invoked within this runtime, the prototype consolidates the role of the Serverless Service as a critical enabler of the framework's modularity and scalability.

5.4.1 HTTP Invocation: Engine-Agnostic Portability

A decisive characteristic of the proposed framework is its ability to treat integrations as independent, self-contained services that can be invoked through simple HTTP calls. This design choice is not accidental but rather stems from the observation that modern workflow engines—regardless of their underlying paradigm—converge on HTTP REST as the de facto standard for external communication. Whether one is working with BPMN-based orchestrators such as Camunda, or DAG-oriented schedulers such as Apache Airflow, all engines expose native mechanisms to perform REST invocations, either through prebuilt connectors, declarative service tasks, or small code blocks embedded within the workflow logic.

By encapsulating integration logic into services that register their own HTTP listener, the framework ensures that any workflow engine can uniformly interact with them. For BPMN-based engines, such as Camunda, this means that service tasks can be configured with a simple HTTP connector pointing to the integration endpoint, making the invocation part of the process model itself. In practice, a task such as *"Update Product Name"*, in the BPMN diagram may call the integration previously deployed under http://localhost:8080/productname_integration, with input parameters mapped directly from the workflow variables. Airflow, while conceptually different, achieves the same outcome by embedding HTTP operators into its DAGs, ensuring that at runtime the corresponding REST call is issued and the result fed into subsequent tasks.

The consequence of this design is twofold. First, integrations remain completely technology-agnostic: whether written in Ballerina, Java, Python, or any other language, they become callable units so long as they expose a REST interface. Second, the orchestration layer remains uncluttered by engine-specific plugins or libraries, since the act of invoking an integration reduces to an HTTP exchange—a protocol universally supported across all execution environments. This strategy dramatically simplifies portability, since the same integration may participate in a Camunda workflow, an Airflow DAG, or even a custom orchestrator, without requiring any adaptation of its internal code.

In short, the reliance on HTTP as a unifying invocation mechanism transforms integrations into reusable assets that transcend the boundaries of individual engines. Instead of binding logic to a single orchestration platform, the framework promotes a modular approach in which integrations behave as services accessible to any engine capable of issuing an HTTP call. This not only guarantees interoperability but also future-proofs the framework against technological shifts in workflow tooling, since the contract is reduced to the most widely adopted communication standard in distributed systems.

5.4.2 Layered Security for Integration Execution

As part of the integration framework developed in this project, one of the most critical requirements was protecting executable integrations against unauthorized access. Nuclio functions, once deployed, automatically expose lightweight HTTP triggers, which make them

immediately callable by any external client. While this behavior is convenient during rapid development and local testing, it becomes a liability in production environments. Exposing these endpoints without any control mechanism would allow unauthenticated users to trigger functions directly, undermining both security and governance.

To meet this challenge, the prototype adopts a layered architecture that funnels all traffic through a carefully controlled pathway, as illustrated in Figure 30 Network Architecture. Rather than allowing external actors to access Nuclio functions directly, requests must traverse three distinct control layers, each contributing a specific security property:

1. Private networking of Nuclio containers provides strict isolation by ensuring that functions are not directly reachable from the outside world.
2. An internal Nginx proxy acts as the sole bridge between the private network and the public-facing environment, mapping human-readable paths to internal Nuclio addresses and ports.
3. Kong API Gateway serves as the final external entry point, integrating with Keycloak to enforce authentication, authorization, and observability.

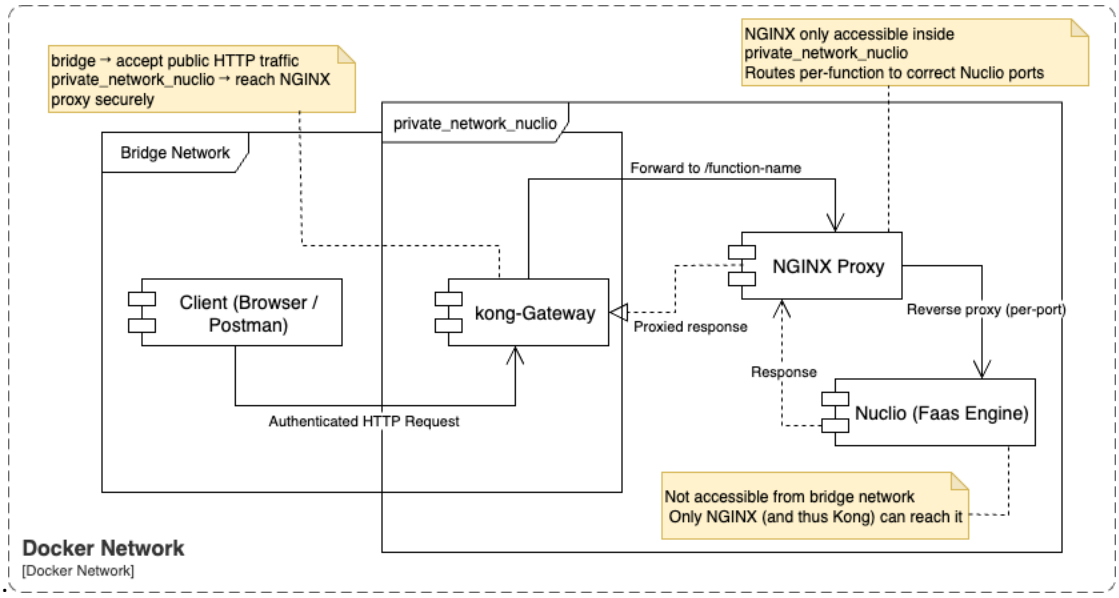


Figure 30 Network Architecture

This design achieves two complementary objectives: Nuclio functions remain insulated from direct exposure, and authorization is centralized and uniform, with policies defined in the Identity and Access Management (IAM) service and enforced consistently at the edge.

When deploying a Nuclio function via nuctl, it is configured to run within a private Docker network, inaccessible to any external clients. This means that, even if a function listens on a valid HTTP port, it cannot be reached directly from the host machine or external services. This network boundary ensures that any communication with the function must be routed through approved internal pathways.

To make the private Nuclio containers reachable in a controlled way, an internal Nginx proxy container is introduced. For every deployed integration, the system performs the following actions:

- Discovers the HTTP port exposed by the function using Nuclio CLI and parses its metadata.
- Automatically modifies the nginx.conf file, adding a new location block that maps a specific path (e.g., /finance, /stock) to the Nuclio container's internal address and port.
- Reloads the Nginx configuration on the fly, so that the new route becomes immediately active.

Through this mechanism, the Nginx container becomes the only bridge capable of reaching Nuclio's private network from the outside. However, even this Nginx instance is not directly exposed to users. To complete the secure path, all user-facing traffic is routed through Kong Gateway, which acts as a full API management layer. A logical service is registered in Kong pointing to the Nginx proxy container (http://nuclio-proxy:8080), and for every deployed integration, a corresponding Kong route is created pointing to this service as demonstrated with example in the Figure 31.

```
curl -X POST http://localhost:8001/services/nuclio-proxy-service/routes \
  --data "paths[]=/finance" \
  --data "strip_path=false" \
  --data "methods[]=POST"
```

Figure 31 Kong Integration Route Creation

This means that when a client sends a request to /finance, Kong first intercepts the call, validates any authentication tokens (if configured), and only then proxies the request to the internal Nginx container, which in turn proxies it to the correct Nuclio function.

This layered model brings numerous benefits. First, it ensures security through isolation, as Nuclio functions are never directly exposed to the outside. All requests are funneled through Kong, which acts as a central access control point—enforcing authentication methods such as JWT, OAuth2, or Keycloak. Additionally, the system supports dynamic routing; as new integrations are developed, their routes are automatically registered in Nginx, as shown in Figure 32, and Kong.

```
http {
    server {
        listen 8080;
        location /productname_integration {
            proxy_pass http://host.docker.internal:55287;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }

    }
}
```

Figure 32 Nginx Configuration File

This setup also offers excellent scalability, allowing multiple functions to operate simultaneously, each with its own URL and port, while maintaining a modular and programmatically manageable configuration. This aligns with modern API gateway architectures, where edge proxies enforce security and routing policies, while backend services remain simple and focused on business logic.

5.5 Process Execution Manager: Engine-Agnostic BPMN Hub

The Process Execution Manager became a decisive element during the prototyping stage, as it provided the concrete mechanism through which BPMN workflows were validated, deployed, and executed. Unlike the architectural design presented in the chapter 4, which focused on conceptual responsibilities and modular separation, the prototype required a pragmatic consolidation of these ideas into a functioning runtime service.

From the outset, the Process Execution Manager was designed as a standalone Spring Boot component, exposing a REST interface to the rest of the framework. This enabled the Core Backend to forward commands in a standardized manner, while the Execution Manager handled the operational details of BPMN lifecycle management. Instead of being handled as abstract entities, BPMN models were actually kept in a specific database along with their XML definitions, deployment timestamps, statuses, and related metadata (like descriptive tags). This decision provided not only traceability but also resilience, since processes could be re-deployed, rolled back, or audited without requiring constant re-uploads from external tools.

Central to this prototype was the implementation of a unifying action model. To avoid hard-coded or engine-specific semantics, all operations—such as deployment, start, stop, or deletion—were expressed through a dedicated enumeration called BpmnAction. This abstraction proved critical when integrating with heterogeneous engines. Although adapters were emphasized in the architectural model, the prototype made this idea a reality: commands coming in from the Core Backend were converted into BpmnAction values, which engine adapters then converted into the necessary API calls. For example, when targeting Camunda, a START action started a new process instance in the running engine, while a DEPLOY action caused the adapter to push the raw BPMN XML via the Zeebe API.

The introduction of adapters in the prototype highlighted both the flexibility and the challenges of this approach. In the current implementation, a dedicated Camunda Adapter was created, acting as the bridge between the generic Process Execution Manager and the Zeebe workflow engine. Its responsibilities included translating normalized actions into REST or gRPC calls, handling authentication tokens, and ensuring that failed operations were logged and retried when possible. The design leaves room for additional adapters—such as for Airflow or Temporal—but within the scope of the prototype, Camunda served as the concrete demonstration. This validated the framework's pluggability principle: the Process Execution Manager did not need to be rewritten to accommodate a new engine, only extended with a new adapter.

Equally important was the role of validation. During the prototype phase, the BPMN Validator component was implemented to perform both syntactic checks on the XML structure and semantic checks on the logical flow of models. These validations, though initially simple, prevented frequent runtime errors that arose when incomplete or malformed models were submitted. The Process Execution Manager refused to deploy invalid processes, thus enforcing a first layer of governance before any orchestration reached the engine itself.

Beyond deployment and validation, the prototype also emphasized process state management. Each BPMN process stored in the database carried a status field (e.g., deployed, suspended, active) and a timestamp of the last operation. This made it possible for administrators to query the current catalog of available processes, filter them by status, and maintain a historical record of changes. Tags were also introduced, enabling processes to be grouped or classified, thereby improving governance when multiple workflows coexisted within the same environment.

Problems were also discovered during the prototyping stage. During deployment, for instance, Camunda's APIs occasionally returned ambiguous error codes, making it challenging to differentiate between network-level failures and malformed models. This was lessened by adding improved logging to the Process Execution Manager, which records unprocessed responses and links them to the original BpmnAction. After being linked to the framework's monitoring layer, these logs—which were crucial for debugging—ensured visibility for administrators and developers alike.

In conclusion, the prototype's Process Execution Manager consolidated its function as the BPMN workflow execution center. It materialized the architectural vision through a Spring Boot service that persisted processes, validated their correctness, normalized commands through BpmnAction, and executed them via engine adapters. More than a simple execution client, it became the practical guarantor of BPMN lifecycle governance, demonstrating that the framework could remain engine-agnostic while offering a consistent and secure way to manage processes. This materialization confirmed the feasibility of the design described in the previous architectural design Chapter 4.

5.6 Integration Layer: Systems Interaction Operationalization

Within the prototype, the Integration Layer emerges as the conduit through which abstract orchestration logic is transformed into concrete interactions with heterogeneous systems. The layer establishes a disciplined methodology for integration, ensuring that services are onboarded in a standardized, reproducible, and technology-agnostic manner. External system contracts, defined through formal specifications, are converted into executable modules by following a contract-first approach. In this context, the prototype concretely adopted Ballerina as the implementation language, given its native support for integration patterns and its capacity to operationalize contract-first development in a streamlined and reproducible way.To reinforce this discipline, the Integration Layer provides intuitive interfaces and diagrammatic tools that allow developers and stakeholders to visualize system-to-system interactions clearly, making the design of integrations transparent, auditable, and aligned with governance requirements while leaving the technical implementation details to subsequent stages of the framework. In this model, descriptions such as OpenAPI serve as the foundation for generating strongly typed clients and scaffolding integration projects, which are then containerized, deployed, and securely exposed through the serverless runtime and the gateway. Once operational, these integrations are invoked as first-class services within business processes. The subsections that follow expand on this lifecycle, from specification exposure to deployment, showing how the prototype validates the design while demonstrating its scalability and reproducibility across diverse enterprise contexts.

Although the prototype employs Ballerina as the reference technology, primarily due to its native support for OpenAPI and its alignment with integration-centric development, the framework remains deliberately neutral with regard to programming languages. The choice of Ballerina is therefore illustrative rather than prescriptive, serving as a proof-of-concept for how integrations can be systematized within the architecture.

5.6.1 Project Structure and Developer Workflow

In the prototype, each integration is organised as a self-contained Ballerina project with a predictable template that supports repeatable engineering practice. The project exposes a single, explicit entry point—main.bal—which declares the service listener and, therefore, the public interface by which the integration is invoked. System interactions are encapsulated in generated client modules, placed under the modules/directory, which are produced from the external systems' OpenAPI descriptions. This separation of concerns—entry-point orchestration in main.bal, protocol and payload handling in typed client modules—ensures that integrations are authored against stable contracts rather than ad-hoc connectors, preserving conformance as those contracts evolve.

The project includes a scripts/ workspace that automates the integration lifecycle from scaffold to artifact. Automation covers: (i) project creation and naming, (ii) contract retrieval and client generation, (iii) static checks and build, and (iv) containerisation via bal build --cloud=docker.

Packaging the integration as a Docker image is a deliberate design choice: it embeds the Ballerina runtime with the code, yielding an immutable artifact that executes consistently across environments and sidesteps the absence of native FaaS support for Ballerina. The same image can thus be promoted through development, staging, and production with configuration supplied externally, aligning with the framework's emphasis on reproducibility.

Connectivity is governed by a minimal exposure configuration maintained alongside the code. An nginx.conf fragment is generated and updated by the automation to register a private upstream for each integration; routes are then surfaced through the platform's ingress components so that functions remain reachable only through controlled, proxy-mediated paths. In other words, the Integration Layer prepares integrations for safe publication (route registration and addressing), while policy enforcement at the edge is handled elsewhere in the framework. This workflow yields integrations that are standardised, versionable units—easy to build, reason about, and operate without relying on scenario-specific assumptions.

5.6.2 Enabling and Retrieving OpenAPI Specification

The prototype implementation of the Integration Layer operationalizes the principle of contract-first design by grounding every integration in a formally defined OpenAPI specification. To enable this, each external service must provide a machine-readable description of its interface. In the case of the services developed for the prototype—such as those in the stock and finance domains—this was achieved through the use of *SpringDoc*, a library that automatically generates and publishes an OpenAPI specification. As illustrated in Figure 33, the configuration required only the inclusion of a Maven dependency, which activated the production of Swagger documentation and exposed it at a dedicated endpoint.

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.1.0</version>
</dependency>
```

Figure 33 Enabling Swagger/OpenAPI Specification through SpringDoc

It is important to note, however, that the framework does not mandate the use of SpringDoc specifically. Other systems may employ different mechanisms to generate and expose their Swagger or OpenAPI specifications, whether through alternative libraries, platform-native tools, or manually curated descriptors. The essential requirement is simply that the system provides a consistent contract in OpenAPI format, since this artifact constitutes the canonical basis for subsequent integration steps.

Once enabled, the specification becomes available at the /v3/api-docs endpoint. At this stage, the JSON representation of the API is retrieved, which constitutes the canonical contract for the

integration lifecycle. Unlike the visual interface, which primarily aids human comprehension, the JSON file ensures strict consistency and verifiability, serving as the definitive description against which integration modules are generated. Retrieval requires explicit content negotiation, for instance:

curl -H "Accept: application/json" -o swagger.json http://localhost:8080/v3/api-docs

This ensured that the output was not merely documentation but a precise and machine-consumable artifact, free of inconsistencies and suitable for automated processing. The framework then consumed this JSON specification through Ballerina's OpenAPI tooling, specifically the ballerina openapi export command. This operation transformed the abstract description into a concrete client module, structured as a strongly typed library of callable methods:

```
ballerina openapi export \
  --input swagger.json \
  --output modules/finance_system \
  --mode client
```

The --mode client flag is particularly significant. Ballerina's tooling is capable of generating either resource-based services, which expose endpoints, or remote-based clients, which consume them. Since the framework's requirement is to integrate with existing systems rather than expose new ones, this flag ensures that the output is a remote client module rather than a redundant service scaffold.

The generated module—finance_system in this example—contains a file (finance_system.bal) where each endpoint defined in the original Swagger contract is faithfully translated into a method of a typed Client class. For example:

```
public isolated client class Client {
    remote function getAllFinanceItems() returns http:Response|error;
    remote function updateFinanceItem(string id, json payload) returns
http:Response|error;
}
```

This translation elevates the role of the specification beyond static documentation. By converting it into a type-safe library, the framework ensures that integrations can be developed with compile-time guarantees, reduced runtime errors, and full fidelity to the external system's published contract. In this way, the specification itself becomes an active participant in the integration lifecycle, providing the foundation for consistent and reproducible development across the Integration Layer.

5.6.3 Integration Logic Composition

At this stage, the framework elevates the generated client libraries into full-fledged integration projects. Rather than treating integrations as ephemeral scripts or isolated functions, each one is scaffolded as a structured Ballerina project with its own repository-like codebase and lifecycle.

This design decision reflects a methodological commitment: integrations are not simply runtime artifacts, but reproducible and maintainable units of software engineering that can evolve, be versioned, and be deployed independently.

The process begins with project scaffolding. When a new integration is defined, the framework executes the bal new <projectName> command, which generates the canonical folder layout expected by the Ballerina compiler. This includes a Ballerina.toml configuration file, a main.bal entry point under the src/ directory, and dependency folders. Scaffolding ensures that all metadata and compilation settings are initialized correctly and that the integration can be treated as a first-class citizen of the Ballerina runtime. This step is invoked programmatically by the Integration manager UI, which will be addressed in further chapters, and issues the command inside the dedicated generated_integrations/ directory, isolating each integration as a deployable unit. Figure 34, presented below, confirms these actions, showing the project scaffolded, the main.bal file written, and the modules copied into the integration package.

```
2025-08-06 14:50:58 Saved spec to /Users/rafaeloliveira/IntegratedSystemsFramework/integratic
2025-08-06 14:50:58 bal add finance_system
2025-08-06 14:50:59 bal openapi ... --mode client
2025-08-06 14:53:49 Scaffolded project → productname_integration
2025-08-06 14:53:49 Wrote main.bal for 'productname_integration'
2025-08-06 14:53:49 Copied modules into 'productname_integration'
```

Figure 34 Integration Manager Console Trace

Immediately after scaffolding, the framework initializes the project by injecting two critical components: (i) a minimal main.bal file that creates a listener tied to the integration's name, and (ii) the client modules generated from OpenAPI specifications. The injected main file, though simple at first, ensures that the integration has an executable entry point and a listener that exposes it over HTTP.

```
main_bal = f"""
import ballerina/http;
import ballerina/io;

type ParamRequest record {{
    string[] params;
}};

listener http:Listener integrationEP = new(8080);

service / on integrationEP {{
    resource function post {integration_name}(http:Caller caller, ParamRequest data) returns error? {{
        io:println(">> Received params: ", data.params.toString());
        check caller->respond({{ received: data.params }});
    }}
}}
"""
```

Figure 35 Integration's Base Code Snippet

This is fundamental to the philosophy of the framework: every integration is not only a piece of logic but also a service, identifiable by a unique listener that allows other components, users, or orchestration engines to invoke it. At this point, the integration can be deployed, invoked, and tested even before its logic has been enriched with orchestration or API calls, as shown in the following code snippet (Figure 35).

The second injection involves copying the generated client modules into the project's modules/ directory. These modules, produced in the previous subsection via Ballerina's OpenAPI CLI, encapsulate the APIs of the external systems that the integration must call. By embedding them within the integration project, the framework guarantees that each integration is self-contained, with no hidden dependencies or runtime code generation required. This approach makes the integrations portable and reproducible, qualities that are essential when scaling across multiple services or environments. A schema of the modules injection workflow previously described is present in the Figure 36.

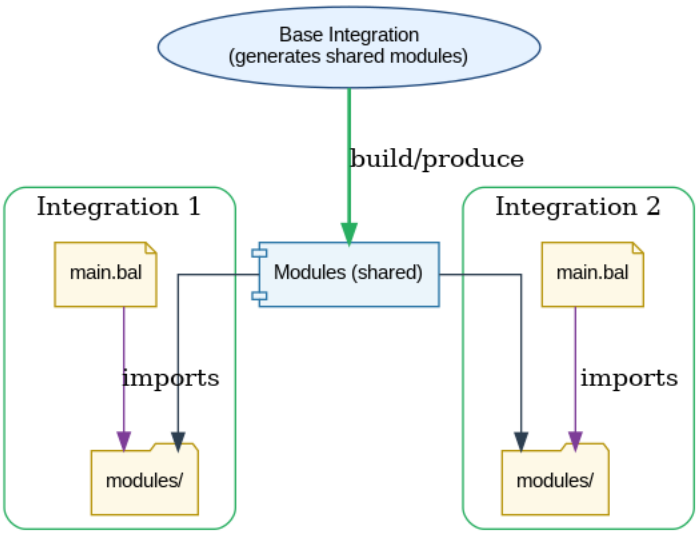


Figure 36 Modules injection workflow

Although Ballerina's module system underpins the prototype, the methodology remains neutral with respect to programming language. If the integration were developed in Python, one might rely on "bravado" to consume the Swagger definition; in Java, tools such as OpenAPI Generator would produce a client with equivalent semantics. What matters is not the specific technology but the abstraction: the external service is encapsulated as a callable dependency that adheres to the contract.

5.6.4 Integration Deployment Workflow

The deployment of integrations within the prototype follows a structured and layered process designed to ensure both operational reliability and security. As illustrated in Figure 37, the framework deliberately avoids exposing raw integration endpoints directly to the outside world. Instead, it enforces a deployment pipeline in which integrations are encapsulated, isolated, and routed through controlled layers of mediation.

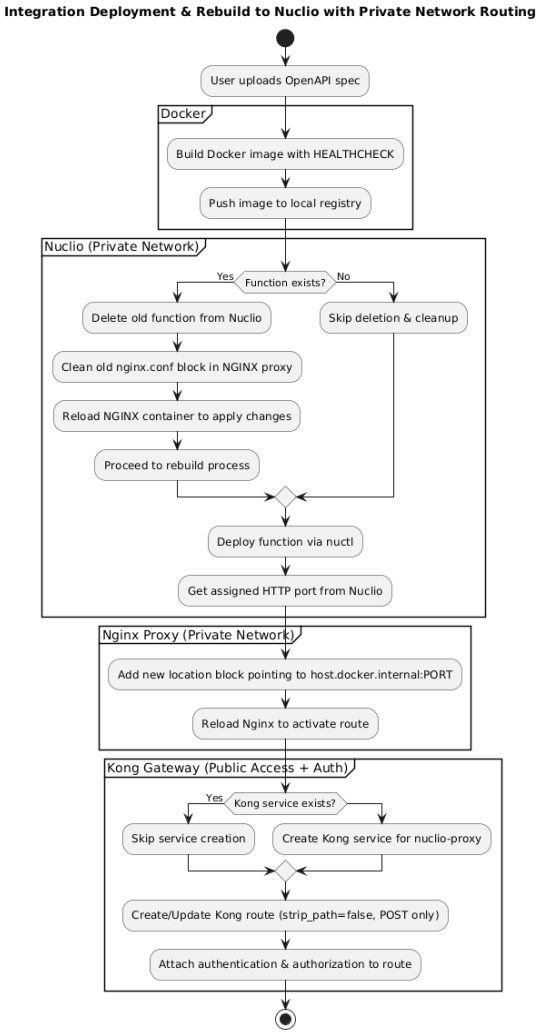


Figure 37 Integration Deployment Pipeline

At its foundation, the integration code is compiled into a Docker image using Ballerina's build tooling. This image incorporates not only the integration logic but also the runtime environment. Once built, the image is pushed to the internal registry, where it becomes available for orchestration. The image is then deployed into the serverless runtime, instantiated as a Nuclio function within a private Docker network. This confinement ensures that, while the function exposes lightweight HTTP triggers for invocation, these are not reachable from the external

network. By restricting accessibility in this way, the framework prevents unauthorized requests and enforces a baseline of isolation and fault containment.

From this point, routing and exposure are carefully layered. The nginx proxy provides stable addressing and connection pooling, mapping the internal endpoints of each function to controlled ports. Above this, the Kong Gateway registers these routes and applies the governance logic defined by the Identity and Access Management (IAM) system. It is at this stage that authentication and authorization are enforced, ensuring that only validated and policy-compliant requests ever reach the underlying functions. The layered model thus distributes responsibilities: Nuclio guarantees scalability and elastic execution, the proxy ensures network stability, and Kong centralizes policy enforcement at the system boundary.

This deployment pipeline shows that the prototype extends the framework's contract-first methodology with strong runtime governance in addition to operationalizing it. Integrations are handled as disciplined services that adhere to a repeatable lifecycle, which includes packaging, deploying, isolating, and securing, rather than as standalone scripts. The prototype confirms that it is feasible to manage heterogeneous integrations at scale while maintaining architectural clarity and security by enforcing this layered architecture.

5.6.5 Bridging the Runtime Gap: Ballerina on Serverless Platforms

One of the biggest obstacles faced during the deployment phase was that Ballerina was not natively supported by widely used Function-as-a-Service (FaaS) platforms like AWS Lambda, Azure Functions, Google Cloud Functions, or even open-source frameworks like Fission and Nuclio. For popular runtimes like Node.js, Python, Go, or Java, these platforms usually provide preconfigured execution environments. Developers can upload their code and let the platform take care of the underlying runtime configuration. Nevertheless, Ballerina has not yet gained enough recognition to merit direct runtime support in these environments since it is still a specialized programming language, even though it excels in integration scenarios.

This lack of support presented an efficient obstacle. Whereas a developer working with Node.js or Python could immediately deploy source code into a managed FaaS runtime, a Ballerina developer is instead confronted with an empty or incompatible execution layer. Attempts to define a custom Fission environment based on a “ballerina” image, for example, revealed that the platform could not properly recognize or register the runtime, resulting in deployment failures or functions that could not be scheduled. In other words, without explicit support, the orchestration platform was unable to launch or invoke Ballerina functions reliably.

To resolve this, the framework adopted a container-first deployment strategy. Rather than relying on FaaS environments to provide a Ballerina runtime, each integration was compiled into a self-contained Docker image using Ballerina’s built-in cloud tooling .This command, (bal build -- cloud=Docker), automatically generates a Dockerfile and packages the integration code together with the full Ballerina runtime as shown with the Figure 38.

```
FROM ballerina/ballerina:2201.10.0-slim
WORKDIR /home/ballerina
COPY target/bin/productname_integration.jar .
CMD ["bal", "run", "productname_integration.jar"]
```

Figure 38 Generated Ballerina Dockerfile

By encapsulating the runtime in this way, the resulting image is guaranteed to execute consistently across any container-supporting platform, including Nuclio, Kubernetes, or Fission's container-based executor model. The container is then deployed as a Nuclio function within a private Docker network, ensuring both reproducibility and isolation.

This approach, although requiring the overhead of building and pushing Docker images, provides several advantages: (i) it gives complete control over the runtime environment, eliminating the uncertainty of whether the platform will support specific language features; (ii) it guarantees consistency: the exact same image can be reused across testing, staging, and production environments without modification; (iii) it integrates seamlessly with the rest of the framework's deployment pipeline, where images are automatically pushed to the local registry, deployed to Nuclio, and exposed through the secure proxying layers of Nginx and Kong.

By adopting container encapsulation as the default deployment mechanism, the framework effectively sidesteps the absence of direct Ballerina support in FaaS platforms. It establishes a robust methodology where integrations, regardless of the language's popularity or ecosystem maturity, can still be treated as first-class functions within the serverless execution model. This decision reflects both a pragmatic workaround and a strategic design choice, ensuring that the framework remains flexible enough to incorporate less mainstream languages without sacrificing reliability or scalability.

5.7 User Interfaces: Bringing the Framework to Users

While the preceding sections of this chapter have progressively assembled the core architectural components of the framework—gateway, backend, process execution manager, BPMN engine, integration layer, and serverless runtime—the prototype only becomes operationally meaningful when these components are exposed to human actors through dedicated user interfaces. Administrators, developers, and end users cannot interact directly with BPMN controllers, adapters, or function triggers; their interaction is mediated by interfaces that translate the orchestration logic into intelligible and verifiable operations. These interfaces, namely the Admin UI, the Developer Portal, and the User UI (Tasklist), were already presented in their architectural detail in Chapter 4, where their modular composition was analyzed. What follows in this chapter is their positioning in the prototype: how they complete the assembly of the framework and serve as the finishing layer that transforms a distributed architecture into a usable system.

5.7.1 Admin UI

The Admin UI is designed as the cockpit of governance. Through it, administrators exercise authority over configurations, authorization rules, and system health, ensuring that the framework remains secure and operationally sound. What distinguishes this interface is its capacity to unify oversight and action in one environment: administrators can simultaneously monitor service status, adjust engine settings, and enforce route-level policies, thereby bridging the traditional gap between observability and control. This features are represented by the navbars presented in Figure 39.

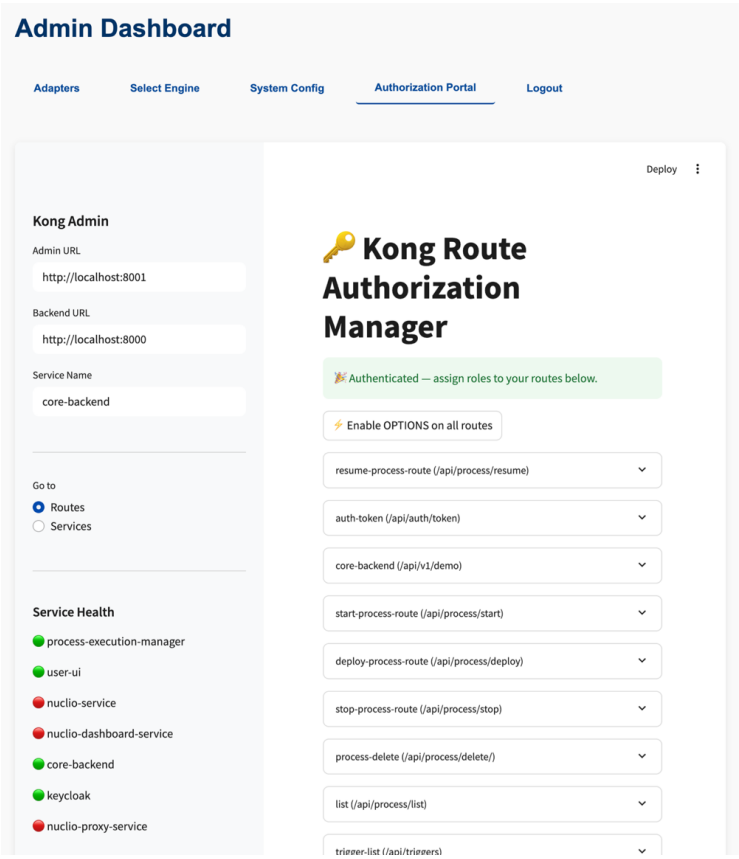


Figure 39 Admin UI (Authorization Portal View)

At the core of the Admin UI lies the Authorization Portal**,** which integrates directly with the Kong Gateway and the identity provider (Keycloak). Every route exposed by the gateway is surfaced here as a configurable entry, and administrators can assign or revoke roles with a simple interaction and in an integrated way with the IAM. This design embodies the principle of centralized authorization: instead of scattering role checks across services or embedding them in integration logic, all access control is consolidated in one place, where it can be audited, adjusted, and aligned with enterprise policies. The interface not only displays the available routes but makes explicit their binding to the identity realm, transforming abstract security models into visible, enforceable structures.

Complementing authorization management is the Service Health Check**,** a lightweight but effective monitoring layer that reveals the availability of the framework's core components. By fetching the services registered in Kong and pinging each endpoint, the UI produces a real-time status overview: whether the Process Execution Manager is reachable, whether the Core Backend responds within acceptable latency, or whether the Nuclio proxy is active, as shown in the Figure 40. Although this monitoring is not intended to replace full observability stacks like Prometheus or Grafana, it provides administrators and developers with immediate assurance that the system's backbone is functional. In this way, the Admin UI embodies the dual responsibilities of governance: defining who may access what, and ensuring that the underlying infrastructure is alive to honor those definitions.

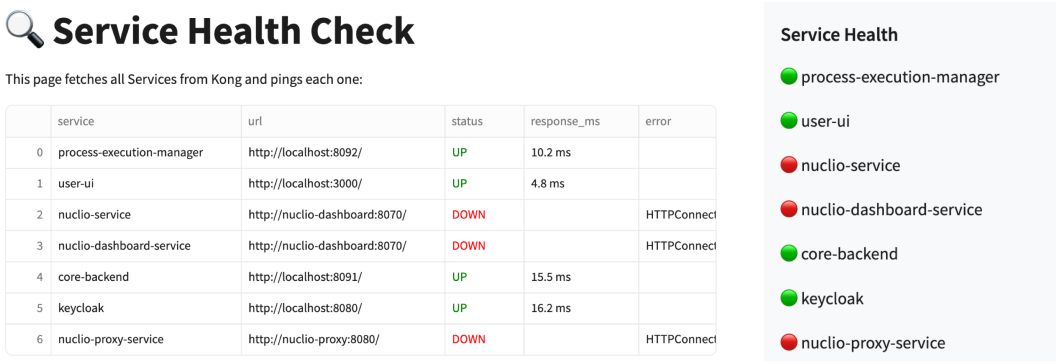


Figure 40 Admin UI (Service Health Check View)

Taken together, the Authorization Portal and the Service Health Check consolidate the Admin UI as the framework's governance hub. The former ensures that authorization policies are enforced coherently across all exposed routes, tightly coupled with the enterprise identity realm, while the latter guarantees that these policies rest upon an infrastructure that is continuously monitored for availability. This dual emphasis on policy control and operational assurance elevates the Admin UI beyond a conventional configuration dashboard, situating it instead as the architectural cockpit through which governance is both declared and validated. In this way, the prototype demonstrates that administrative oversight must not only articulate security models but also verify the vitality of the systems that sustain them, thereby uniting normative authority with empirical assurance in a single, integrated environment.

5.7.2 Developer Portal

If the Admin UI is concerned with security and governance, the Developer Portal is dedicated to creation and lifecycle management. It is here that integrations are designed, processes are deployed, and runtime behaviors are inspected. The Dev Portal abstracts the technical scaffolding of integrations into structured workflows, ensuring that developers do not have to interact with raw containers or gateway APIs, but can operate within a curated environment that enforces consistency and best practices. The navbars are organized by feature as shown in the Figure 41.

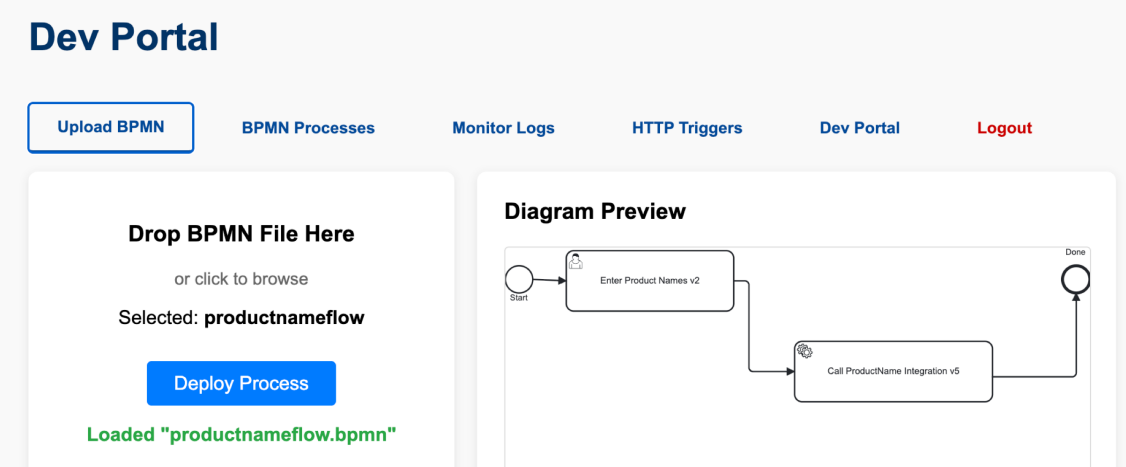


Figure 41 Dev Portal (BPMN Upload View)

The starting point is the BPMN upload interface, which allows developers to submit process models directly into the framework. This feature is more than a file uploader: the portal immediately renders the uploaded model into a diagrammatic preview, ensuring that the developer can visually validate its correctness before deployment. Once confirmed, the BPMN file is compiled and registered with the Process Execution Manager, transforming abstract diagrams into executable processes. By integrating preview and deployment, the portal makes explicit the principle of contract-first orchestration: models are verified, validated, and executed in one seamless flow.

Once deployed, processes are catalogued in a process management view. Each deployed workflow appears with metadata such as its identifier, status, tags, and last updated timestamp. From here, developers can inspect, start, suspend, or delete process instances. This catalogue transforms the Dev Portal into a governance layer for execution, ensuring that process management is visible and traceable rather than opaque or hidden in engine internals, as shown in Figure 42.

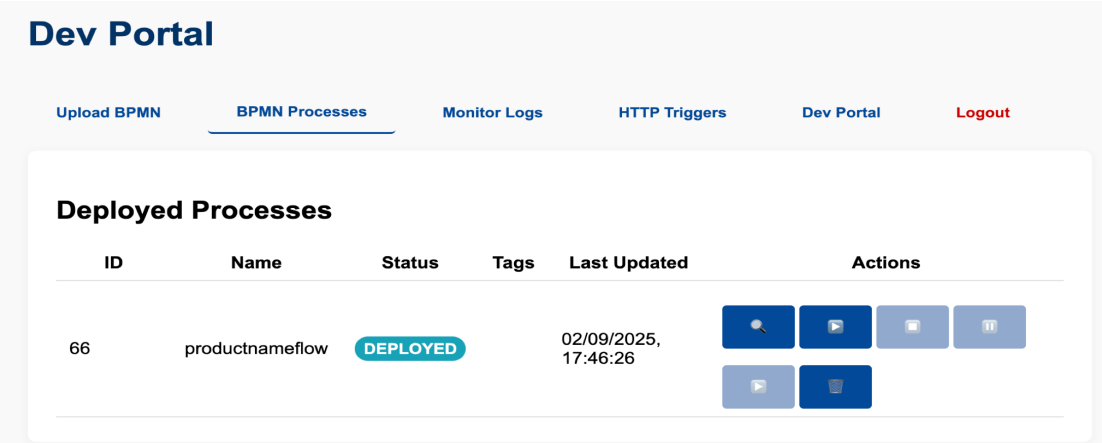


Figure 42 Dev Portal (Deployed BPMN Processes View)

Beyond workflows, the Dev Portal also encompasses the Integration Manager, which enables developers to generate new integrations and publish them as serverless services. Through this interface, integrations are scaffolded, built, and deployed to Nuclio with a single interaction, following the container-first methodology that underpins the framework, as demonstrated in Figure 43. Developers can track logs, verify deployments, and ensure that integrations are correctly exposed through the gateway. By encapsulating this pipeline into one environment, the Integration Manager aligns with the framework's goal of reducing the cognitive burden of integration development.

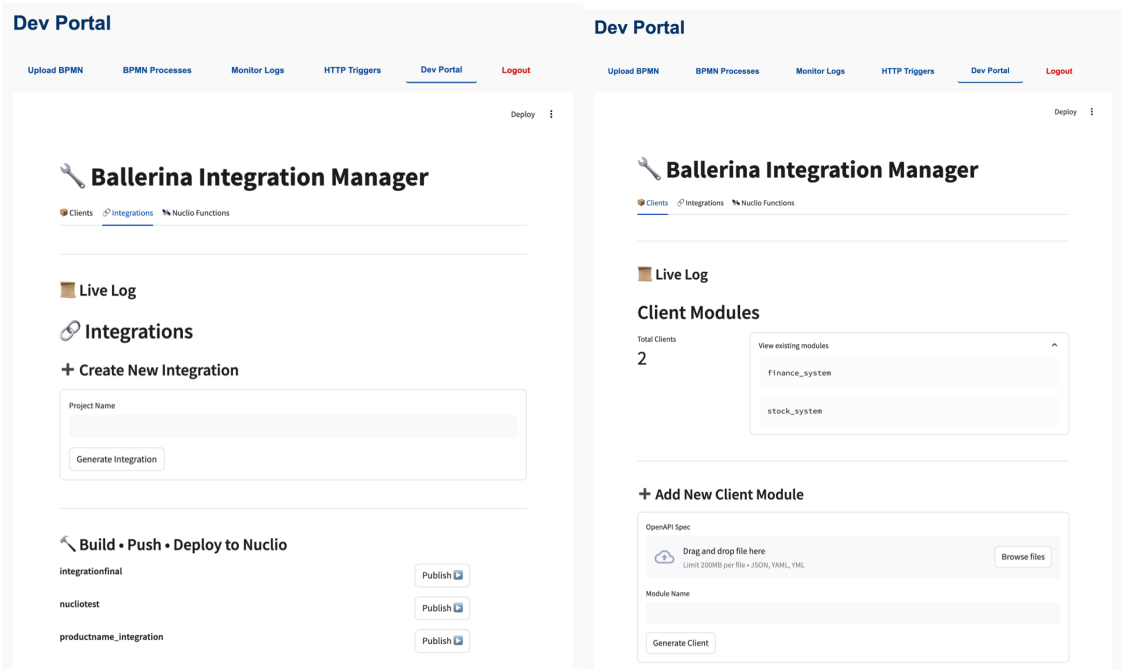


Figure 43 Dev Portal (Integration Manager – Create Integration View and Client Module View)

Finally, the Integration Manager supports the generation of client modules directly from OpenAPI specifications. Developers upload service contracts, and the portal transforms them into typed clients that can be invoked by integration logic, as shown in the left part of the Figure 43. This feature enforces the discipline of contract-first integration: external systems are consumed through their documented APIs, reducing ambiguity and ensuring interoperability. Together, the upload of BPMN processes, the management of deployed workflows, and the generation of integrations and clients form a coherent ecosystem in which developers can move seamlessly from specification to execution without leaving the portal.

5.7.3 User UI (Tasklist)

The User UI completes the triad by providing the interface for execution, where human actors interact directly with running processes. While the Admin UI governs and the Dev Portal creates, the Tasklist is where users perform the tasks that processes assign to them. It embodies the

human-in-the-loop principle, ensuring that workflows can pause for input, collect structured data, and resume seamlessly once the task is complete.

When a workflow instance reaches a human task, the engine suspends execution and creates an entry visible to the designated user group. This entry appears in the Tasklist, where the user can claim the task and finish it using a structured form. The workflow proceeds automatically after the form is submitted, triggering the following service tasks specified in the BPMN model (for example, updating the product name in the validation scenario). With the same level of dependability as automated calls to outside services, this guarantees that user input is not auxiliary but rather integrated straight into the orchestration path.

By surfacing tasks in this way, the User UI makes explicit the dual nature of workflows: they are both automated sequences of service calls and collaborative processes that require human decision-making. The Tasklist becomes the space where these two worlds converge, ensuring that the contributions of human actors are captured with precision and orchestrated in harmony with the automated logic of the system.

5.8 Difficulties and Remedial Strategies

During the refinement of the deployment pipeline, several obstacles emerged. One of the most recurring issues was related to the generated Dockerfiles. In their original form, these files lacked any HEALTHCHECK instructions, which meant that when a container silently crashed, the orchestration environment had no reliable mechanism to detect or restart it. This omission was harmless in trivial tests but became a source of instability in long-running services due to the fact that Nuclio only permits the integrations to run if they are in a healthy state. To address this, the framework was extended with a patching step that automatically injects appropriate health check instructions into each Dockerfile before the final build. In this way, container failures could be surfaced early, and the platform could react accordingly.

Another difficulty was encountered during redeployment. Nuclio enforces uniqueness in function naming, which means that an attempt to deploy an integration under a name that already existed was rejected outright. This behavior led to stale functions remaining active, creating confusion in testing and disrupting routing. The solution consisted of introducing a pre-deployment cleanup stage, where the system checks for existing functions with the same name and, if found, deletes them before proceeding with a fresh deployment. What initially seemed like a limitation of Nuclio was transformed into a predictable and stable redeployment strategy.

Routing itself also presented its own pitfalls. Because Nginx and Kong maintained their configurations across executions, there were cases where outdated routes persisted even after functions had been replaced. As a result, client requests were being forwarded to nonexistent or obsolete containers. This inconsistency was resolved by introducing dynamic reconfiguration logic: once a new function is successfully deployed, the framework updates the Nginx configuration with the function’s internal port, reloads the service, and immediately refreshes

the corresponding Kong route. With this mechanism, routing always reflects the current state of deployed functions and prevents accidental mismatches.

6 Practical Scenario

The present chapter consolidates the trajectory developed thus far in this dissertation by shifting from architectural abstraction (Chapter 4) and prototypical implementation (Chapter 5) to the demonstration of the framework in a realistic enterprise setting. Whereas earlier chapters outlined the conceptual underpinnings of the Integration Framework and validated its feasibility through the implementation of core containers, the following pages aim to situate the framework within a concrete scenario. The purpose is twofold: first, to show how the proposed design behaves under practical conditions of distributed service integration; and second, to highlight its capacity to incorporate organizational governance, human participation, and business-specific logic in addition to purely technical interoperability.

In order to achieve this validation, a practical scenario was constructed that simulates a recurring problem in enterprise information systems: the synchronization of overlapping business entities across independently developed and deployed services. The scenario is deliberately chosen to represent not only the technical complexity of integrating heterogeneous systems but also the managerial and organizational requirements that frequently condition integration flows in real-world environments. By doing so, the scenario tests both dimensions of the framework: its technical robustness and its adaptability to enterprise governance.

6.1 Synchronizing Overlapping Business Entities through the Integration Framework

Figure 44 introduces the overall setting. Two services—Finance and Stock—are deployed as containerized microservices on isolated servers, each with its own database and operational autonomy. As discussed, such decentralization is a hallmark of modern enterprise architectures, promoting agility while simultaneously complicating interoperability. Although both services must manage the same real-world entity—the product name—the attribute is represented differently within each domain. In the Finance Service, the field may appear as `FinanceItem`, reflecting its use in invoicing and financial reporting. In the Stock Service, it might be denoted as `StockItem`, serving inventory control and supply chain management. Despite these variations in syntax and naming conventions, the attributes are semantically equivalent and must remain consistent. Without a mechanism to reconcile such differences, updates applied in one domain would fail to propagate reliably to the other, leading to discrepancies across financial and operational processes. This challenge highlights the need for an integration

framework that can map heterogeneous representations onto a shared semantic layer, ensuring coherence across distributed systems.

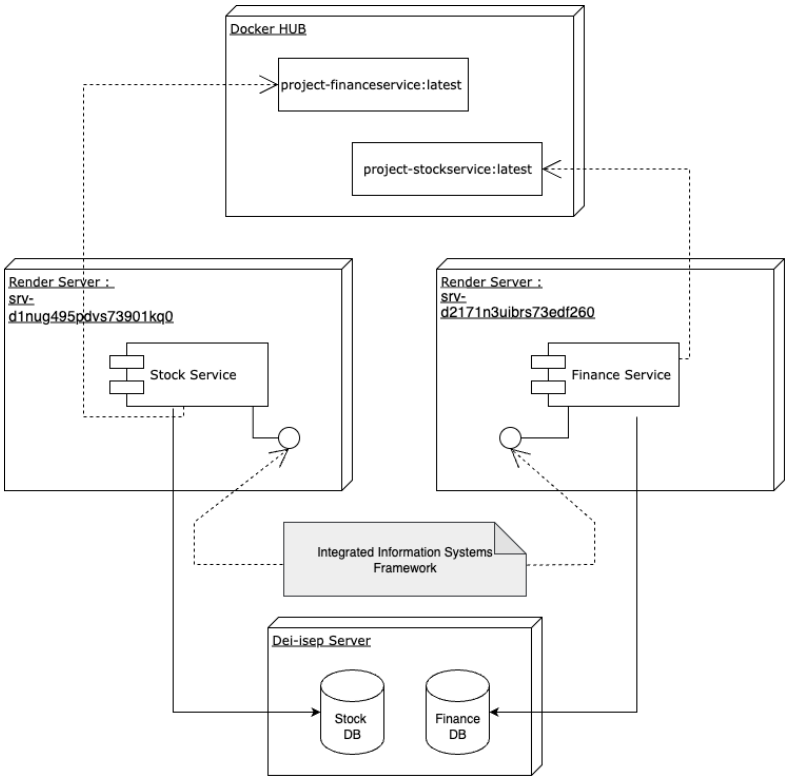


Figure 44 Practical scenario overview

What distinguishes this scenario from a simple synchronization exercise, however, is the incorporation of workflow orchestration with human participation. Instead of automatically propagating updates from Finance to Stock, the framework requires that certain operations be reviewed and approved by a manager before the change is enacted. This step, modelled explicitly in the BPMN process presented later in this chapter, demonstrates the flexibility of the framework to integrate human tasks and business rules into what would otherwise be a purely technical process. By doing so, the scenario mirrors the operational realities of many organizations, where automated flows are subject to managerial oversight, compliance checks, or conditional branching based on organizational policies.

As mentioned, at the center of this scenario are two microservices: the Finance Service and the Stock Service. Each is independently developed, containerized, and hosted on isolated render servers—srv-d2171n3uibrs73edf260 and srv-d1nug495qdxs73901kq0, respectively. Both services are built and versioned via Docker, with their respective images (project-financeservice:latest and project-stockservice:latest) maintained and pulled from a remote Docker Hub registry. This deployment strategy ensures operational autonomy and promotes decentralized service development and delivery pipelines. Yet, this very independence, while beneficial for scalability and modularity, creates the risk of fragmentation: updates made in one

domain cannot be assumed to propagate correctly to the other without an overarching coordination mechanism.

This is precisely where the Integration Framework becomes indispensable. Unlike traditional point-to-point integrations, where a change in System A is automatically and rigidly propagated to System B, the framework introduces a hybrid paradigm that combines automation with human oversight. In the present scenario, the synchronization of product names is not triggered blindly; rather, it begins as a workflow within the BPMN Engine, where updates are first reviewed and approved by an administrator before being propagated downstream. This design ensures that technical consistency is preserved while also embedding organizational policies, managerial validation, or compliance checks directly into the process. Moreover, because the orchestration is modeled in BPMN, the mode of execution can be adapted at any moment: the same workflow could be reconfigured to operate fully automatically, or conversely, to introduce additional decision gateways and human approvals. In doing so, the framework demonstrates a flexibility that point-to-point integrations inherently lack, transforming synchronization from a rigid data-transfer mechanism into a governed business process that can evolve alongside enterprise requirements.

The orchestration of this scenario builds directly on the use cases that were introduced earlier in Chapter 4 (cf. Section 4.1) as part of the Big Requirements analysis. Those diagrams outlined, at a conceptual level, how developers, administrators, and end users interact with the framework's core containers, from creating and publishing integrations in the Integration Layer to assigning roles and policies in the Authorization Portal, and invoking processes via the Core Gateway. In the present chapter, these previously defined use cases are not repeated but further elaborated through complementary sequence diagrams and narrative explanations. By doing so, the prototype contextualizes the abstract interactions already established in the architectural design and demonstrates their operational realization within the practical scenario. This layered exposition ensures continuity: what was initially presented as high-level requirements in Chapter 4 is here translated into concrete execution steps that govern how BPMN models are deployed, how integrations are invoked, and how end-to-end flows materialize across heterogeneous systems.

6.2 Step 1 – Accessing the Framework

The entry into the framework begins with its initialization, which is achieved by executing the startup script (./start-framework.sh). This procedure instantiates the necessary containers and activates the supporting infrastructure, thereby laying the operational foundations for subsequent integration tasks. Once the environment is live, actors are directed towards role-specific interfaces: developers access the Developer Portal (DevPortal) to design, upload, and manage processes, while administrators rely on the Admin UI to configure engines, manage authorizations, and monitor system health. These dual entry points embody the role-based

interaction model previously articulated in Chapter 4 (User Interfaces), where each interface was deliberately associated with a governance or operational perspective.

Irrespective of the entry point, all interactions are funneled through the Kong Gateway, which enforces authentication and authorization in coordination with Keycloak. This arrangement materializes the architectural commitment, introduced in Chapter 4 (cf. Section 4.4), to establish a single controlled ingress point where policies and identities converge. In the prototype described in Chapter 5 (cf. Section 5.1.2), this was operationalized through an introspection plugin that validates JSON Web Tokens (JWTs) at the edge of the framework.

The practical sequence of this access flow is captured in UC6 – Authenticate (OIDC via Gateway), which had already been introduced in the Actor–Goal Use Case Diagram of Chapter 4 and is revisited here with a sequence diagram (Figure 45). The diagram represents the intended uniform path of authentication: the user initiates login through the UI, the request traverses the Kong Gateway, and is forwarded to the Core Backend, where Keycloak is engaged for credential validation. Upon successful verification, the gateway performs token introspection, after which requests are permitted to flow towards the orchestration layers, whether the Process Execution Manager or the Integration Layer.

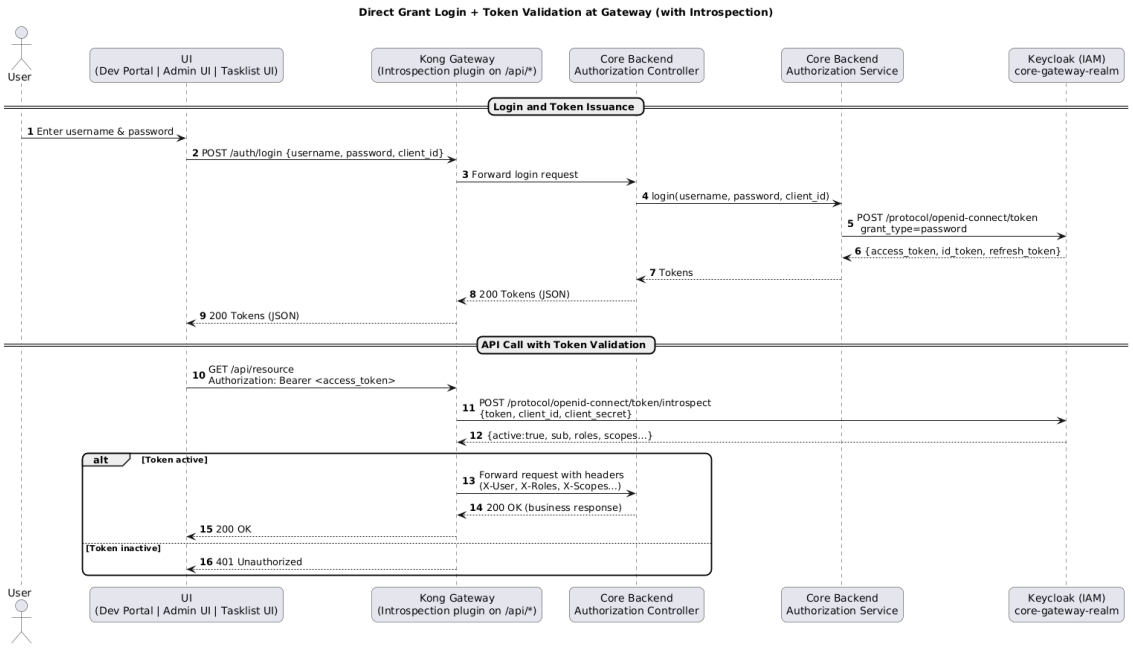


Figure 45 UC6 Sequence Diagram – Authenticate (OIDC via Gateway)

Although conceptually sound, this sequence also exemplifies the discrepancies that often arise between architectural intent and implementation reality. As demonstrated in Chapter 5, the introspection-based approach introduced fragilities, particularly concerning issuer alignment and dependencies on enterprise-only features of Kong. These limitations do not invalidate the architectural principle itself—the centralization of authentication at the gateway—but they reveal the adaptive challenges inherent in moving from abstract design (Chapter 4) to concrete

deployment (Chapter 5). In practice, the outcome is a more nuanced understanding of access: not merely the enablement of user logins, but the establishment of a governance perimeter where identity, authorization, and observability converge, validating the framework's vision of a unified ingress while acknowledging the practical adjustments required to sustain it in real-world scenarios.

6.3 Step 2 – Preparing the Integration Environment

Before heterogeneous systems can be orchestrated through the framework, the integration environment must be carefully prepared in a structured and technology-agnostic manner. The central requirement of this stage is to expose the APIs of external services in a standardized format—typically through OpenAPI (Swagger) specifications—and to transform those specifications into callable artifacts that can later be invoked by the orchestration logic. As established in Chapter 4 (Framework Proposal), this principle of contract-first design ensures that integrations are not fragile ad hoc connectors but rather governed, verifiable units that support reproducibility and long-term maintainability.

In the prototype implementation, this preparation was realized by exposing the APIs of the Finance and Stock services as OpenAPI specifications. As discussed in Chapter 5 (Enabling and Retrieving OpenAPI Specification), these specifications could be accessed in two complementary ways: a human-readable visualization through Swagger UI, and a machine-readable JSON artifact retrieved via the /v3/api-docs endpoint. The latter serves as the canonical contract for the integration lifecycle, ensuring that all subsequent modules remain faithful to the published interface. Once retrieved, typically through a request configured with proper content negotiation, the specification is processed by the framework to generate strongly typed clients. The detailed mechanisms of this transformation, including the use of Ballerina's OpenAPI tooling, are presented in Chapter 5.

The generated artifact described above does not remain an isolated output but immediately enters the interactive workflow mediated by the DevPortal. As shown in Figure 46, the administrator or developer uploads this JSON specification through the UI, assigns it a module name, and thereby triggers its translation into a reusable, standards-compliant client library. In practice, this means that what begins as an abstract description of an external API now becomes a tangible software component injected into the generated_integrations directory. From that point onward, the integration logic can call endpoints directly through strongly typed methods, ensuring that the entire lifecycle—from contract retrieval to client injection—remains verifiable and governed. What might appear at first glance as a simple drag-and-drop action therefore conceals a critical architectural guarantee: no integration can bypass the formal path of

specification validation, client generation, and managed injection, reinforcing the framework's methodological commitment to contract-first integration.

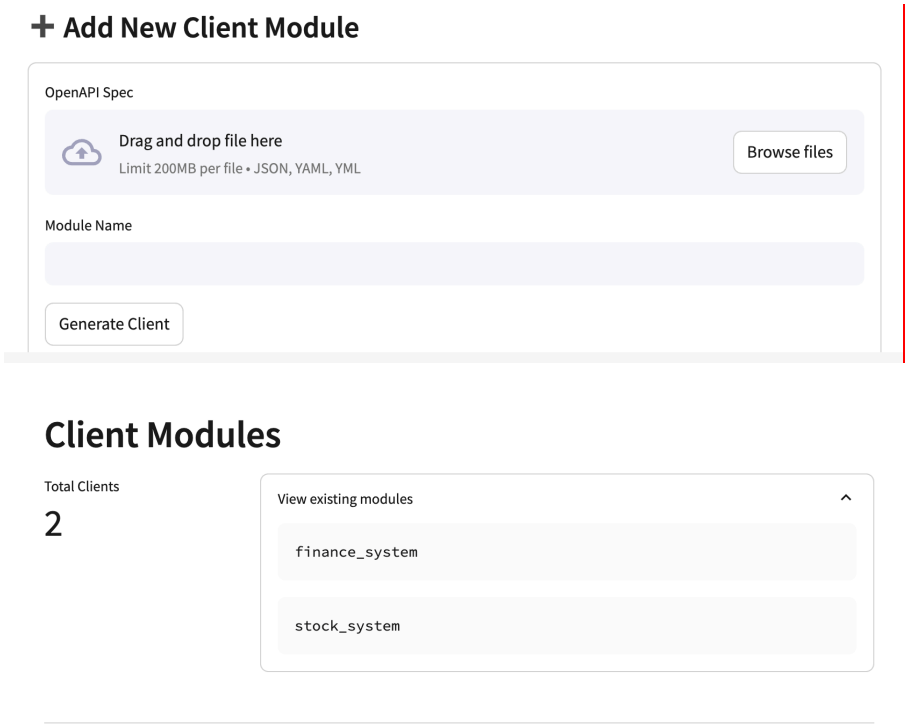


Figure 46 DevPortal Interaction for Generating and Assigning Client Modules

The process begins with what the framework formalizes as UC3 – Generate Client, which constitutes the first step in transforming an abstract specification into a usable integration component. As depicted in Figure 47, and mentioned before, the develper uploads or provides the OpenAPI document through the DevPortal.

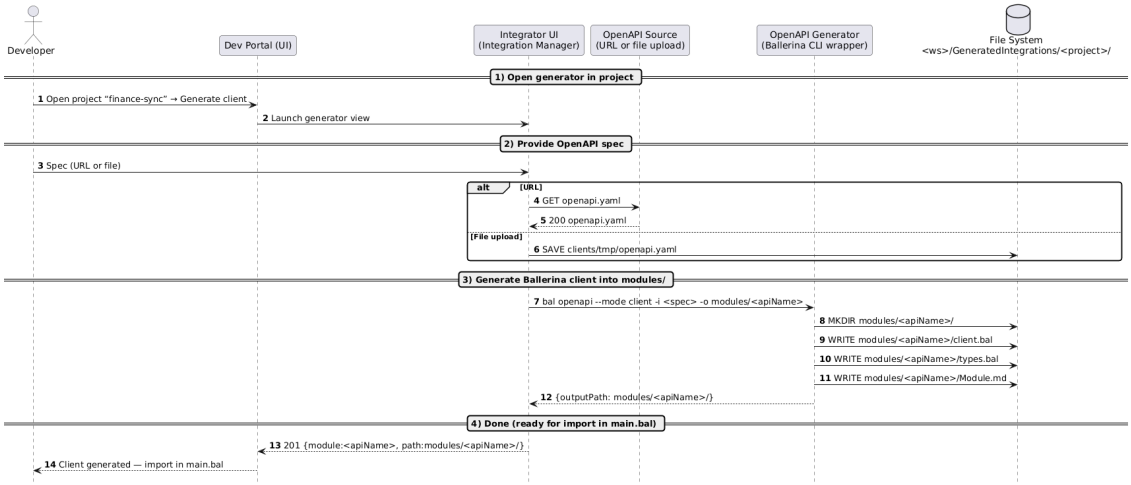


Figure 47 UC3 Sequence Diagram– Generate Client

From there, the Integration Manager invokes the generation pipeline—in the prototype, the ballerina openapi export command—which interprets the specification and produces a strongly typed client module.

Each endpoint listed in the specification is translated into callable methods, complete with explicit parameters and return types. This approach, already detailed in Chapter 5 (cf. Section 5.6), eliminates the risks of manual request construction or payload handling, embedding compile-time safety and semantic fidelity into the generated code. Architecturally, UC3 stabilizes the dependency surface, since consumers interact with the library rather than with brittle ad hoc requests, and it promotes reuse: a single client module can support multiple integrations and remain compatible with diverse engines, consistent with the framework's engine-agnostic vision outlined in Chapter 4.

Architecturally, leading with UC3 has two benefits. First, it stabilizes the dependency surface: consumers code against a generated library that is traceably derived from the published contract. Second, it promotes reuse: a single client module can serve multiple integrations and even multiple engines, consistent with the framework's engine-agnostic stance (cf. Chapter 4).Any language/toolchain capable of producing an API client is acceptable; the prototype's Ballerina implementation is provided merely as a concrete instantiation.

Importantly, the framework remains deliberately technology-agnostic. While the prototype employs Ballerina due to its alignment with integration-centric development, the same lifecycle could be achieved using Java, Python, or Node.js toolkits capable of generating API clients. This reinforces the framework's architectural principle of neutrality, ensuring that integration artifacts are not bound to a single runtime or programming paradigm but instead conform to a uniform methodology of contract-first onboarding.

By the end of this step, the framework has established a disciplined environment in which services are represented as standardized, reusable clients. This environment constitutes the foundation upon which later orchestration occurs, bridging the conceptual requirements outlined in Chapter 4 with their concrete operationalization in Chapter 5, and preparing the ground for the more complex synchronization workflows examined in the practical scenario.

6.4 Step 3 – Creating the Integration

Once the necessary client modules have been prepared, the next step consists of creating a new integration that orchestrates these modules into a coherent flow. This creation process can be initiated in two complementary ways: through the DevPortal interface or directly from the Ballerina Integrator UI editor. Both approaches converge toward the same outcome: the instantiation of a new integration project within the generated_integrations directory, preconfigured with the client modules retrieved in the previous step. Figure 48 illustrates the DevPortal screen for creating a new integration. Here, the user specifies a project name (for example, productname_integration) and selects *Generate Integration*.

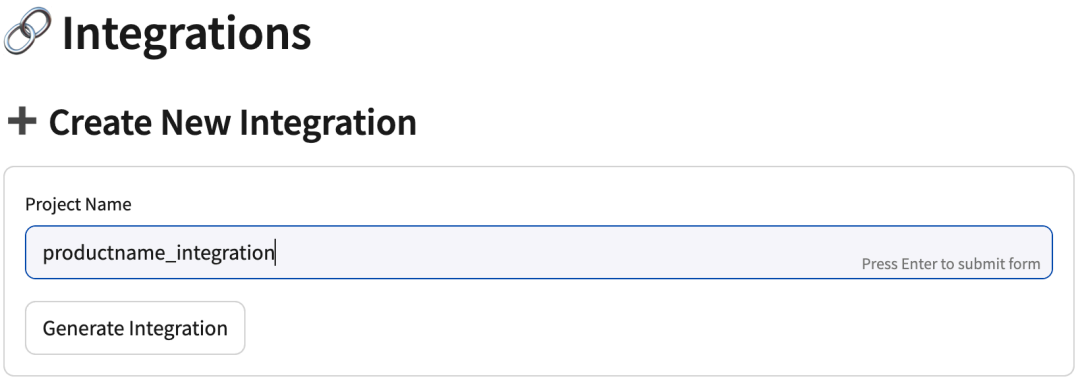


Figure 48 DevPortal Create Integration Interface

Only after the generation of the client module does the framework move to UC2 – Create Integration Project, captured in Figure 49. At this point, the DevPortal scaffolds a new project within the integration workspace, complete with canonical layout, metadata files, and a runtime entry point. This scaffold does not yet contain any domain logic; instead, it provides the structural substrate into which the previously generated client module can later be injected. Client generation is a contract-driven activity that can be repeated whenever an upstream API evolves, while project scaffolding prepares the operational container, with packaging, triggers, and deployment hooks. This sequencing enables lifecycle decoupling: clients may evolve and be regenerated independently of the projects that consume them, while integration projects remain stable units of deployment within the framework's runtime environment.

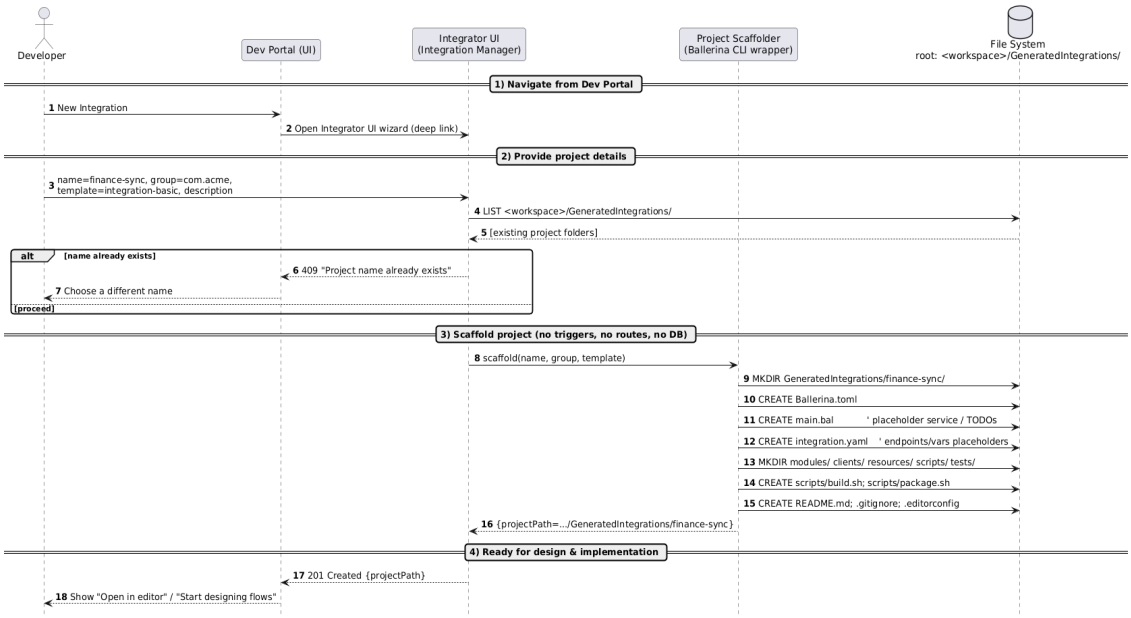


Figure 49 UC2 Sequence Diagram – Create Integration Project

What follows is the transition from structural preparation to functional design. The scaffold generated in UC2 serves as the foundation upon which orchestration logic can now be implemented. This next stage, formalized as UC4 – Develop Integration, elevates the bare project into an executable integration unit. Here, the framework captures the process of scaffolding and implementing the orchestration logic. The DevPortal initiates the project structure, injects the previously generated client modules, and provisions a minimal main.bal file with an HTTP listener bound to the project name. This ensures that each integration is both a piece of executable logic and an identifiable service within the runtime. The sequence diagram (Figure 50) illustrates the DevPortal-assisted scaffolding, the injection of client modules, and the initialization of the listener that exposes the integration as a service.

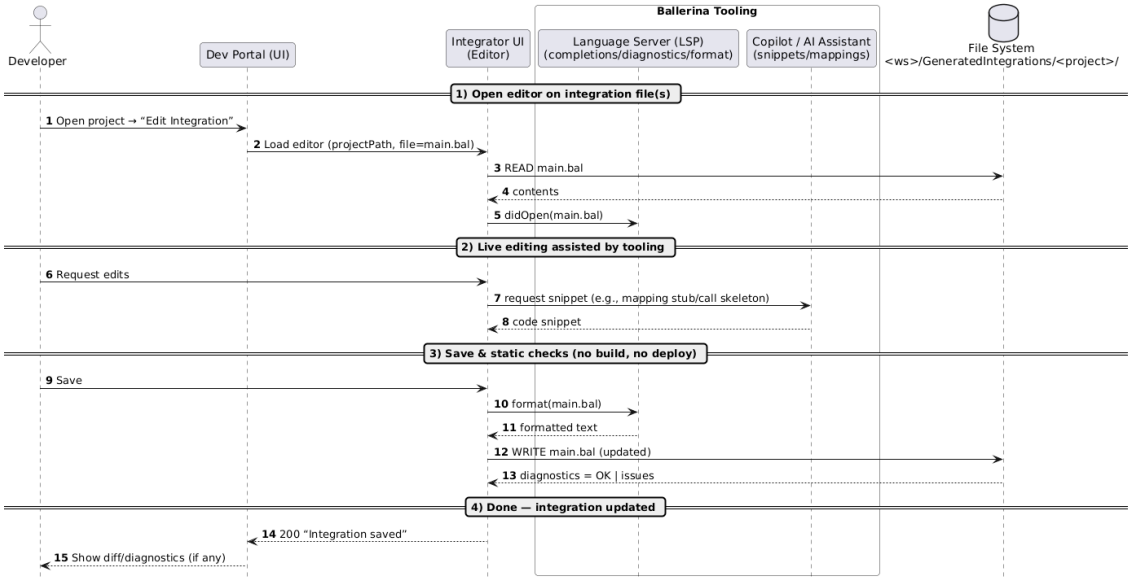


Figure 50 UC4 Sequence Diagram – Develop Integration

The generated scaffold is then made available for inspection and extension within the Ballerina Integrator editor, as illustrated in Figure 51. Here, the developer is presented with a familiar development environment, equipped with the client modules injected by the DevPortal and with the minimal listener serving as an entry point. The editor thus becomes the arena in which organizational logic is expressed—whether through direct coding, reuse of pre-generated clients, or higher-level orchestration patterns—while simultaneously remaining aligned with the architectural commitments outlined in Chapters 4 and 5.

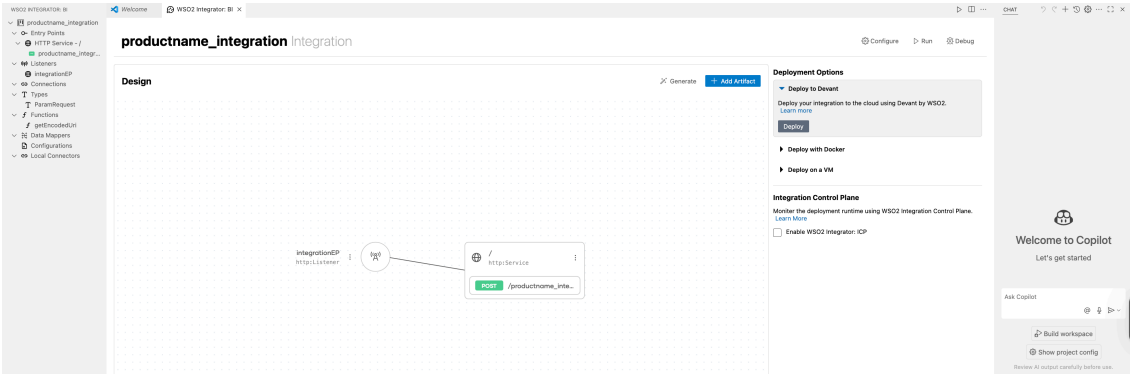


Figure 51 Ballerina Integrator Editor

Beyond manual development, the framework also integrates intelligent assistance through Ballerina Copilot, as represented in Figure 52. This AI-supported coding companion leverages its contextual awareness of available client modules and their declared endpoints to accelerate the implementation process. Developers may issue natural language prompts, such as *"synchronize product name changes between Finance and Stock services"*, which the Copilot then translates into executable Ballerina logic. This augmentation illustrates the dual philosophy underpinning the framework: on the one hand, strict adherence to contract-first governance, and on the other, a pragmatic embrace of productivity-enhancing tools. By embedding Copilot into the development workflow, the framework demonstrates how AI assistance can coexist with, and even reinforce, methodological discipline by reducing boilerplate, minimizing human error, and expediting integration design.

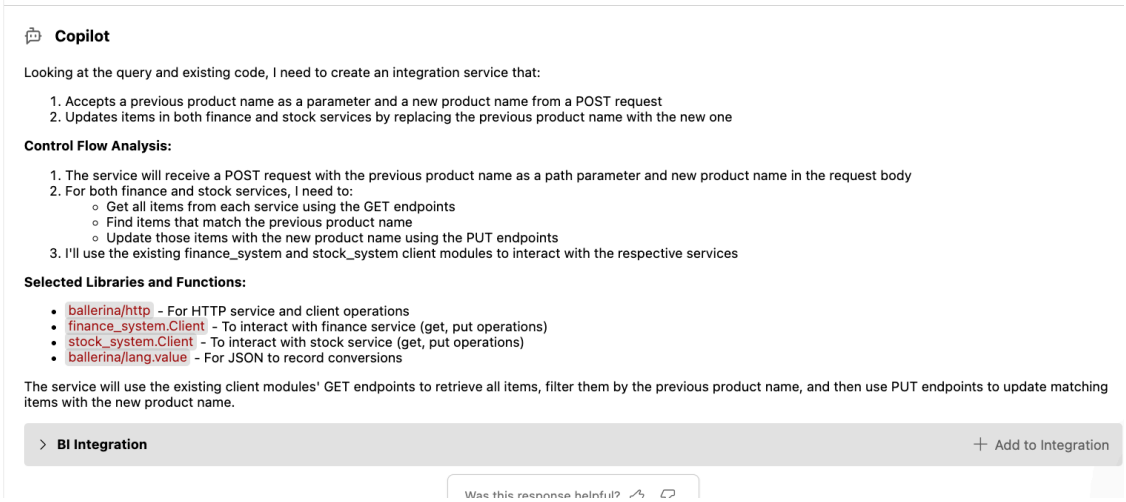


Figure 52 Ballerina Copilot Interface

The results of these interactions are consolidated in the state represented in Figure 53, where the view captures the integration after Copilot intervention and developer refinement. At this point, the integration is no longer a skeletal scaffold but a fully operational module, complete with orchestration logic that invokes multiple clients, processes inputs, and exposes outputs through the listener. The framework thus transforms an abstract specification into a tangible,

executable integration—one that is not only technically viable but also aligned with organizational governance and operational requirements. This final state validates the framework's capacity to merge automation, human agency, and AI assistance into a coherent lifecycle, demonstrating its robustness as a platform for sustainable enterprise integration.

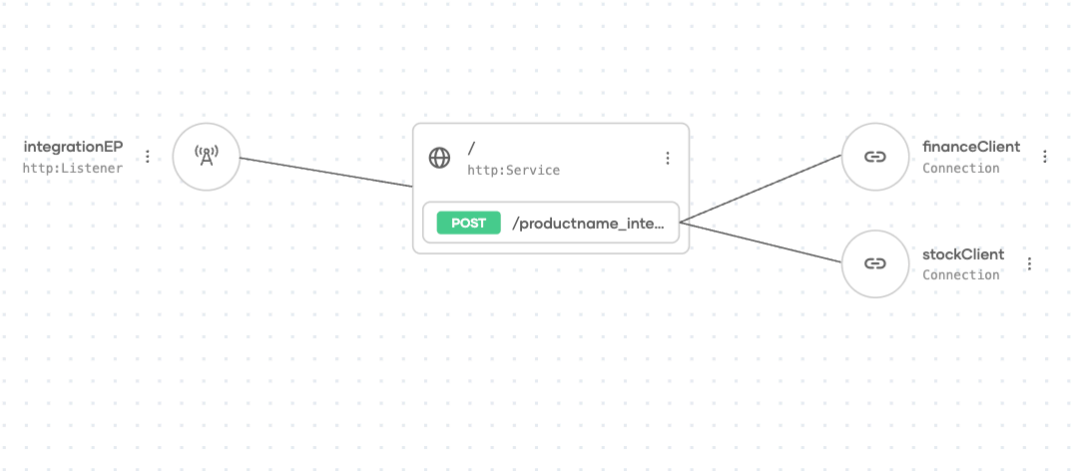


Figure 53 Integration Result

6.5 Step 4 - Validation

Once the integration project is scaffolded and initialized, it must transition from a static code artifact into a fully executable service. This moment marks the passage from design to runtime validation, ensuring that the integration is not only structurally sound but also functionally reliable. Within the framework, this transition is carefully staged so that integrations can operate in two distinct modes: as synchronous programs invoked directly through the command line, or as persistent services that expose their own HTTP listeners within the Serverless Services layer. The second mode, which reflects the target operational model, allows integrations to run continuously, respond to multiple invocations, and interoperate seamlessly with other components of the enterprise environment. In this way, the framework embodies the architectural intent articulated in Chapter 4, namely the establishment of integrations as autonomous yet governed services within a distributed ecosystem.

To facilitate this passage from static artifact to active service, the framework relies on the resources offered by the Ballerina platform. As outlined in Chapter 5 (cf. Section 5.6), the platform provides not only a textual programming interface but also a set of visualization tools that render the integration logic in diagrammatic form. This feature is particularly valuable in the validation stage, since it transforms what might otherwise be an opaque block of source code into a set of accessible and verifiable models.

Two types of diagrams are of particular relevance here. The sequence diagram captures the temporal ordering of interactions: it shows the requests that the integration issues to external services, the responses it receives, and the dependencies that structure this dialogue. In

contrast, the flow diagram abstracts away from the temporal dimension to present the orchestration as a process-centric path, illustrating the main logical steps and their interrelations. Taken together, these diagrams provide both engineers and analysts with complementary perspectives: one that emphasizes control flow, and another that highlights orchestration design. Figure 54 documents these visualizations, demonstrating how the integration is conceptualized before it is executed. These figures do not simply accompany the implementation; they constitute an essential part of the validation process, offering a preliminary confirmation that the orchestration is internally coherent and aligned with the intended specification.

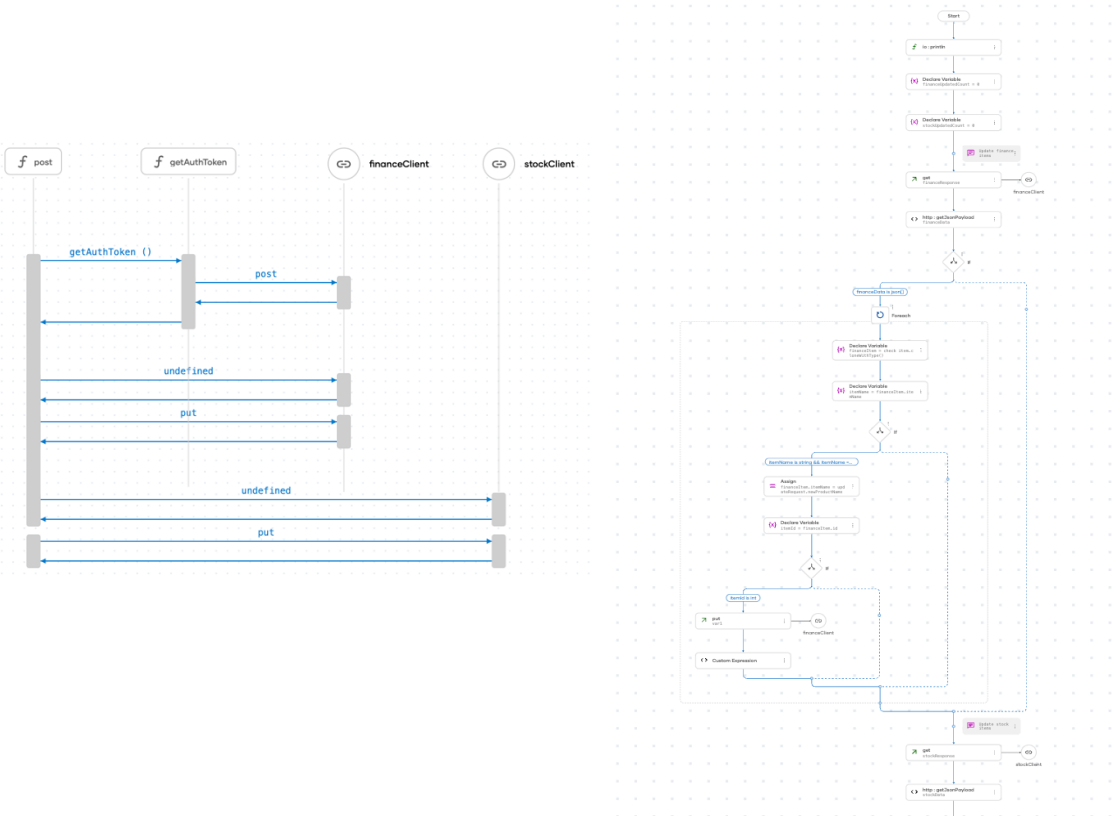


Figure 54 Integration Sequence and Flow Diagrams

Once this visual validation has taken place, the framework proceeds to UC5 — Execute Integration locally, which represents the first concrete test of the integration in a controlled, local environment. At this stage, the integration is compiled and executed through the commands bal build && bal run, which transform the source code into a runnable artifact and activate the associated HTTP listener.The presence of this listener signals that the integration has become an active service, capable of receiving requests and orchestrating updates across heterogeneous systems.

The local execution path is depicted in Figure 55. The diagram illustrates the entire validation loop: beginning with the submission of a test request to the integration listener, continuing with the orchestration of updates across the Finance and Stock services, and culminating in the

delivery of a confirmation response. This graphical representation thus serves as both a technical specification and a validation tool, showing at a glance how the integration behaves once instantiated.

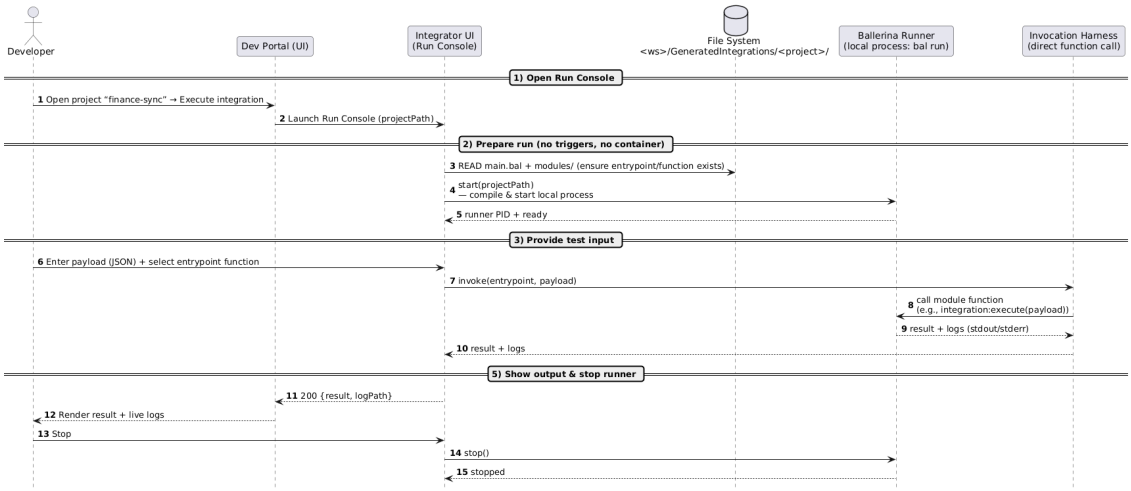


Figure 55 UC5 Sequence Diagram – Run Locally

At this stage, validation proceeds through a structured sequence of interactions designed to confirm the framework's capacity to manipulate data consistently across heterogeneous systems. The first step involves the establishment of a baseline state. A direct query to the Finance Service, shown in Figure 56, returns two records, one of which corresponds to the entry labelled **"**Widget A**"**. Within the finance domain, this record is represented as a FinanceItem, an entity type used for invoicing, expense tracking, and financial reporting. Its presence in the dataset establishes a clear point of reference against which the integration process can later be validated.

```
Get https://project-financeservice-latest.onrender.com/api/finance
[
    {
        "id": 1,
        "itemName": "Office Supplies",
        "itemType": "Expense",
        "amount": 123.45,
        "timestamp": "2025-08-06T14:51:20.430506"
    },
    {
        "id": 2,
        "itemName": "Widget A",
        "itemType": "Expense",
        "amount": 123.45,
        "timestamp": "2025-08-06T14:51:38.693802"
    }
]
```

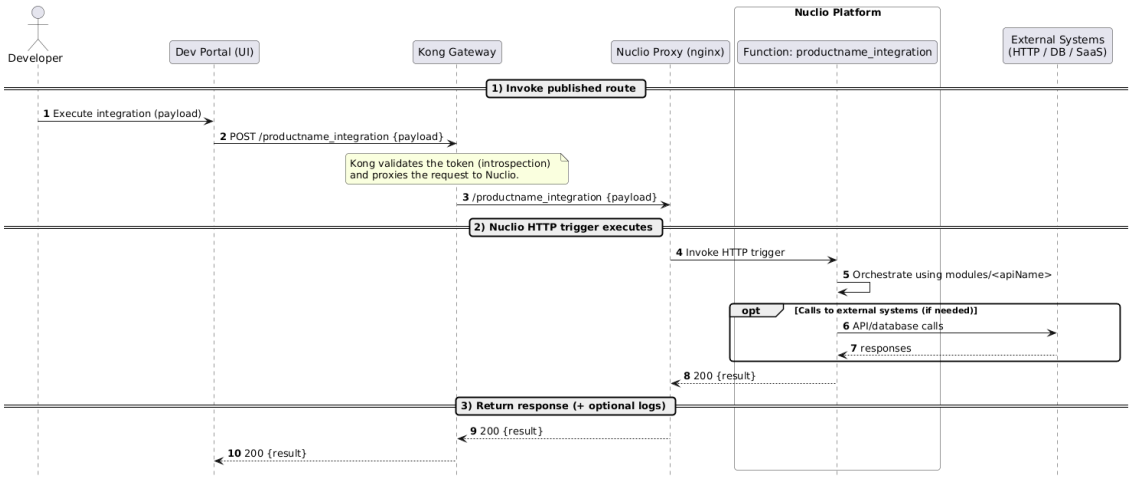
Figure 56 FinanceItem Object Baseline

To reinforce the baseline, a subsequent request is submitted to the Stock Service, as represented in Figure 57. The response again contains the record *"Widget A"*, but here it is represented as a StockItem**,** an entity used for inventory control and warehouse management. Despite the difference in attribute naming and domain-specific semantics, the two entries point to the same real-world object. This semantic equivalence provides the necessary condition for synchronization: the framework must ensure that an update applied in one service is propagated to the other, despite their structural heterogeneity.

Figure 57 StockItem Baseline

Once the baseline state has been confirmed across the Finance and Stock services, the framework proceeds to the execution phase, formalized as UC5 – Execute Integration. Unlike the preliminary validation carried out in the local development environment, this stage corresponds to the execution of the integration within the core runtime of the framework, where the service is exposed through the Kong Gateway and subsequently routed to the Nuclio Platform. This distinction is critical: while local execution validates the syntactic and functional correctness of the integration, core execution ensures that the orchestration logic operates reliably when embedded in the distributed infrastructure of the framework.

Figure 58 illustrates this process. The developer initiates the execution by submitting a POST request to the gateway endpoint associated with the integration (`http://localhost:8080/productname_integration`). The request payload contains two fields—`previousProductName` and `newProductName`—explicitly instructing the framework to identify the record *"Widget A"* across both domains and replace it with *"Integration Success"*. Once the request enters the gateway, it traverses the proxy layer, is dispatched to the Nuclio function that encapsulates the integration logic, and from there triggers updates across the heterogeneous backends. The diagram captures this flow step by step, from the initial user invocation to the coordinated updates and finally the delivery of a structured confirmation response.




```
Post : http://localhost:8080/productname_integration
{"previousProductName":"Widget A","newProductName":"Integration Success"}

Response:
{
    "message": "Successfully updated product name from 'Widget A' to 'Integration Success'",
    "financeItemsUpdated": 1,
    "stockItemsUpdated": 1
}
```

Figure 58 UC5 Sequence Diagram -Execute Integration and Integration Request Payload

After the execution of the integration within the framework, a final verification was performed against the baseline that had been previously established. This comparison ensures that the orchestration not only executed correctly but also achieved the intended semantic alignment between the Finance and Stock services. As shown in Figure 59, both services now reflect the updated value *“Integration Success”*, confirming that the change was consistently propagated across heterogeneous domains.


```
Finance Server-> [
    {
        "id": 1,
        "itemName": "Office Supplies",
        "itemType": "Expense",
        "amount": 123.45,
        "timestamp": "2025-08-06T14:51:20.430506"
    },
    {
        "id": 2,
        "itemName": "Integration Success",
        "itemType": "Expense",
        "amount": 123.45,
        "timestamp": "2025-08-06T14:51:38.693802"
    }
]

Stock Server-> [
    {
        "id": 1,
        "itemName": "Office Supplies",
        "itemType": "Expense",
        "amount": 123.45,
        "timestamp": "2025-08-06T14:51:20.430506"
    },
    {
        "id": 2,
        "itemName": "Integration Success",
        "itemType": "Expense",
        "amount": 123.45,
        "timestamp": "2025-08-06T14:51:38.693802"
    }
```


Figure 59 Final Verification

6.6 Step 5 – Deploying the Integration

Once the integration logic has been implemented and validated locally, the next milestone is deployment. In the designed framework, deployment is not a trivial matter of making the code executable, but rather a carefully orchestrated process that encapsulates the integration in a secure and manageable runtime environment. This stage ensures that the integration is not only functional but also resilient, discoverable, and protected from unauthorized access.

In the Actor–Goal Use Case Diagram mentioned before, this stage corresponds primarily to UC1 — Publish Integration, which formalizes the automated pipeline described in detail in Chapter 5 (cf. Section 5.6.4). From the perspective of the developer, the act of publishing is reduced to a single command in the DevPortal. Behind the scenes, however, this action triggers a chain of infrastructural tasks: the integration is compiled into a Docker image, pushed into the internal registry, deployed as a Nuclio function, and finally registered through Kong so that requests can reach it via the secure gateway. The pipeline is captured in Figure 60, which depicts the end-to-end orchestration of these actions, consolidating multiple infrastructural components into a streamlined developer-facing experience.

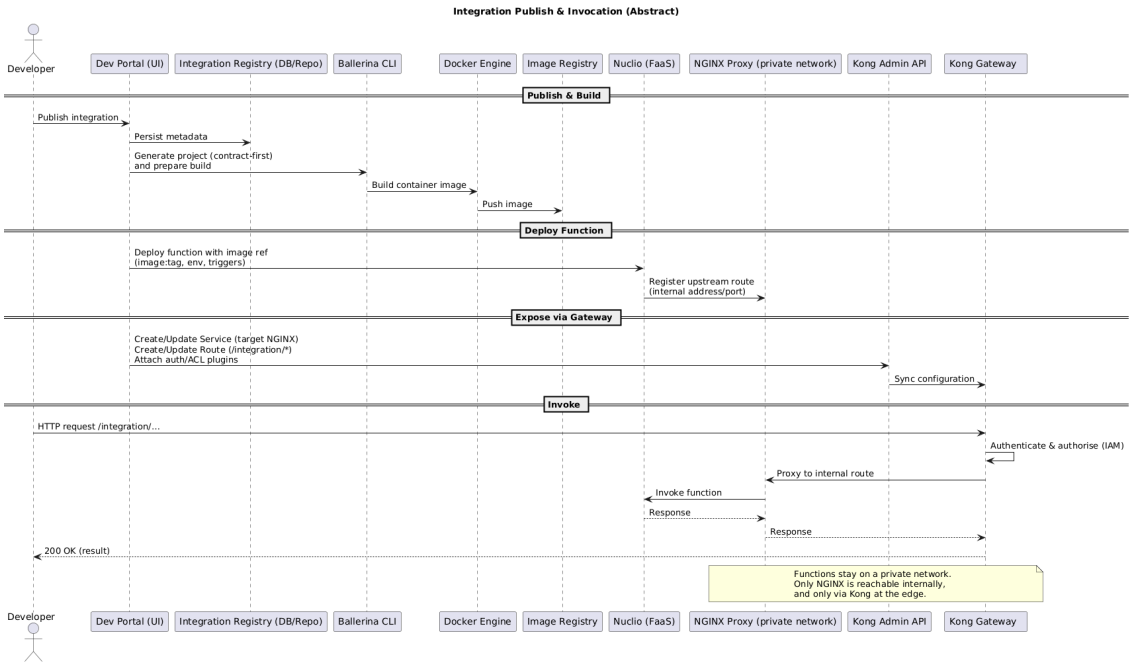


Figure 60 UC1 Sequence Diagram - Publish Integration

The DevPortal thus plays a crucial role in abstracting complexity. As depicted in Figure 61, the interface provides a dedicated option for publishing an integration, ensuring that the developer need only select the target project and confirm the action. This single interaction belies the depth of infrastructural activity it triggers. The moment the *Publish* button is pressed, the pipeline described in Chapter 5 is executed in its entirety, ensuring that the integration is packaged, deployed, and registered in a manner consistent with the framework's architectural principles. The reference to Chapter 5 is particularly important here, as it was in that section

that the internal mechanics of the pipeline were explained in detail, including the compilation process, Docker image construction, registry push, and deployment within Nuclio's private runtime environment.

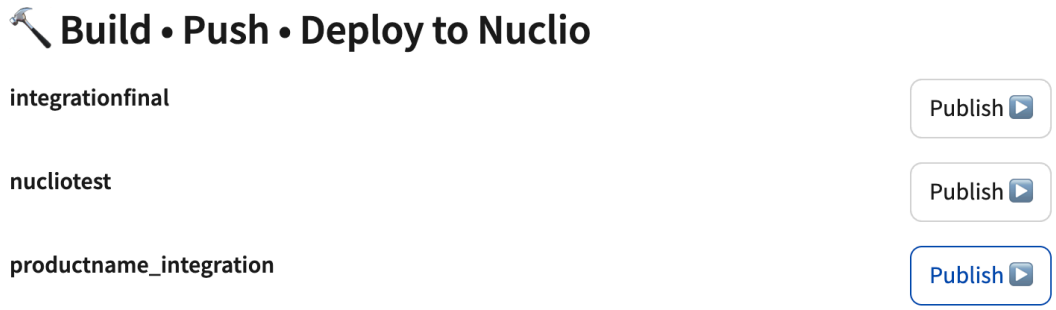


Figure 61 DevPortal Publish Integration Interface

The automation does not stop at packaging and deployment. Once the function is successfully instantiated, the framework formalizes its accessibility by registering the service within the gateway. This activity corresponds to UC8 — Register Integration Route (Auto). From the developer's standpoint, the effect is subtle: the integration simply appears as an entry in the Integration Catalog. Yet, in the background, the Kong Admin API is programmatically invoked to generate a route that fronts the deployed function. The sequence diagram in Figure 62 documents this automation, showing how the framework transparently integrates new services into the existing governance model. Thus, deployment is never left incomplete; every published integration is immediately routable through Kong, with access mediated by authentication and policy enforcement.

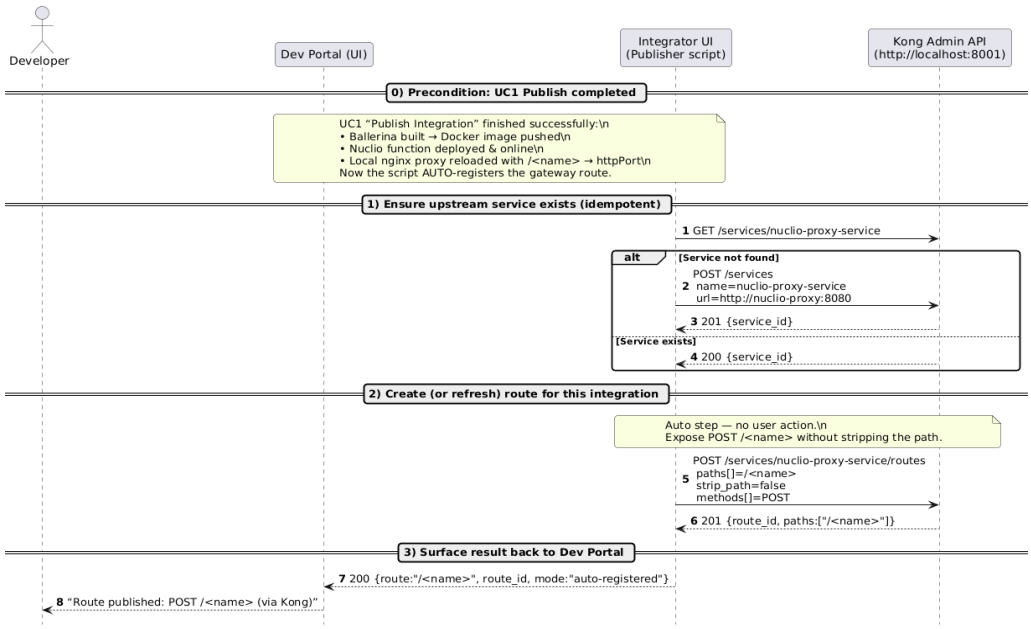


Figure 62 UC8 Sequence Diagram - Register Integration Route on Gateway

Nevertheless, the framework also acknowledges that automated pipelines cannot always anticipate every operational contingency. For this reason, a manual fallback is available in UC7 — Create Route Paths (Manual). In such cases, administrators can rely on the Admin UI to directly interact with the Kong Admin API, creating or adjusting routes in scenarios where automated registration is insufficient. Figure 63 illustrates this manual pathway, which is not intended as the norm but as a necessary operational safeguard. Its existence reflects an architectural commitment to flexibility: while automation is privileged to reduce human error and guarantee consistency, the framework still accommodates manual intervention when infrastructure demands it.

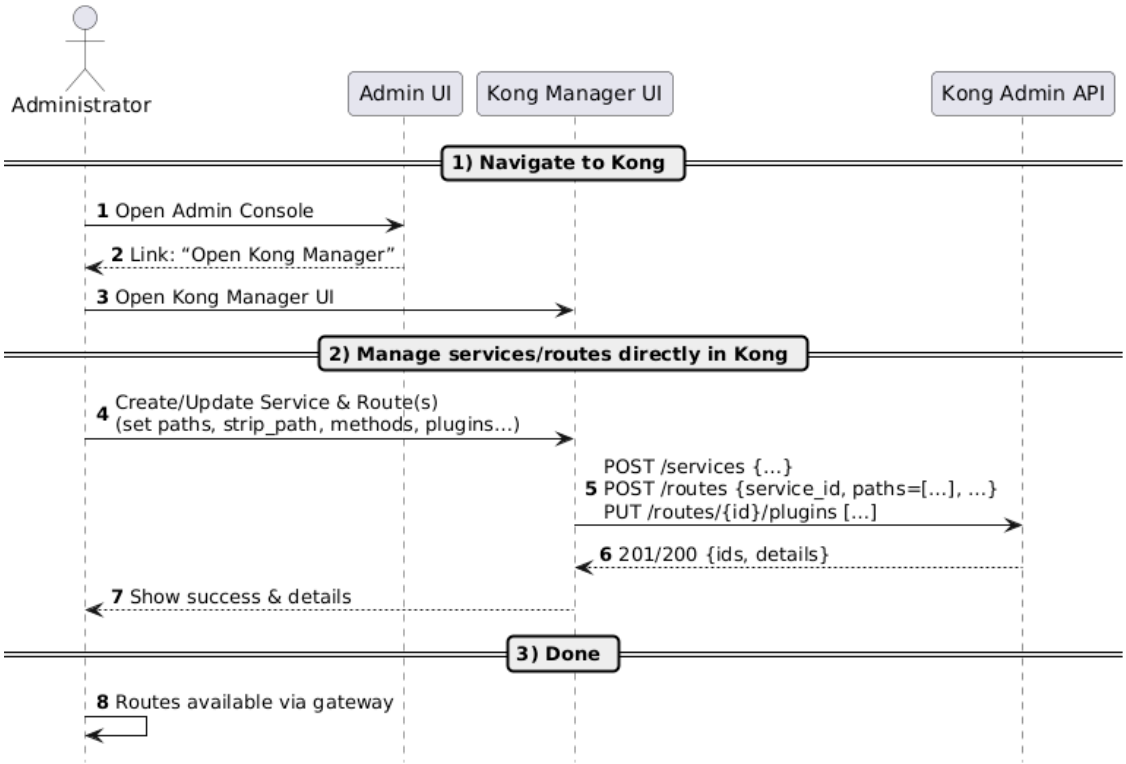


Figure 63 UC7 Sequence Diagram - Create Route Paths on Gateway

Deployment, however, is not concluded at the moment routes are registered. The final dimension of this stage is governance, which extends beyond containerization and routing to the interface of the DevPortal itself. Once a function is published and its routes refreshed through Kong and Nginx, it becomes visible in the Integration Catalog. This catalog is not merely a listing, but a governance artifact: each entry is automatically generated from metadata captured during deployment, including the integration name, associated routes, and pipeline status logs. In this way, the catalog simultaneously functions as documentation, an operational register, and an access control surface.

Governance is further reinforced through the Administration Manager, which binds integration routes to enterprise roles. When a new route is created, it is surfaced in the portal and can be explicitly associated with Keycloak roles. This role-based authorization model ensures fine-

grained and contextual access control, aligning operational reality with enterprise security policies. Importantly, access checks are enforced at runtime: whenever a request is made to the Kong-managed endpoint, the associated JWT token is verified, and only if the correct role is present will the request be forwarded to the Nuclio function.

Figure 64 shows an example where the integration route /productname_integration has been published, and specific roles are assigned to control its accessibility. This duality of automation and human-driven governance ensures that integrations remain secure, contextual, and aligned with enterprise policies.

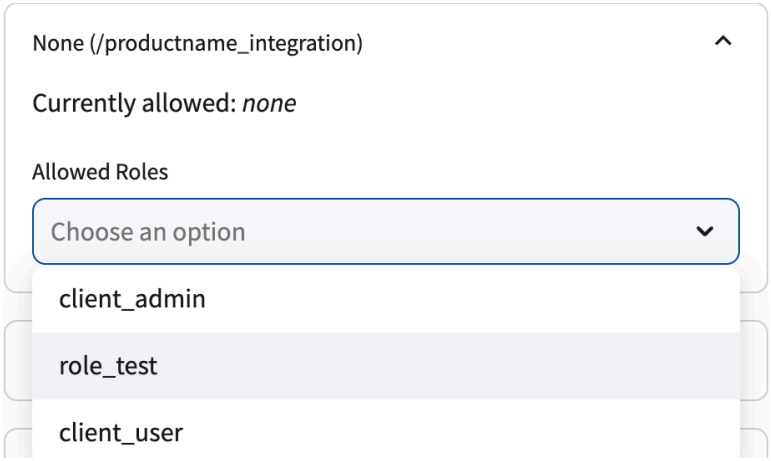


Figure 64 Adminsitration Manager Interface for Role Assignment

To complement this access control model, the framework integrates a lightweight monitoring capability that enhances transparency and operational assurance. This functionality is formalized as UC9 — Visualize Service Health Checks. Instead of presenting developers with full observability stacks, the framework offers a pragmatic view of essential service availability: each registered route is displayed with its corresponding URL, current status (UP/DOWN), response latency, and any connection anomalies. As shown in Figure 65, this mechanism allows administrators to verify at a glance whether critical services—such as the Core Backend, Process Execution Manager, or specific Nuclio functions—are active and properly proxied. By co-locating this functionality within the DevPortal, the framework reduces operational friction and ensures that both governance and availability monitoring are consolidated into a single environment.

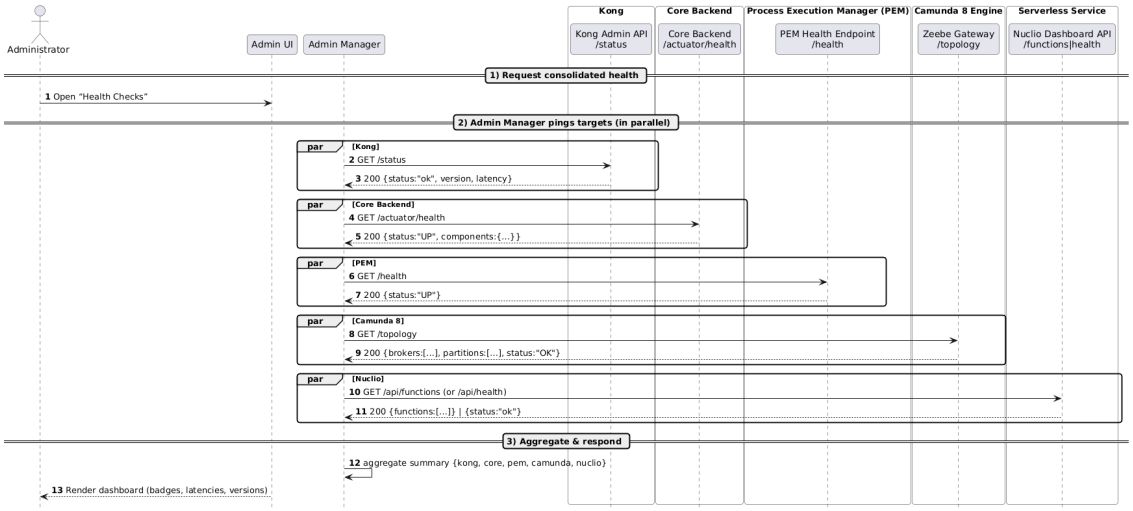


Figure 65 UC9 Sequence Diagram - Visualize Service Health Checks

Building upon this health view, the governance layer extends to runtime auditing through UC10 — Visualize Logs & Metrics. This use case enables administrators and developers to consult gateway logs, Nuclio function logs, and basic operational metrics directly within the portal interface. The capacity to inspect logs and metrics at runtime provides not only diagnostic support in the event of failure but also transparent evidence that policy enforcement is operating as intended. Figure 66 exemplifies this feature, showing how logs are surfaced as part of the same governance console that manages deployment and authorization. By integrating logging and deployment visibility into a unified cockpit, the framework ensures that operational transparency becomes an inherent property of every integration.

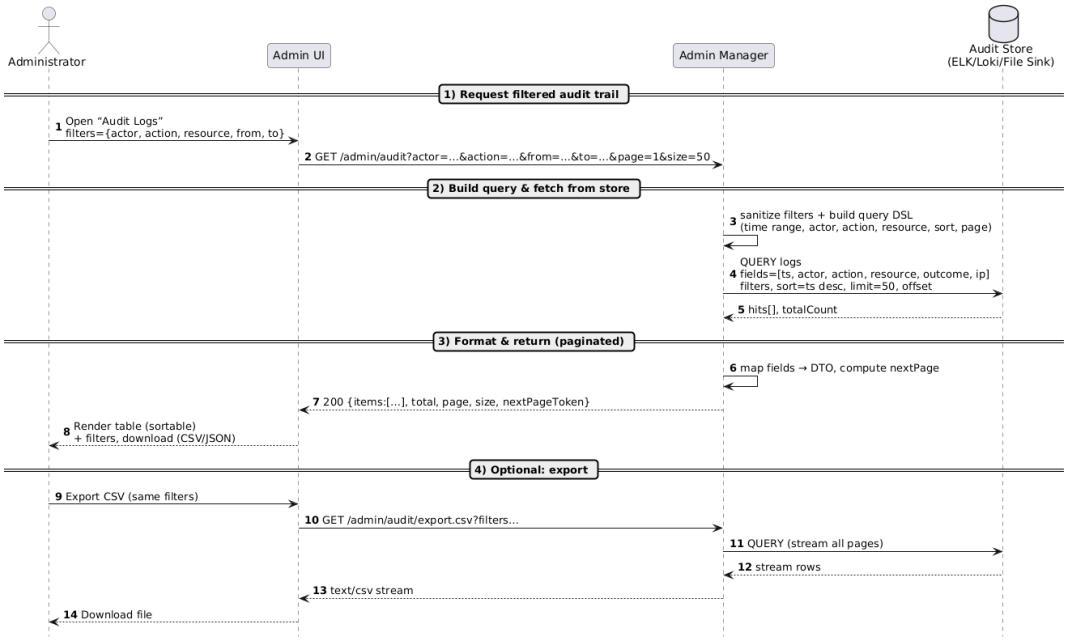


Figure 66 UC10 Sequence Diagram - Visualize Logs & Metrics

The integration of logs and portal visibility also provides transparency. Administrators can inspect the deployment outputs directly in the portal, confirming that the build, push, and deployment steps succeeded, and that Kong's routes are in sync. In this way, the Developer Portal becomes the operational cockpit for integrations: not only listing them as catalogued assets but also binding them to security policies that govern their execution. Finally, deployment governance encompasses the discovery of triggers, a task captured in UC11 — Visualize HTTP Triggers. Through the DevPortal, the Trigger Controller queries Nuclio to retrieve the list of *HTTP triggers* associated with each published function as shown in Figure 67.

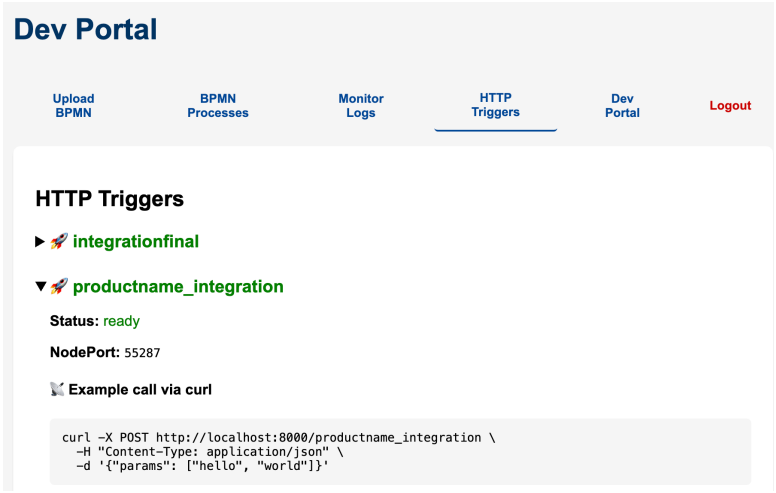


Figure 67 DevPortal HTTP Triggers Interface

Having examined the DevPortal interface as the visible point of interaction, it is now necessary to detail the underlying process that sustains what the user perceives. For this purpose, Figure 68 illustrates the UC11 flow and consequently the interactions that make the visualization of triggers possible. What appears on screen is not a static or pre-compiled record but the result of an orchestration that begins with the developer's action, continues through the DevPortal, and is delegated to the Trigger Controller. The controller then queries Nuclio's internal API, completing a request–response cycle that guarantees fidelity to the runtime state of deployed functions.

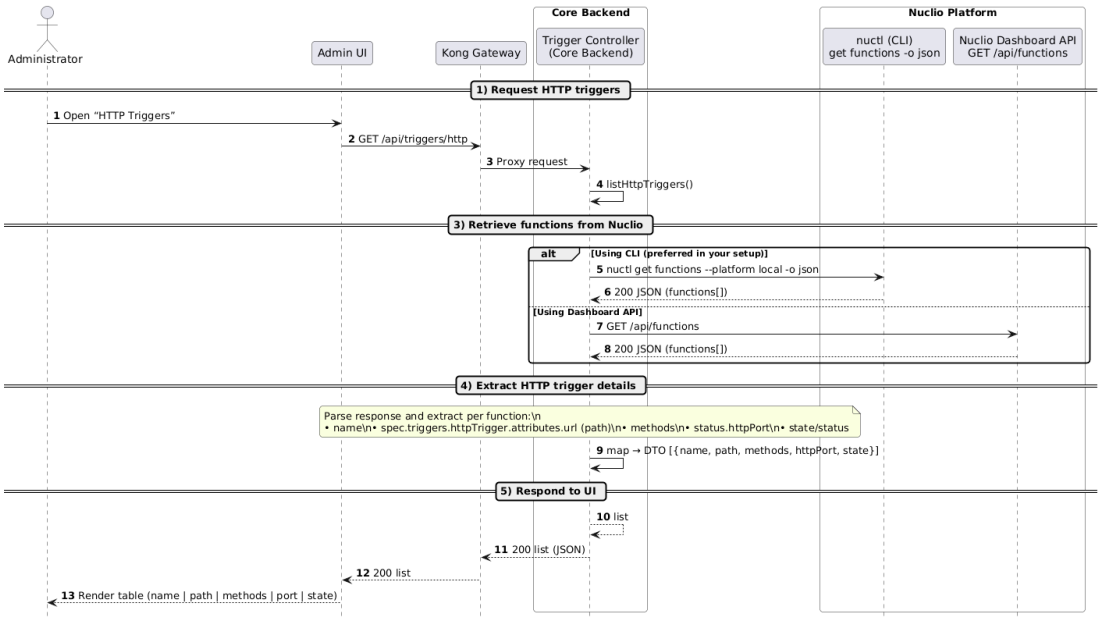


Figure 68 UC11 Sequence Diagram – Visualize HTTP Trigger List

6.7 Step 6 – Designing the Workflow (BPMN)

Once the integration is securely deployed and exposed through the gateway, the next milestone is the definition of workflows that orchestrate how integrations are invoked in practice. To achieve this, the framework relies on BPMN (Business Process Model and Notation) as the modeling standard. BPMN provides a structured yet human-readable way to describe process logic, ensuring that technical execution aligns with business semantics.

In the Actor–Goal Use Case Diagram mentioned before, this stage begins with UC16 — Configure Engine, the preparatory task that aligns the execution environment with the framework. Operationally, this configuration writes into system settings and applies changes through the Process Execution Manager (PEM), ensuring that subsequent deployments and invocations proceed without ad-hoc authorization cycles. This connects the workflow engine to the integration runtime *before* models are introduced (Figure 69). Conceptually, UC16 realizes at runtime the architectural responsibilities detailed in Chapter 5 —namely, centralized validation, deployment coordination, and adapter-based dispatch to the configured engine (e.g., Camunda 8/Zeebe). Where identity and policy are concerned, UC16 also presupposes the boundary enforcement described in Chapter 5 (cf. Section 5.1) and the engine-side identity arrangement in (cf. Section 5.2.1), so that the engine participates in the same Keycloak-governed trust domain as the remainder of the prototype.

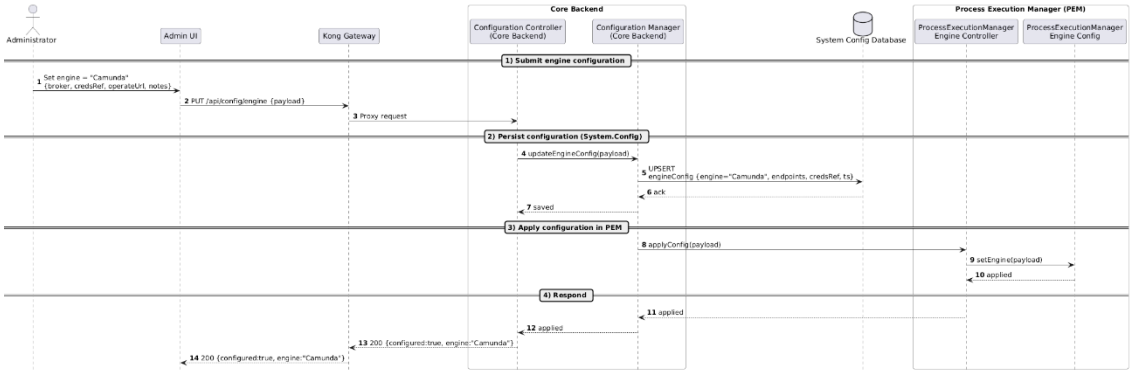


Figure 69 UC16 Sequence Diagram - Configure Engine

A key principle embedded in the framework is openness in the choice of modeling tools. The platform does not restrict users to a proprietary editor or to a single modeling environment. Instead, developers and analysts may rely on any BPMN-compliant modeler to design their processes—ranging from professional environments such as Signavio or Bizagi, to open-source editors like the Camunda Modeler, or even the lightweight browser-based modeler bundled with the execution engine itself as already addressed in Chapter 5. By maintaining strict adherence to the BPMN specification, the framework guarantees that diagrams authored in any of these tools can be imported, validated, and executed without additional transformations.

In the context of the practical scenario presented in this chapter, this openness is illustrated by the *praticalscenarioworkflow* process (Figure 70). The workflow models the synchronization of product names across heterogeneous services while embedding organizational oversight. The process begins with a managerial authorization task (*Authorize Product Name Change*), after which an exclusive gateway evaluates whether the update is approved. If rejected, the workflow terminates; if approved, the process advances to a human task (*Enter Product Names*), where an operator provides both the previous and new product name. The collected data is then forwarded to a service task (*Call ProductName Integration*), which invokes the previously deployed integration through the gateway. This integration ensures that the approved change is consistently propagated across the Finance and Stock services. By combining human oversight, data entry, and automated orchestration in a single BPMN model, the framework demonstrates its capacity to bind technical interoperability with organizational governance, confirming the real-world applicability of its architectural principles.

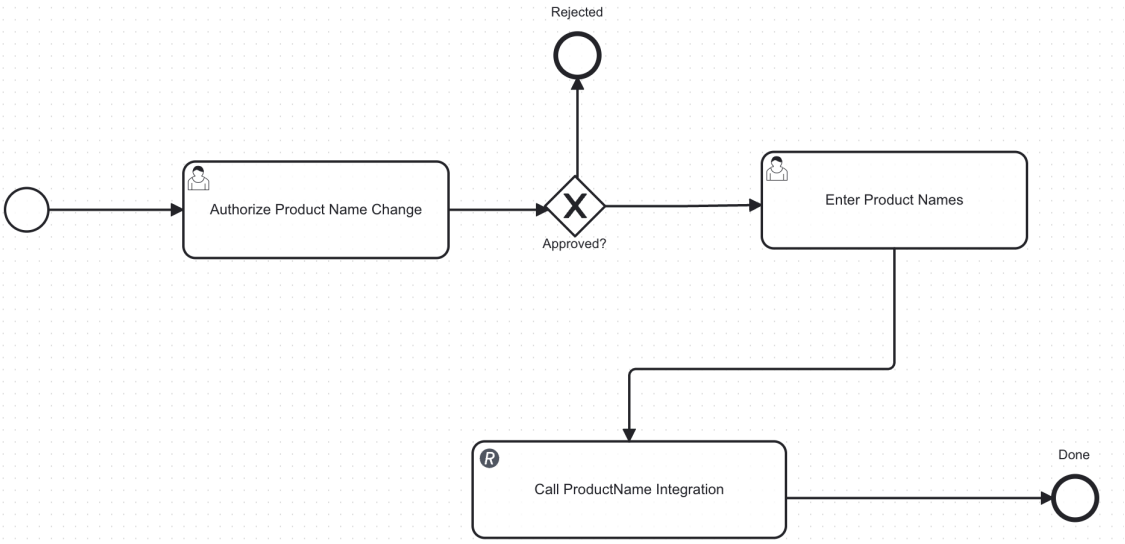


Figure 70 Pratical Scenario Workflow BPMN Workflow

At this stage, the developer or analyst interacts with the Dev Portal to upload the BPMN model. Figure 71 illustrates this workflow, where a BPMN file is uploaded using the graphical interface, and the diagram is immediately rendered for preview.

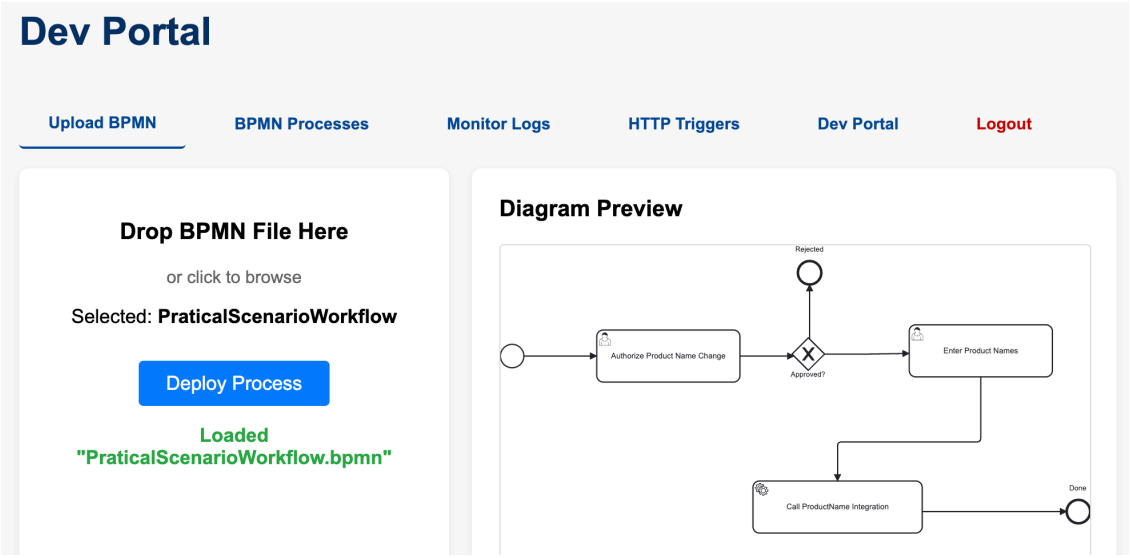


Figure 71 DevPortal BPMN Upload Interface

The uploading and deployment of a model is captured in UC12 — Upload/Deploy BPMN Model, which formalizes the cycle of submitting a process definition, validating its correctness, persisting it in the database, and updating its status to *deployed*. The sequence diagram (Figure 72) illustrates this tightly controlled workflow, highlighting how deployment transforms a static diagram into an active component of the framework.

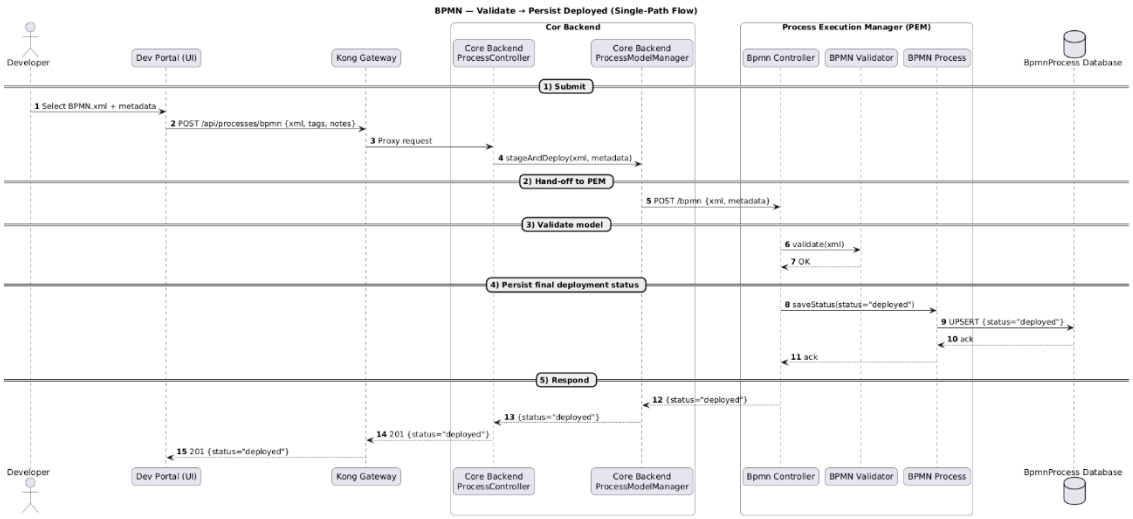


Figure 72 UC12 Sequence Diagram - Upload/Deploy BPMN Model

Once validated, the BPMN model can be deployed directly from the portal. Deployment transforms the BPMN definition into an executable workflow, registering it in the Process Manager and linking it with the underlying execution engines as addressed in the Chapter 5. The Dev Portal then exposes the status of the process, confirming that it is active and ready to receive input.

As shown in Figure 73, once the praticalscenarioworkflow process has been deployed, it is listed in the portal with metadata including its identifier, deployment status, and timestamp. From this interface, administrators can start, stop, or delete processes, providing complete lifecycle control over business workflows.

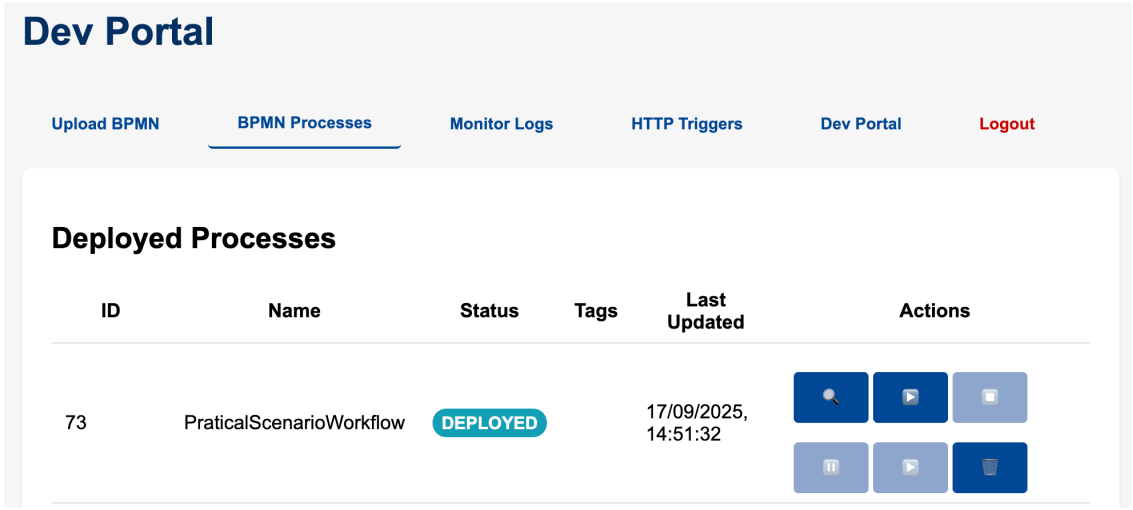


Figure 73 BPMN Models Manager Interface

This retrieval of deployed workflows is represented by UC15 — Check Deployed BPMN Processes, a use case that enables users to visualize the set of active processes available for

execution. The sequence diagram (Figure 74) captures this read-only interaction, emphasizing transparency and governance.

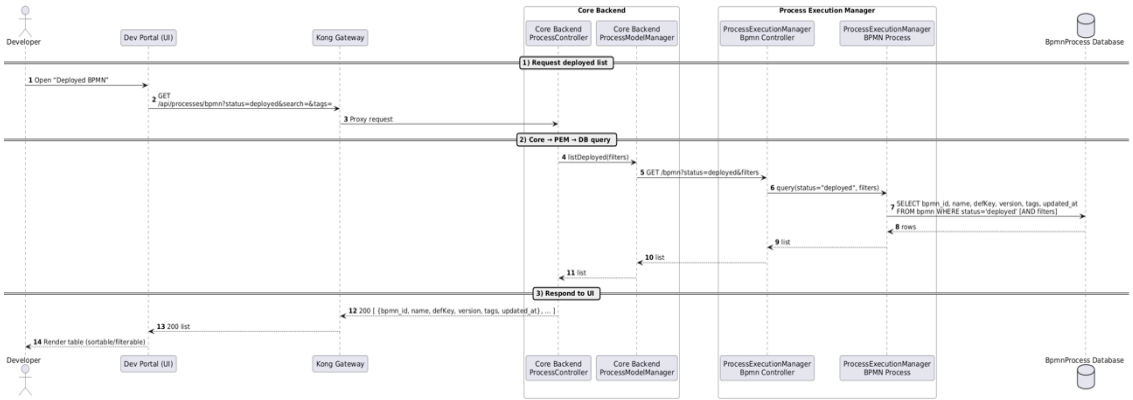


Figure 74 UC15 Sequence Diagram - Check Deployed BPMN Processes

Crucially, the Dev Portal does not conclude its role at the point of deployment. It extends its functionality by enabling administrators and testers to instantiate new process instances directly from the same interface. In doing so, it completes the methodological cycle that links modeling, deployment, and execution. A process can thus be authored in any BPMN-compliant tool, uploaded and deployed through the Dev Portal, and immediately subjected to validation in a live execution context. This continuity accelerates iteration cycles and allows both technical teams and business stakeholders to evaluate the operational consequences of process changes in real time, reinforcing the principle that orchestration must remain both transparent and verifiable.

The interoperability of this mechanism rests on a broader architectural principle: integrations are exposed as independent HTTP services and thereby rendered consumable by heterogeneous orchestration engines. The conceptual underpinnings and technical realization of this design are examined in detail in Chapter 5, (cf. Section 5.4.1). By encapsulating integration logic as REST-addressable services, the framework achieves strict technology neutrality, avoids reliance on proprietary connectors, and ensures that the same unit of logic may be invoked seamlessly across engines such as Camunda or Apache Airflow. In the context of the practical scenario, the invocation of the *productname_integration* within the *praticalscenarioworkflow* model demonstrates the operational materialization of this principle: the workflow remains portable across orchestration environments, while integration governance and access control remain consistently centralized.

6.8 Step 7 – Executing the Practical Scenario

Once a BPMN process has been deployed and registered in the framework, the next milestone concerns its execution within the practical scenario. This is the stage where abstract workflow definitions are instantiated as concrete activities, orchestrating the interplay between human actors and automated services. In the Actor–Goal Use Case Diagram mentioned in the chapter,

this execution phase begins with UC13 — Start Process Instance, which represents the creation of a new workflow instance in the engine. From the Tasklist interface, a user selects a deployed process such as *praticalscenarioworkflow* and initiates its execution. The corresponding sequence diagram (Figure 75) captures this interaction, showing how the request from the Tasklist results in the instantiation of a new process instance that transitions immediately into its first activity.

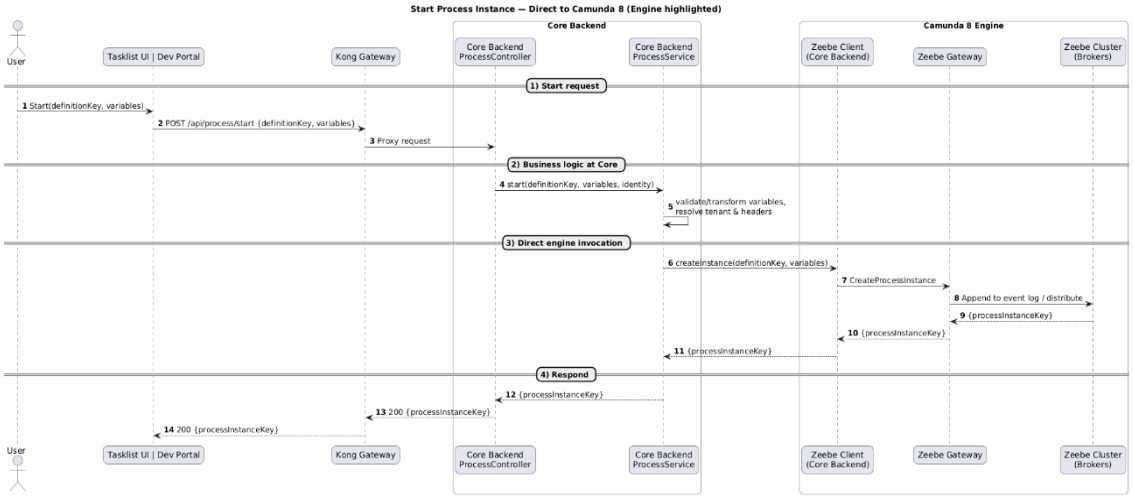


Figure 75 UC13 Sequence Diagram - Start Process Instance

Execution unfolds through the Tasklist, where the *praticalscenarioworkflow* process is instantiated and its sequence of activities triggered. The *praticalscenarioworkflow* BPMN model, discussed in the preceding step, structures execution as a combination of administrative validation, user participation, and automated orchestration. Its first activity, *Authorize Product Name Change*, introduces a managerial approval point, which requires a distinct role—typically an administrator or an actor with delegated authority—to evaluate whether the proposed modification may proceed, as explicit in Figure 76. By explicitly situating this task at the beginning of the workflow, the framework demonstrates its ability to embed organizational governance directly into technical processes, rather than treating such oversight as an external add-on. In practice, the execution path bifurcates at this decision: if approval is withheld, the workflow terminates; if approval is granted, the instance progresses to the next stage.

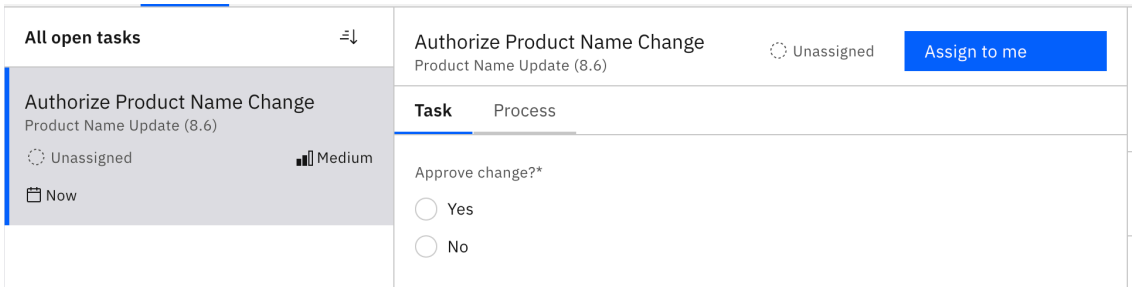


Figure 76 Tasklist Interface – Authorize Product Name Change

Following authorization, the workflow advances to the user-facing task *Enter Product Names*. Here, the engine generates a task entry visible to the relevant candidate group, which initially appears as *unassigned* and therefore open to claim. This sequence of assignment and completion embodies UC14 — Claim & Complete User Task, a use case that codifies the governance of responsibility and accountability in user interactions. The sequence diagram (Figure 77) captures this cycle, highlighting the temporary suspension of execution during human intervention and the subsequent resumption of the workflow upon task completion.

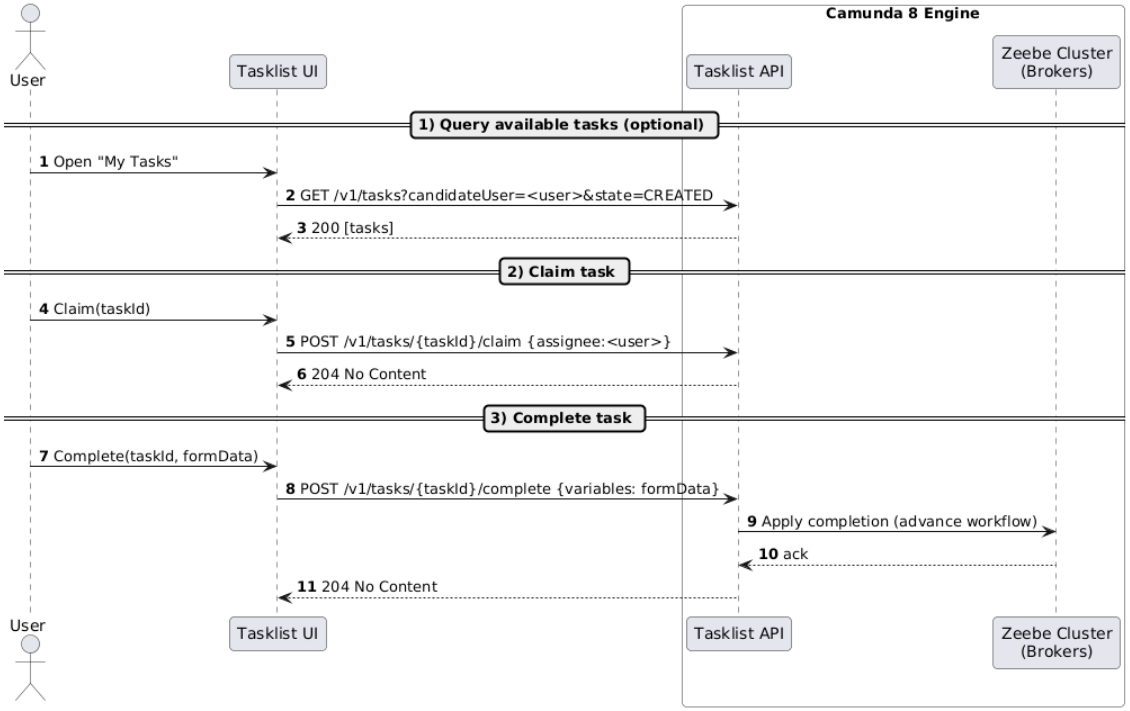


Figure 77 UC14 Sequence Diagram - Claim & Complete User Task

As mentioned, once claimed, the interface presents a structured form where the operator supplies the necessary inputs (Figure 78). This interaction exemplifies the human-in-the-loop orchestration capabilities of BPMN engines such as Camunda: the workflow suspends execution until a human actor intervenes, thereby ensuring that managerial oversight and operational precision are not bypassed in favor of blind automation. After submission, the engine resumes process execution, automatically invoking the service task *Call Productname Integration*. Because this integration is encapsulated as an HTTP-based service explained in (cf. Section 5.4.1) and exposed through the gateway, the execution is routed securely via Kong and delivered to the Nuclio function in a private network. This ensures that the update—once validated by both an administrator and an end user—is consistently propagated across the Finance and Stock services, with authorization guaranteed.

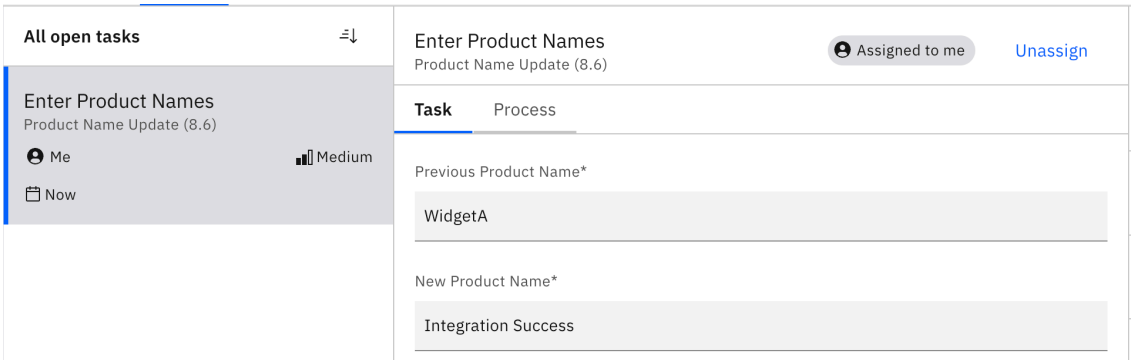


Figure 78 Tasklist Interface – Enter Product Names

The alternation between human and service tasks observed in this execution reflects the dual semantics extensively analyzed in Chapter 5, (cf. Section 5.6.2). Human tasks embody suspension, state persistence, and explicit actor intervention, while service tasks represent autonomous system-level operations. Although platforms such as Camunda and Airflow differ in their technical implementations—REST-based Tasklist operations in the former, operator scheduling in the latter—the semantics remain invariant. The practical scenario thus validates empirically what was argued conceptually in the prototype: workflow engines govern hybrid sequences of human and automated tasks in a manner that is both consistent across technologies and adaptable to organizational requirements.

Equally central to the scenario is the framework's uniform treatment of authorization and identity propagation. When the *productname_integration* is invoked, the assurance that only authenticated and authorized actors can reach the service derives from the governance perimeter established at the gateway. As examined in Chapter 5, (cf. Section 5.2.3), three integration contexts can be encountered in practice: systems with external identity providers (Option A), systems with in-code authorization logic (Option B), and systems with no native authorization (Option C). The framework abstracts over these differences by centralizing validation at the gateway, so that all requests passing to the integration layer are already secured before being delivered to the integration logic.

The execution of the *praticalscenarioworkflow* process thus demonstrates the full range of the framework's capabilities. By incorporating an administrative approval step, it captures organizational oversight; by requiring user input, it illustrates human–system collaboration; and by invoking a service task, it validates the seamless integration of automated orchestration. Together, these dimensions consolidate the framework's two core commitments. First, they affirm the universality of workflow semantics across engines, demonstrating that governance and execution are consistent irrespective of technological implementation. Second, they reaffirm the principle of centralized authorization, confirming that integrations remain focused exclusively on business logic while all security concerns are managed uniformly at the gateway. In sum, this practical scenario consolidated that the framework is not merely a technical device for system interoperability, but a comprehensive architecture capable of embedding

organizational governance, human participation, and secure automation within a single coherent process.

7 Proposal Evaluation

This chapter outlines the evaluation of the proposed framework through the Goal/Question/Metric (GQM) methodology [90]. The purpose is not only to validate whether the framework meets its intended objectives but also to demonstrate, in measurable terms, how its architecture fosters interoperability, security, and maintainability across heterogeneous systems. The GQM approach offers a structured way to link abstract research goals with concrete evaluation metrics, ensuring that every measurement is meaningful and contributes to the validation of the system.

7.1 Methodology

The GQM method, originally introduced at NASA's Goddard Space Flight Center, provides a structured framework for establishing goal-oriented measurements. Its foundation lies in the principle that meaningful evaluation begins with clearly defined goals. These goals are progressively refined into a series of guiding questions, which in turn determine the concrete metrics required to generate the necessary evidence. The framework thus enforces a top-down definition process—starting from goals and working toward measurable indicators—while ensuring that the interpretation of data follows a bottom-up logic, where raw measurements are aggregated and analyzed to answer the questions and ultimately validate or refute the goals [80]. Figure 79 illustrates this layered structure, emphasizing how GQM unites strategic intent with operational evidence through its bidirectional perspective: definition flows top-down, while interpretation flows bottom-up.

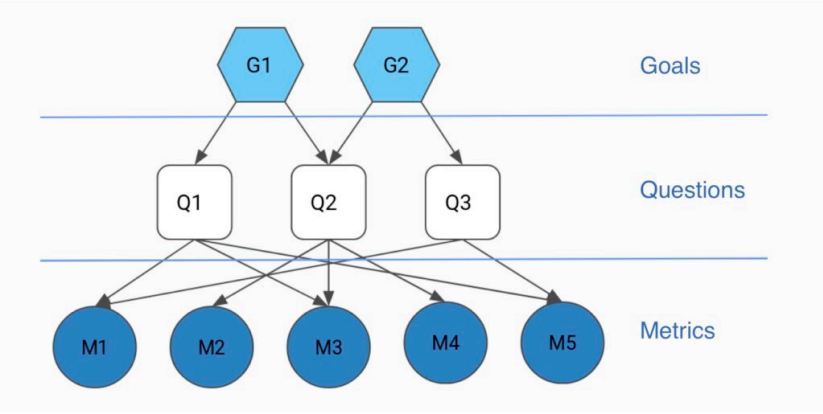


Figure 79 Goal Question Metric Methodology [91]

The GQM has, therefore, three levels. Each one of them is related to the three main components [81]:

- Conceptual Level (Goal) – At this level, the intention is defined in the form of a goal, shaped by the organizational context, quality models, and other environmental factors. The goal represents the high-level objective that the evaluation seeks to address.
- Operational Level (Question) – Goals are then translated into questions that guide the evaluation. These questions specify the quality aspects under scrutiny and frame the perspective from which the framework will be assessed.
- Interpretative Level (Metric) – In classical GQM, this level is associated with quantitative measures that provide objective evidence to answer the guiding questions. In the context of this dissertation, however, the metrics are predominantly qualitative in nature, expressed through architectural inspection, repository analysis, and scenario-based validation (Evidences). While not numerical, these metrics nonetheless offer systematic and reproducible evidence, ensuring that the evaluation remains rigorous and aligned with the GQM methodology.

In place of quantitative metrics, this dissertation adopts the dual notion of Evidences and Evidence Sources, recognising the qualitative orientation of the evaluation. Here, Evidences denote the concrete manifestations by which the achievement of goals can be verified, while Evidence Sources identify the underlying artifacts, repositories, or system components from which such evidences are drawn. This shift preserves methodological rigour by ensuring that the evaluation remains reproducible and systematically grounded, even when numerical measurement is not applicable.

7.2 Applying the GQM

The following section operationalises the GQM methodology within the context of this framework. Having outlined its three levels—goals, questions, and metrics—the discussion now turns to their concrete application across the evaluation dimensions of the dissertation. Each goal is first contextualised, then expanded into specific research questions, and finally supported by qualitative evidence and evidence sources, ensuring that the evaluation is both systematic and empirically grounded.

7.2.1 G1 — Framework Agnosticism

Intent

The first goal addresses the requirement that the framework remain agnostic to specific technological stacks across its orchestration, runtime, and integration layers. Independence is expected at the level of workflow engines, serverless platforms, and integration languages.

Identity management, however, follows a different paradigm: rather than being agnostic, it is centralised through Keycloak, which federates external providers into a unified model. This distinction highlights the distribution of agnosticism: workflow orchestration, runtime environments, and integration logic exhibit neutrality, while identity and authorisation are deliberately centralised. By adopting this hybrid approach, the framework mitigates vendor lock-in, ensures long-term adaptability, and positions itself as resilient to technological shifts.

Key Questions

Q1.1 — Can the framework dispatch BPMN actions across different workflow engines without requiring upstream changes?

This question probes orchestration neutrality. If the same process definition can be executed in Camunda Zeebe or Apache Airflow without changes in the upstream code, orchestration semantics can be considered engine-independent.

Q1.2 — Can the same integration logic be deployed seamlessly across distinct serverless runtimes such as Nuclio and Fission?

This question evaluates runtime portability. Demonstrating that an identical integration artifact executes correctly on heterogeneous serverless platforms validates runtime neutrality.

Q1.3 — Can integrations be developed in different languages (e.g., Ballerina, Java, Python) and still be deployed within the framework without requiring structural changes?

This question addresses the polyglot character of the Integration Layer. If services implemented in multiple languages can be encapsulated and exposed as HTTP endpoints, later deployable on serverless backends, the framework demonstrates agnosticism at the level of integration logic itself.

Q1.4 — Does the identity and authorisation layer remain functional when external IdPs are federated into Keycloak?

This question recognises that the IAM is not agnostic but centralised. The relevant test is whether tokens and roles from external IdPs are successfully federated into Keycloak and enforced consistently at Kong, thereby guaranteeing interoperability in security.

Q1.5 — Can new engines or service environments be integrated by implementing only adapters, without changes to the core framework?

This question evaluates extensibility. If adding new engines or runtimes requires only the creation of an adapter, the principle of decoupling is validated.

Evidence Sources

Validation of G1 requires evidence drawn from multiple architectural layers. At the orchestration level, E1 is provided by the Process Execution Manager, which offers execution logs and adapter mappings of the BpmnAction enumeration, as detailed in Chapter 5 (cf. Section 5.5), demonstrating whether orchestration verbs are consistently abstracted across engines. At the runtime level, E2 comes from the Serverless Services container, which provides build artifacts, deployment logs, and container images from Nuclio and Fission (cf. Section 5.4), showing whether integrations are portable across heterogeneous runtimes. At the integration layer, E3 is represented by repositories of Ballerina projects, as well as proof-of-concept integrations in Java or Python, function as sources: they demonstrate that different languages can be encapsulated and exposed as uniform HTTP services, later deployed seamlessly into serverless environments. For identity management, E4 corresponds to the Kong–Keycloak federation layer, where configurations, introspection traces, and mapping rules (cf. Section 5.1) illustrate how external IdPs are absorbed into Keycloak while preserving consistent enforcement at Kong. Finally, E5 is constituted by the adapter repositories and their interfaces, which provide the clearest source for evaluating the extensibility, confirming that the integration of new environments is achieved through modular adapters rather than changes to the core framework.Table 5 summarizes the mentioned questions and Evidence Sources.

Table 5 Adopted Questions and Evidence Sources for Goal 1

Goal (G1)	**Ensure framework agnosticism across orchestration, runtime, and integration layers, while centralising identity under IAM.**
Key Questions (Q)	**Evidence Sources (E)**
Q1.1 — Can the framework dispatch BPMN actions across different workflow engines without requiring upstream changes?	E1: Execution traces of BpmnAction verbs dispatched to Camunda Zeebe and Apache Airflow
Q1.2 — Can the same integration logic be deployed seamlessly across distinct serverless runtimes such as Nuclio and Fission?	E2: Deployment logs showing identical artifacts running in Nuclio and Fission
Q1.3 — Can integrations be developed in different languages (Ballerina, Java, Python) and still be deployed without structural changes?	E3: Integration repositories proving encapsulation of services in Ballerina, Java, Python.
Q1.4 — Does the identity and authorisation layer remain functional when external IdPs are federated into Keycloak?	E4: Kong–Keycloak federation traces and introspection logs
Q1.5 — Can new engines or service environments be integrated by implementing only adapters?	E5: Adapter repositories confirming extensibility without core changes.

7.2.2 G2 — Consistent Unified Authentication and Authorisation

Intent

The second goal evaluates the framework's ability to enforce authentication and authorisation in a centralised and consistent manner. Both identity validation (authentication) and access control (authorisation) must occur once, at the edge, and their results must be uniformly propagated across containers. By concentrating these responsibilities in the IAM and the gateway, the framework preserves clarity of responsibility, avoids policy drift, and reduces the likelihood of inconsistencies or duplicated enforcement in downstream code. A further dimension concerns integrations: they must remain strictly agnostic to security enforcement, relying exclusively on the validated identity context passed from the gateway. This unified approach reinforces trust in the security perimeter, ensures that governance is applied consistently, and guarantees that business logic remains free from embedded security checks. Table 5 summarizes the mentioned questions and Evidence Sources.

Key Questions

Q2.1 — Do all externally reachable requests traverse the Core Gateway?

This establishes whether the gateway serves as the exclusive controlled ingress, ensuring no bypass of the authentication/authorisation perimeter.

Q2.2 — Is authentication always performed against the central IAM before any request is admitted?

This probes whether all users and services are authenticated by Keycloak (via OIDC flows) before interacting with the framework, thereby guaranteeing a uniform source of identity.

Q2.3 — Are authorisation rules defined only in the IAM/Gateway system and not duplicated elsewhere?

This verifies whether access decisions are centralised, ensuring that no integration or BPMN logic embeds role checks.

Q2.4 — Can route-level policies be updated centrally without modifying integrations?

This assesses governance agility, confirming that changes in access control can be made at the IAM/gateway level without impacting deployed business logic.

Q2.5 — Are tokens validated consistently across all containers, with no redundant re-validation?

This confirms runtime consistency: tokens validated at the edge must be propagated downstream unchanged, avoiding unnecessary repetition.

Q2.6 — How is authorisation handled in integrations, and is it guaranteed that they do not implement their own access control logic?

This question clarifies the treatment of authorisation in the Integration Layer. It confirms whether integrations are strictly dedicated to business logic, with security enforcement being centralised upstream at the IAM and Gateway.

Evidence Sources

Validation of G2 requires evidence from multiple sources. E6 is provided by Kong access logs, which confirm whether all inbound traffic traverses the gateway , ensuring a single controlled ingress. E7 comes from authentication traces in Keycloak (login events, token issuance, and OIDC flows) demonstrating that all requests are first authenticated centrally. E8 is drawn from role mappings and policy bindings in Keycloak, together with Kong route definitions, serve as sources for verifying centralised authorisation. To address governance agility, E9 is offered by the Konga configurations, showing that route-level policies can be updated centrally without requiring modifications to integrations. Runtime consistency is evidenced by E10, where traces demonstrate that tokens validated at the edge are propagated downstream without redundant re-validation. Finally, E11 is derived from inspections of Nuclio and Ballerina integration repositories, which confirm the absence of embedded authentication or role-check logic, showing that integrations operate purely as business services dependent entirely on upstream authorisation. Table 6 summarizes the mentioned questions and Evidence Sources.

Table 6 Adopted Questions and Evidence Sources for Goal 2

Goal (G2)	**Guarantee centralised and consistent enforcement of identity and access control.**
Key Questions (Q)	**Evidence Sources (E)**
Q2.1 — Do all externally reachable requests traverse the Core Gateway?	E6: Kong access logs and network policies proving single ingress.
Q2.2 — Is authentication always performed against the central IAM before any request is admitted?	E7: Keycloak OIDC traces confirming uniform authentication.
Q2.3 — Are authorisation rules defined only in the IAM/Gateway and not duplicated elsewhere?	E8: Role mappings and Kong route definitions; repository inspection of integrations showing no role checks.
Q2.4 — Can route-level policies be updated centrally without modifying integrations?	E9: Konga configs proving governance agility.
Q2.5 — Are tokens validated consistently across all containers, without redundant re-validation?	E10: Runtime traces showing downstream token propagation without re-validation.
Q2.6 — How is authorisation handled in integrations, and do they avoid implementing their own access control logic?	E11: Nuclio/Ballerina repositories confirming absence of embedded access control.

7.2.3 G3 — Flexible, Extensible and Reliable Integration Layer

Intent

The third goal concerns the Integration Layer, focusing on its flexibility, extensibility, and reliability. The framework must demonstrate that developers can rapidly scaffold integrations, generate client modules automatically from service contracts, and deploy these integrations across heterogeneous runtimes without excessive effort. At the same time, the Integration Layer must prove its resilience by ensuring consistent execution across environments and by fostering modularity, so that integrations remain independent of specific workflows or services and can be reused in multiple scenarios. This combination of automation, portability, and reusability is central to confirming that the Integration Layer provides not only technical convenience but also long-term maintainability within enterprise contexts.

Key Questions

Q3.1 — Can a developer scaffold a new integration project with minimal effort?

This question verifies whether the developer experience is optimised, reducing time-to-delivery through automation. If scaffolding commands can produce a project skeleton in seconds, the framework proves its capacity to accelerate the integration lifecycle.

Q3.2 — Can client modules be generated automatically from OpenAPI specifications for subsequent injection into integrations?

This question addresses the contract-first principle. By generating reusable clients directly from service contracts, the framework ensures consistency with upstream APIs while reducing the likelihood of manual errors. The automation of client generation is crucial, as it enables integrations to consume heterogeneous systems in a structured and maintainable way.

Q3.3 — Is polyglot support viable, with Ballerina as default but not exclusive?

This question evaluates technological flexibility. The framework should not impose a single language; although Ballerina is the default, developers must be free to employ Java, Python, or other integration-oriented languages, which can then be encapsulated within the same deployment pipeline.

Q3.4 — Can integrations be deployed across different serverless platforms (Nuclio, Fission) without modification?

This question tests runtime portability. If the same integration package executes seamlessly across multiple function-as-a-service environments, it confirms that the framework avoids runtime lock-in.

Q3.5 — Are integrations designed to remain modular, avoiding coupling with specific workflows or services, so that they can be reused across scenarios?

This question probes the framework's capacity to preserve modularity and reusability. By ensuring that integrations are not hardwired to a single BPMN process or business service, the Integration Layer validates its role as a general-purpose abstraction rather than a scenario-specific implementation.

Evidence Sources

The validation of G3 relies on a diverse set of architectural components and operational artifacts that serve as sources for inspection. To confirm scaffolding efficiency , the primary sources are the Integration Manager logs and CLI outputs associated with project generation, where evidence E12 demonstrates that project skeletons can be created with minimal effort as detailed in Chapter 5. For client module generation , the relevant sources are the DevPortal interactions, where OpenAPI specifications are uploaded and translated into reusable client modules, together with the repositories where these generated modules are stored; evidence E13 confirms that this process adheres to the contract-first approach and produces consistent reusable artifacts. Polyglot support is assessed by inspecting integration repositories that include examples of Ballerina projects alongside alternative language implementations (e.g., Java or Python), all encapsulated within the same containerisation workflow; evidence E14 ilustrates that the framework accommodates this diversity without compromising the uniform deployment pipeline. Runtime portability is evaluated through build and deployment logs targeting heterogeneous serverless platforms such as Nuclio and Fission, with evidence E15 confirming that identical integration artifacts execute seamlessly across both environments. Finally, modularity and reusability are validated by analysing repository structures and BPMN process associations, where evidence E16 substantiates that a single integration can be bound to multiple workflows without alteration, reinforcing the claim that the Integration Layer supports long-term maintainability and reuse. Table 7 summarizes the mentioned questions and Evidence Sources.

Table 7 Adopted Questions and Evidence Sources for Goal 3

Goal (G3)	**Ensure the Integration Layer supports rapid scaffolding, contract-driven generation, polyglot support, runtime portability, and reusability.**
Key Questions (Q)	**Evidence Sources (E)**
Q3.1 — Can a developer scaffold a new integration project with minimal effort?	E12: CLI logs and Integration Manager outputs showing scaffolding efficiency.
Q3.2 — Can client modules be generated automatically from OpenAPI specifications?	E13: DevPortal interactions and repositories containing generated OpenAPI client modules.
Q3.3 — Is polyglot support viable, with Ballerina as default but not exclusive?	E14: Repositories with Ballerina, Java, Python modules encapsulated under the same pipeline.
Q3.4 — Can integrations be deployed across different serverless platforms (Nuclio, Fission) without modification?	E15: Deployment logs confirming execution on Nuclio and Fission.
Q3.5 — Are integrations modular and reusable across workflows?	E16: Repository structures and BPMN associations confirming reusability.

7.3 Results

The purpose of this section is to present the outcomes of the evaluation conducted through the Goal–Question–Metric methodology, applied to the proposed integration framework. By aligning the predefined goals with the corresponding evidence obtained from architectural inspection and the practical scenario, the analysis seeks to verify the extent to which the framework fulfils its intended objectives. In doing so, this section provides a critical account of the framework's performance, offering empirical grounding for claims of interoperability, security, and flexibility advanced in earlier chapters.

7.3.1 G1 — Framework Agnosticism

The first goal assessed the extent to which the framework preserved technological agnosticism across its principal layers, namely workflow orchestration, serverless execution, and integration development. As established in Chapter 5, the architecture was deliberately designed to avoid vendor lock-in by introducing abstraction points: the Process Execution Manager was conceived as an engine-independent orchestration layer, the Integration Layer as polyglot and service-oriented, and the Serverless Services container as runtime-neutral. Identity management, however, was intentionally centralised under Keycloak, reflecting a strategic choice of consistency over neutrality. The evaluation, therefore, set out to determine whether these design assumptions could be corroborated through the practical scenario.

Evidence collected confirmed that the framework achieved engine neutrality. As raised in Q1.1, execution traces showed that abstract verbs defined in the `BpmnAction` enumeration were correctly mapped to both Camunda Zeebe and Apache Airflow adapters, enabling processes to be executed across heterogeneous engines without any upstream modifications. This result validates the orchestration independence discussed in the previous chapters. Similarly, addressing Q1.2, container images of the productname integration, from the pratical scenario, were executed in both Nuclio and Fission without changes to the codebase, thereby corroborating runtime portability. These two findings jointly demonstrate that the framework effectively mitigates lock-in at both orchestration and execution levels.

The evaluation also confirmed polyglot support in the Integration Layer, as questioned in Q1.3. Although Ballerina was employed as the principal implementation language, the architectural design ensured that equivalent services in Java or Python could be encapsulated as HTTP endpoints and deployed transparently to the serverless environment. This flexibility validates the claim that integration logic is not bound to a single technological stack. By contrast, identity management could not be deemed agnostic. With respect to Q1.4, the evidence showed that all external IdPs must be federated into Keycloak before being enforced at Kong. While this does not establish neutrality, it guarantees consistent and centralised enforcement, ensuring interoperability in security without duplicating policies downstream.

Finally, the extensibility of the framework, posed in Q1.5, was confirmed by the adapter design. Repository analysis showed that supporting new execution environments required only the implementation of additional adapters, leaving the Process Execution Manager and gateway unmodified. This validates the decoupling principle emphasised in Chapter 5. Taken together, these observations demonstrate that the framework is engine-neutral, runtime-portable, and polyglot at the integration layer. The evidence therefore substantiates that Goal 1 was successfully achieved, with agnosticism realised wherever feasible and centralisation applied where strategically necessary.

7.3.2 G2 — Consistent Unified Authentication and Authorisation

The second goal investigated the framework's ability to unify authentication and authorisation under a single, consistent perimeter. As argued in Chapter 5 (cf Section 5.1), the architectural decision to concentrate both identity validation and access control at the edge sought to resolve two recurrent problems in distributed architectures: the risk of fragmented authentication paths and the duplication of authorisation policies across heterogeneous services. The evaluation, therefore, aimed to determine whether the conceptual commitment to centralisation was realised in practice, and whether this unification genuinely simplified governance while safeguarding downstream services from security concerns.

The first aspect, raised in Q2.1, concerned ingress centralisation. Evidence drawn from Kong access logs and corroborated by network policy configurations confirmed that every inbound request was received exclusively at the Core Gateway, with no alternative routes available. This finding validates the principle that the gateway constitutes the unique point of entry into the

framework, ensuring that all requests are subject to consistent security enforcement from the outset. In turn, Q2.2 focused on authentication, and here the results demonstrated a rigorous reliance on the central IAM: every user and service authenticated solely against Keycloak through OIDC flows before being issued a token. No other container generated or validated identities independently, thereby guaranteeing that identity management stemmed from a single authoritative source.

Addressing Q2.3, repository inspection of Nuclio and Ballerina functions revealed the absence of role-checking logic, annotations, or middleware that might otherwise replicate authorisation concerns. Instead, enforcement was declaratively bound in Keycloak role mappings and Kong route definitions, as confirmed by the policy configurations visible in Konga. This substantiates the claim that access control was centralised and free from duplication. Furthermore, as anticipated in Q2.4, these declarative bindings ensured that modifying a policy required only changes in the IAM or gateway configuration: the integrations themselves remained untouched, thereby confirming that governance agility was preserved without impacting business logic.

The runtime behaviour addressed in Q2.5 further reinforces this picture of consistency. Execution traces across Kong, PEM, and Nuclio showed that once validated at the edge, tokens were propagated downstream intact. No container revalidated the token, nor attempted to override its claims; the validated identity was simply passed along, allowing subsequent services to trust the original enforcement without incurring redundant checks. This consistency eliminates the risk of divergence between services and streamlines runtime execution. Lastly, Q2.6 examined the function of integrations in general. These modules only ran using the identity context that was passed down from the gateway, according to repository analysis and execution logs, which showed that there was no embedded security code. While trust and authorization were fully delegated upstream, their responsibility was still limited to business logic. By keeping security consistent, this division of responsibilities not only makes the integration layer easier to use but also fortifies the governance perimeter.

Taken together, these findings provide converging confirmation that the framework delivers a consistent and centralised enforcement of both authentication and authorisation. Each of the six questions finds an affirmative response in the evidence, with logs, repository analyses, and runtime trace. The framework thus establishes a clear division of responsibilities: security decisions are made once at the edge, while integrations and processes focus exclusively on business logic. In doing so, it not only prevents the fragmentation and drift of policies but also consolidates governance into a single, reliable perimeter. The evidence therefore substantiates that Goal 2 was fully achieved, providing a coherent and operational realisation of consistent and unified authentication and authorisation.

7.3.3 G3 — Flexible and Reliable Integration Layer

The third goal evaluated the flexibility and reliability of the Integration Layer, conceived in Chapter 5 (cf. Section 5.6) as the architectural locus where integrations should be created rapidly, reused consistently, and executed independently of specific technologies. While Ballerina was selected as the default implementation language, the design was explicitly polyglot, allowing equivalent logic to be expressed in other languages and encapsulated under the same deployment model. The assessment therefore sought to determine whether this theoretical openness was matched in practice, particularly with regard to project scaffolding, client generation from OpenAPI contracts, runtime neutrality across serverless platforms, and the consistent inheritance of identity and authorisation context from the security perimeter.

Evidence from the scenario confirmed that development workflows could be initiated with minimal effort, thereby addressing Q3.1. CLI logs showed that the bal new command produced fully structured project skeletons in seconds, while DevPortal interactions (cf. Chapter 6) demonstrated that scaffolding scripts were reproducible and conformed to framework conventions. This ensured that developers were not burdened with boilerplate setup and could focus directly on integration logic. In response to Q3.2, outputs from bal openapi confirmed the automatic generation of typed client modules for both the Finance and Stock services, which were then injected directly into the integration logic. This contract-first approach not only guaranteed alignment with upstream APIs but also significantly reduced the likelihood of errors stemming from manual request handling.

The polyglot capacity of the Integration Layer, highlighted in Q3.3, was also validated. Although Ballerina was used in the prototype, the architecture demonstrated that equivalent modules could be implemented in Java or Python and deployed transparently through containerisation. HTTP-based encapsulation ensured that such modules behaved as first-class citizens within the framework, showing that Ballerina was a pragmatic default rather than a rigid imposition. Runtime neutrality, as raised in Q3.4, was confirmed by deployment artifacts: the same integration container executed without modification in both Nuclio and Fission, thereby corroborating the runtime portability anticipated in Chapter 4. This evidence establishes that integrations are not tied to a single serverless platform but can be redeployed across environments without code changes.

Finally, Q3.5 concerned the modularity and reusability of integrations, asking whether they could remain independent of specific workflows or business services and thus be redeployed across scenarios. Evidence from repository analysis and BPMN associations confirmed that a single integration was indeed capable of serving multiple processes without modification. In practice, this meant that the same productname integration could be invoked in distinct workflows, thereby validating the abstraction principle. This confirmed that integrations were not hardwired into a single context but were instead designed as general-purpose, reusable artifacts.

Taken together, these findings show that the Integration Layer delivered precisely on its design objectives. Scaffolding and client generation streamlined the development workflow, polyglot support and runtime neutrality safeguarded long-term adaptability, and modularity ensured that integrations could be reused across scenarios without duplication. The evidence therefore confirms that Goal 3 was successfully achieved: the Integration Layer emerged as a flexible, reliable, and strategically robust element of the framework, enabling sustainable integrations across heterogeneous enterprise environments.

7.3.4 Results Evaluation Synthesis

The evaluation of the framework through the three defined goals has demonstrated that its architectural principles hold consistently when applied in practice. With orchestration, serverless execution, and integration logic functioning independently of particular technologies, framework agnosticism was not just a theoretical assertion but a verified characteristic. A single, uniform perimeter was created by the unified handling of authorization and authentication, which also ensured governance clarity and stopped security policy drift. By facilitating quick scaffolding, contract-driven client creation, and deployment across various runtimes, the Integration Layer also demonstrated its adaptability and dependability. When combined, these results offer strong evidence that the framework is both theoretically solid and workable.

However, the outcomes demonstrate that intentional architectural discipline is necessary for such accomplishments. Only by enforcing stringent boundaries between concerns—identity was validated only at the edge, orchestration remained abstract, and integrations were kept lightweight and reusable—was it possible for agnosticism, security unification, and integration reliability to cohere. This methodical division guarantees that the framework not only meets its short-term objectives but also establishes a strong basis for long-term development, confirming that the framework is successful in converting design concepts into real-world outcomes and solidifying its position as an organized and reliable method of enterprise system integration.

8 Conclusion

This final chapter brings the dissertation to its close by reflecting on the overall outcomes of the research. Its purpose is to synthesise the work undertaken, showing how the initial objectives and research questions have been addressed and what contributions emerge from their resolution. In doing so, the chapter not only highlights the achievements and practical relevance of the proposed integration framework but also acknowledges its limitations, outlines directions for future development, and formulates the final considerations that situate this work within the broader landscape of enterprise system integration.

8.1 Achievements

At the outset of this dissertation, five objectives were articulated in cf. Section 1.2 as guiding commitments for the design and validation of a process-centric integration framework. These objectives were conceived not as isolated milestones but as interdependent dimensions of a coherent vision. The first of these commitments (O1) was the creation of a modular and process-centric architecture that could mitigate the fragmentation of enterprise information systems. By layering the framework into the Core Gateway, Process Execution Manager, Integration Layer, Serverless Services, and Monitoring, that objective was accomplished. In order to counteract the systemic incoherence that frequently characterizes fragmented landscapes, each container was designed with distinct boundaries of responsibility, guaranteeing separation of concerns while maintaining alignment between orchestration, integration, and governance.

Equally important was the concern with transparency and formalisation in process orchestration (O2). To address this objective, the framework adopted BPMN in strict adherence to the ISO/IEC standard, ensuring that process models were not confined to conceptual diagrams but became executable artifacts deployable across heterogeneous engines. In this way, the modelling language acted as a bridge between organizational and technical domains, reducing ambiguity, enhancing traceability, and providing evidence that orchestration could remain both transparent and standardised across contexts.

The objective of guaranteeing reliable and semantically consistent data exchange (O3) was pursued through a contract-first integration strategy. By generating typed client modules automatically from OpenAPI specifications and injecting them directly into integration logic, the framework ensured that all communication adhered to published service contracts. Repository inspections and runtime traces demonstrated that these modules were reused across different workflows, preventing semantic drift and reducing redundancy, thereby confirming the framework's ability to sustain accuracy and integrity in cross-system communication.

Promoting technological independence and sustainability was another commitment (O4). This was accomplished by intentionally depending on open and configurable technologies: Nuclio provided elastic execution, Ballerina enabled integration design, Camunda provided process abstraction, Kong and Keycloak secured identity and ingress. This dedication was strengthened by the Process Execution Manager's adapter-based architecture, which supports the addition of new engines or runtimes without changing the core. When combined, these design decisions confirmed the framework's long-term flexibility and protected it from vendor lock-in.

The definition of a comprehensive methodology that bridges enterprise architecture principles with integration practices (O5) was demonstrated by showing how the framework aligns technical interoperability with organizational goals. The architecture was created to embed orchestration and governance within a methodological framework influenced by enterprise architecture standards, rather than viewing integration as a purely operational issue. In order to demonstrate how organizational responsibilities and technical automations can be balanced in a single architecture, this orientation was made tangible in the previous pratical scenario where automated services and human tasks coexisted within the same process model. The framework thus confirmed that integration is both technically sound and organizationally significant, advancing the dual imperative of interoperability and strategic alignment.

From these objectives emerged the two central research questions that structured the dissertation. The first, *"What key characteristics and components should a business process-centric framework have to be usable in the wide range of possible information systems architectures?"*, was answered through both architectural reasoning and empirical validation. The framework proved to be modular in its organization, abstract in the separation of process logic from execution details, interoperable across engines and external systems, and scalable through elastic deployment models. These qualities were consistently evidenced by design choices described in Chapter 4, prototyped in Chapter 5, demonstrated in Chapter 6 and by the evaluation results in Chapter 7, showing that the intended properties were achieved not only conceptually but operationally.

The second research question, *"How can enterprise integration patterns and open-source technologies contribute to the conception and construction of a business process-centric framework that promotes seamless communication and adaptability in complex system environments?"*, was addressed through the integration of BPMN, Kong, Keycloak, Ballerina, and Nuclio. Each technology assumed a specific role, such as process abstraction, API and identity governance, integration development, and elastic execution, while preserving smooth interoperability within the architecture. In addition to satisfying the theoretical needs of modularity, interoperability, and scalability, the evaluation demonstrated that open-source ecosystems can support enterprise-grade integration without relying on proprietary solutions.

The framework's practical implementation, which acted as a proof-of-concept for its design principles, was a significant accomplishment of this dissertation in addition to its architectural and technological contributions. The scenario mentioned, which focused on the synchronization of product information between stock and finance services, allowed the

framework to be used throughout its whole stack, demonstrating the real-practical interactions between security, integration, and orchestration mechanisms. This experiment provided the empirical ground on which the subsequent evaluation could be carried out. Building on this, Chapter 7 extended the analysis by applying the Goal–Question–Metric methodology, which systematically connected the framework's architectural intentions with observable evidence. In this way, the combination of practical demonstration and structured evaluation ensured that the framework was not only conceptually coherent but also operationally validated.

8.2 Limitations

The findings presented in this dissertation must be interpreted with awareness of the inherent limitations that constrain their generalisability. The assessment was carried out in a proof-of-concept setting, deliberately limited to a limited number of services—most notably the stock and finance modules—and carried out under carefully monitored deployment conditions. Although this environment was adequate to show correctness, modularity, and architectural soundness, it is unavoidably unable to replicate the complexity of production landscapes, which are characterized by the coexistence of numerous heterogeneous systems, extensive processes, and several concurrent requests. Furthermore, responsiveness and elasticity were found to remain under interactive thresholds, but no systematic benchmarking or stress-testing was conducted. Performance and scalability were evaluated solely qualitatively. Consequently, throughput, latency, and fault tolerance under stress are still open questions, and quantitative validation will be required to support the qualitative evidence that has been provided thus far.

Another potential threat relates to the specific technological stack employed in the implementation. Although the framework was designed to be runtime-neutral, engine-agnostic, and polyglot, the empirical validation was solely dependent on a specific combination: Nuclio for serverless deployment, Kong and Keycloak for identity and API management, Ballerina for integration development, and Camunda for process orchestration. The empirical scope is inevitably limited, even though this set of technologies was chosen for its maturity, open-source accessibility, and experimentation suitability. The claims to universality therefore rest on the abstraction mechanisms embedded in the architecture rather than on direct experimentation with alternative technologies. Further validation would require expanding the assessment to include serverless backends (like Fission, OpenFaaS), multiple workflow engines (like Flowable, jBPM), and polyglot modules written in languages other than Ballerina (like Java, Python). Such proofs would support the framework's claim to long-term adaptability in diverse enterprise contexts by bolstering the empirical foundation of its neutrality and offering more compelling proof of its resistance to technological substitution.

A further limitation concerns operational governance. Features like auditing, fine-grained policy evolution, and compliance integration were only partially addressed by the framework, despite the fact that it introduced mechanisms for centralised identity and policy enforcement. These governance aspects are frequently just as crucial in actual enterprise deployments as orchestration or integration, and their incomplete handling here raises the possibility that

crucial organizational needs, like policy lifecycle management or regulatory compliance, may not yet be fully met. Adoption in settings with more stringent regulatory frameworks may be hampered by the lack of integrated tools for formal compliance checking and ongoing monitoring, which suggests that governance is still dependent on manual oversight. Addressing these aspects in future iterations will be essential to ensure that the framework can operate not only as a technically coherent system but also as one that fully aligns with organizational governance imperatives.

8.3 Future Work

Building on the limitations identified, several trajectories for future work naturally emerge. From a technical standpoint, there is a need to extend the orchestration model towards forms of coordination that go beyond the strictly process-driven approach demonstrated here. Future iterations could explore mechanisms that integrate synchronous and asynchronous modes of execution, thereby strengthening the framework's ability to address high-throughput workloads and long-running processes. Equally important is the maturation of the observability layer: while basic monitoring was achieved, enterprise-grade deployments demand richer instrumentation, including mechanisms for end-to-end tracing, performance benchmarking, and advanced fault diagnosis, which would allow administrators to evaluate the behaviour of the framework under more demanding operational conditions.

At the architectural level, further empirical assessment is required to reinforce the framework's claims of technological agnosticism. The current evaluation confirmed its capacity to accommodate heterogeneous engines and runtimes in principle, but broader comparative studies would provide more robust evidence of its portability and resilience. Extending the Integration Layer to encompass more diverse programming paradigms and development practices would likewise demonstrate that the promise of polyglot integration is not merely conceptual but applicable across varied developer communities. Such evolution would not only broaden the framework's inclusivity but also strengthen its sustainability by fostering adoption in different organizational contexts.

Finally, the governance dimension offers fertile ground for future refinement. Although centralised role-based enforcement was effectively demonstrated, more expressive access-control models and their incorporation into compliance-driven environments could be the subject of future research. Beyond these pressing issues, the framework can be applied to new fields where the conflicts between autonomy, interoperability, and resilience are even more noticeable, like supply chain management, the Internet of Things, or cross-organizational workflows. Such extensions would provide rigorous tests of adaptability, while also demonstrating the framework's potential as a strategic instrument for addressing the systemic challenges of digital ecosystems across industries.

8.4 Final Considerations

This dissertation has argued that process-centric integration offers a viable path to overcome the fragmentation that continues to characterise enterprise information systems. By combining modular architectural design, adherence to open standards, and the disciplined use of open-source technologies, it has been shown that heterogeneous systems can be orchestrated into a coherent, secure, and sustainable whole. The framework proposed here embodies the qualities identified as essential—modularity, abstraction, interoperability, and scalability—and has demonstrated through both architectural reasoning and empirical validation that these qualities can be realised in practice.

The contributions of the work are therefore twofold. On the practical side, it delivers a functioning artifact: an integration framework that was designed, implemented, and exercised in a real scenario of synchronisation between finance and stock services. This artifact demonstrates how BPMN-modeled business processes can be implemented as executable workflows that manage diverse systems via serverless deployments while maintaining edge security and governance. On the academic side, it illustrates how enterprise architecture and integration theory concepts can be operationalized in modern software ecosystems. Thus, the dissertation connects theory and practice, providing a tangible tool that practitioners can use as well as a methodological contribution to academic discussion.

Despite the limitations discussed, the conclusions remain convincing. The evaluation demonstrated that excessive complexity or reliance on monolithic platforms are not necessary for integration. With careful abstraction, separation of concerns, and reliance on open technologies, it is possible to design integration frameworks that are simultaneously technically robust, strategically adaptable, and sustainable over time. The framework presented here stands not as an endpoint but as a foundation: it establishes a clear direction for how organizations can reconcile autonomy with interoperability, and how future research can expand these insights into new domains and at greater scale. Ultimately, this dissertation demonstrates that integration can evolve from being a persistent challenge to becoming a strategic capability, enabling organizations to innovate, adapt, and thrive in increasingly complex digital ecosystems.

Bibliographic References

[1] A. Schmitz and M. A. Wimmer, “Framework for Interoperable Service Architecture Development,” *Gov Inf Q*, 2023, doi: 10.1016/j.giq.2023.101833.

[2] T. Nuutinen, “Introduction to Enterprise Architecture,” *Theseus.fi*, 2024, [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/867388/Nuutinen_Tiia.pdf?sequence=2

[3] L. Cheong and V. Chang, “The need for data governance: A case study,” in *ACIS 2007 Proceedings - 18th Australasian Conference on Information Systems*, 2007.

[4] O. Aziz, M. S. Farooq, A. Abid, R. Saher, and N. Aslam, “Research Trends in Enterprise Service Bus (ESB) Applications: A Systematic Mapping Study,” *IEEE Access*, vol. 8, pp. 31180–31197, 2020, doi: 10.1109/ACCESS.2020.2972195.

[5] G. Liu, B. Huang, Z. Liang, M. Qin, H. Zhou, and Z. Li, “Microservices: architecture, container, and challenges,” in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, IEEE, Dec. 2020, pp. 629–635. doi: 10.1109/QRS-C51114.2020.00107.

[6] K. Sookoo, J. P. van Belle, and L. Seymour, “Implementing an event-driven enterprise information systems architecture: Adoption factors in the example of a micro lending case study,” *Lecture Notes in Business Information Processing*, vol. 268, pp. 293–308, 2016, doi: 10.1007/978-3-319-49944-4_22/TABLES/2.

[7] N. B. Reinprecht, G. White, and M. Peters, “Enabling European electrical transmission and distribution smart grids by standards,” *IBM J. Res. Dev.*, vol. 60, no. 1, pp. 3:1–3:11, Jan. 2016, doi: 10.1147/JRD.2015.2482878.

[8] K. Hinkelmann, A. Gerber, and D. Karagiannis, “A New Paradigm for the Continuous Alignment of Business and IT,” *Comput Ind*, 2016, doi: 10.1016/j.compind.2015.12.002.

[9] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2012.

[10] A. Berti, W. M. P. van der Aalst, D. Zang, and M. A. K. Lang, “An Open-Source Integration of Process Mining Features Into the Camunda Workflow Engine: Data Extraction and Challenges,” *ArXiv*, 2020, doi: 10.18154/RWTH-2020-11538.

[11] M. A. Kohansal, “Navigating Enterprise Architecture (EA) Institutionalization,” *NTNUOpen*, 2024, [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/3125001/Mohammad%20Ali%20Kohansal.pdf?sequence=1

[12] D. N. Chorafas, “Enterprise architecture and new generation information systems,” *Enterprise Architecture and New Generation Information Systems*, pp. 1–384, Apr. 2016, doi: 10.1201/9781420000313.

[13] “ISO/IEC 19510:2013 - Information technology — Object Management Group Business Process Model and Notation.” Accessed: Dec. 16, 2024. [Online]. Available: https://www.iso.org/standard/62652.html

[14] D. N. Chorafas, “Enterprise architecture and new generation information systems,” *Enterprise Architecture and New Generation Information Systems*, pp. 1–384, Apr. 2016, doi: 10.1201/9781420000313/ENTERPRISE-ARCHITECTURE-NEW-GENERATION-INFORMATION-SYSTEMS-DIMITRIS-CHORAFAS/ACCESSIBILITY-INFORMATION.

[15] R. Z. Frantz, R. Corchuelo, and F. Roos-Frantz, “On the design of a maintainable software development kit to implement integration solutions,” *J. Syst. Softw.*, vol. 111, no. C, pp. 89–104, Jan. 2016, doi: 10.1016/j.jss.2015.08.044.

[16] B. Bellman and K. Griesi, “Enterprise architecture advances in technical communication,” in *2015 IEEE International Professional Communication Conference (IPCC)*, Jul. 2015, pp. 1–5. doi: 10.1109/IPCC.2015.7235834.

[17] Instituto Politécnico do Porto, “Regulamento do Código de Boas Práticas e de Conduta do Instituto Politécnico do Porto,” Oct. 2020.

[18] Ordem dos Engenheiros, “Código de Ética e Deontologia,” Nov. 2016.

[19] “IEEE - IEEE Code of Ethics.” Accessed: Dec. 14, 2024. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html

[20] K. Laudon and J. P. Laudon, “Management Information Systems: Organization and Technology in the Networked Enterprise,” 1999.

[21] A. Widianto and A. P. Subriadi, “Understanding ‘IS Effectiveness and Efficiency’: Based on Management Levels in the Organization,” in *Proceedings of the International Conference on Industrial Engineering and Operations Management*, Michigan, USA: IEOM Society International, Apr. 2021, pp. 2146–2151. doi: 10.46254/SA02.20210690.

[22] S. Liu and P. Chen, “Developing Java EE Applications Based on Utilizing Design Patterns,” in *2009 WASE International Conference on Information Engineering*, Jul. 2009, pp. 398–401. doi: 10.1109/ICIE.2009.151.

[23] S. Liu, A. H. B. Duffy, R. I. Whitfield, I. M. Boyle, and I. McKenna, “Towards the Realization of an Integrated Decision Support Environment for Organizational Decision Making,” *International Journal of Decision Support System Technology*, vol. 1, no. 4, pp. 38–58, Oct. 2009, doi: 10.4018/jdsst.2009062603.

[24] Y. Alduraywish, Y. Xu, and K. Salonitis, “State of the art of information systems failure managements,” *Advances in Transdisciplinary Engineering*, vol. 6, pp. 509–514, 2017, doi: 10.3233/978-1-61499-792-4-509.

[25] J. Valacich and C. Schneider, “Information Systems Today: Managing the Digital World,” 2009.

[26] G. Geddes, “Information Systems: An Introduction to Informatics in Organisations,” *Library Review*, vol. 52, no. 6, pp. 278–278, Aug. 2003, doi: 10.1108/00242530310482051.

[27] D. L. Goodhue and R. L. Thompson, “Task-Technology Fit and Individual Performance,” *MIS Q.*, vol. 19, no. 2, pp. 213–233, 1995, doi: 10.2307/249689.

[28] R. Stair and G. Reynolds, “Fundamentals of Information Systems,” 2001.

[29] H. Bidgoli, “Encyclopedia of information systems,” 2003.

[30] W. H. DeLone and E. R. McLean, “Information Systems Success: The Quest for the Dependent Variable,” *Information systems research*, vol. 3, no. 1, pp. 60–95, 1992, doi: 10.1287/ISRE.3.1.60.

[31] W. H. DeLone and E. R. McLean, “The DeLone and McLean Model of Information Systems Success: A Ten-Year Update,” *Journal of Management Information Systems*, vol. 19, no. 4, pp. 9–30, 2003, doi: 10.1080/07421222.2003.11045748.

[32] D. Avison and G. Fitzgerald, “Information Systems Development: Methodologies, Techniques and Tools,” 1988.

[33] Z. Pawlak, “Information systems theoretical foundations,” *Inf Syst*, vol. 6, no. 3, pp. 205–218, Jan. 1981, doi: 10.1016/0306-4379(81)90023-5.

[34] “The OMG® Specifications Catalog.” Accessed: Dec. 18, 2024. [Online]. Available: https://www.omg.org/spec/

[35] S. A. White and D. Miers, *BPMN Modeling and Reference Guide: Understanding and Using BPMN*. Future Strategies Inc., 2008.

[36] B. Silver, *BPMN Method and Style: A levels-based methodology for BPM process modeling and improvement using BPMN 2.0*. Cody-Cassidy Press, 2009.

[37] “6 Example of a business process model with exceptions | Download Scientific Diagram.” Accessed: Dec. 19, 2024. [Online]. Available: https://www.researchgate.net/figure/Example-of-a-business-process-model-with-exceptions_fig3_226414616

[38] T. Allweyer, *BPMN 2.0: Introduction to the Standard for Business Process Modeling*. Books on Demand, 2016.

[39] “What is the BPMN 2.0 Standard? | ProcessMaker.” Accessed: Dec. 19, 2024. [Online]. Available: https://www.processmaker.com/blog/what-is-the-bpmn-2-0-standard/

[40] “The Universal Process Orchestrator | Camunda.” Accessed: Dec. 12, 2024. [Online]. Available: https://camunda.com/

[41] “Bonitasoft : Open Source BPM software - Business Process Management.” Accessed: Dec. 18, 2024. [Online]. Available: https://www.bonitasoft.com/

[42] “Flowable Platform | Low-code Automation | Free Trial.” Accessed: Dec. 18, 2024. [Online]. Available: https://www.flowable.com/product/platform?utm_source=Google&utm_medium=Paid

[43] M. Weske, *Business Process Management: Concepts, Languages, Architectures*. Springer, 2012. doi: 10.1007/978-3-642-25160-3_14.

[44] R. K. L. Ko, S. S. G. Lee, and E. W. Lee, “Business process management (BPM) standards: A survey,” *Business Process Management Journal*, vol. 15, no. 5, pp. 744–791, 2009.

[45] H. Leopold, J. Mendling, and O. Gunther, “Learning from Quality Issues of BPMN Models from Industry,” *IEEE Softw*, vol. 33, no. 4, pp. 26–33, Jul. 2016, doi: 10.1109/MS.2015.81.

[46] J. Recker, “Opportunities and constraints: the current struggle with BPMN,” *Business Process Management Journal*, vol. 16, no. 1, pp. 181–201, Feb. 2010, doi: 10.1108/14637151011018001.

[47] L. Pufahl, F. Zerbato, B. Weber, and I. Weber, “BPMN in healthcare: Challenges and best practices,” *Inf Syst*, vol. 107, p. 102013, Jul. 2022, doi: 10.1016/J.IS.2022.102013.

[48] G. Hohpe and B. Woolf, “Enterprise Integration Patterns,” 2003.

[49] S. Aier and R. Winter, “Fundamental Patterns for Enterprise Integration Services,” *International Journal of Service Science, Management, Engineering, and Technology*, vol. 1, no. 1, pp. 33–49, Jan. 2010, doi: 10.4018/jssmet.2010010103.

[50] A. Curl and K. Fertalj, “A review of enterprise IT integration methods,” *Proceedings of the ITI 2009 31st International Conference on Information Technology Interfaces*, pp. 107–112, 2009, doi: 10.1109/ITI.2009.5196063.

[51] “Introduction to Message Routing - Enterprise Integration Patterns.” Accessed: Dec. 19, 2024. [Online]. Available: https://www.enterpriseintegrationpatterns.com/patterns/messaging/MessageRoutingIntro.html

[52] D. Ritter, S. Rinderle-Ma, M. Montali, A. Rivkin, and A. Sinha, “Formalizing Application Integration Patterns,” in *2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC)*, IEEE, Oct. 2018, pp. 11–20. doi: 10.1109/EDOC.2018.00012.

[53] T. Moser, H. Roth, S. Rozsnyai, and S. Biffl, “Implementing Enterprise Integration Patterns Using Open Source Frameworks,” *Software Engineering Techniques in Progress*, pp. 111–124, 2008, Accessed: Dec. 02, 2024. [Online]. Available: https://repositum.tuwien.at/handle/20.500.12708/52649

[54] M. J. Page *et al.*, “The PRISMA 2020 statement: an updated guideline for reporting systematic reviews,” *BMJ*, vol. 372, 2021, doi: 10.1136/bmj.n71.

[55] L. Z. Atkinson and A. Cipriani, “How to carry out a literature search for a systematic review: a practical guide,” *BJPsych Adv*, vol. 24, no. 2, pp. 74–82, 2018, doi: 10.1192/bja.2017.3.

[56] D. Ritter, “Application Integration Patterns and their Compositions: Foundations, Formalizations, Solutions,” 2019.

[57] J. A. Zachman, “A framework for information systems architecture,” *IBM Systems Journal*, vol. 38, no. 2.3, pp. 454–470, Dec. 1999, doi: 10.1147/sj.382.0454.

[58] L. Gang and W. Quan, “Aggregated framework of enterprise information system based on synergic theory,” in *Proceedings of the 7th International Conference on Web Information Systems*, in WISE’06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 216–222. doi: 10.1007/11906070_21.

[59] J. C. Delgado, “Supporting Enterprise Integration with a Multidimensional Interoperability Framework,” *Int. J. Soc. Organ. Dyn.*, vol. 4, no. 1, pp. 39–70, Jan. 2015, doi: 10.4018/IJSODIT.2015010104.

[60] A. Karande and P. Joshi, “Common quality measures for Enterprise Architecture,” in *2022 22nd International Conference on Computational Science and Its Applications (ICCSA)*, Jul. 2022, pp. 41–45. doi: 10.1109/ICCSA57511.2022.00017.

[61] C. Gui-sheng, “On Integrated Governance of e-Government: Technology, Institution, Market, and Government,” in *2010 International Conference on E-Business and E-Government*, May 2010, pp. 4212–4215. doi: 10.1109/ICEE.2010.1058.

[62] D. Van Nuffel and M. De Backer, “Multi-abstraction layered business process modeling,” *Comput. Ind.*, vol. 63, no. 2, pp. 131–147, Feb. 2012, doi: 10.1016/j.compind.2011.12.001.

[63] T. Weilkiens, J. G. Lamm, S. Roth, and M. Walker, “Architecture Frameworks,” in *Model-Based System Architecture*, Wiley, 2022, pp. 279–299. doi: 10.1002/9781119746683.ch19.

[64] D. Wissal, D. Karim, and K. Laila, “Adaptive Enterprise Architecture: Initiatives and Criteria,” in *2020 7th International Conference on Control, Decision and Information Technologies (CoDIT)*, Jun. 2020, pp. 557–562. doi: 10.1109/CoDIT49905.2020.9263891.

[65] M. K. Haki and M. W. Forte, “Service Oriented Enterprise Architecture Framework,” in *2010 6th World Congress on Services*, Jul. 2010, pp. 391–398. doi: 10.1109/SERVICES.2010.39.

[66] M. von Rosing, M. Hove, R. S. Rao, and T. W. Preston, “Combining BPM and EA in Complex IT Projects: (A Business Architecture Discipline),” in *2011 IEEE 13th Conference on Commerce and Enterprise Computing*, Sep. 2011, pp. 271–278. doi: 10.1109/CEC.2011.49.

[67] E. G. Perdana, B. Sitohang, H. S. Sastramihardja, and M. Z. C. Candra, “A Strategy Framework For Incorporating Sustainability Into Enterprise Architecture,” in *2020 8th International Conference on Information and Communication Technology (ICoICT)*, Jun. 2020, pp. 1–6. doi: 10.1109/ICoICT49345.2020.9166373.

[68] J. Petrikina, P. Drews, I. Schirmer, and K. Zimmermann, “Integrating Business Models and Enterprise Architecture,” in *2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations*, Sep. 2014, pp. 47–56. doi: 10.1109/EDOCW.2014.16.

[69] C. Emmersberger and F. Springer, “Tutorial: open source enterprise application integration - introducing the event processing capabilities of apache camel,” in *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems*, in DEBS ’13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 259–268. doi: 10.1145/2488222.2488269.

[70] D. Ritter, N. May, and S. Rinderle-Ma, “Patterns for emerging application integration scenarios,” *Inf. Syst.*, vol. 67, no. C, pp. 36–57, Jul. 2017, doi: 10.1016/j.is.2017.03.003.

[71] S. Andreozzi, M. Canaparo, and M. Carpenè, “GLUEMan: a WBEM-based Framework for Information Providers in Grid Services,” in *2008 12th Enterprise Distributed Object Computing Conference Workshops*, Sep. 2008, pp. 377–384. doi: 10.1109/EDOCW.2008.34.

[72] S. Bakhaev, “Theoretical frameworks for information systems integration in inter-firm collaborations,” 2021, Accessed: Dec. 10, 2024. [Online]. Available: https://lutpub.lut.fi/handle/10024/163103

[73] R. Mukherji, C. Egyhazy, and M. Johnson, “Architecture for a large healthcare information system,” *IT Prof*, vol. 4, no. 6, pp. 19–27, Nov. 2002, doi: 10.1109/MITP.2002.1114843.

[74] W. Schramm, H. Köstinger, K. Bayrhammer, M. Fiedler, and T. Grechenig, “Developing a hospital information system ecosystem for creating new clinical collaboration methodologies,” in *Proceedings of 2012 IEEE-EMBS International Conference on Biomedical and Health Informatics*, Jan. 2012, pp. 101–103. doi: 10.1109/BHI.2012.6211517.

[75] M. Caldeira, C. Pedron, G. Dhillon, and J. Lee, “Applying EA Perspective to CRM: Developing a Competency Framework,” in *2008 Third International Conference on Convergence and Hybrid Information Technology*, Nov. 2008, pp. 1029–1034. doi: 10.1109/ICCIT.2008.377.

[76] D. Ramljak, “Improving the composition and assembly of services in process-oriented environments,” in *2008 16th International Conference on Software, Telecommunications and Computer Networks*, Sep. 2008, pp. 341–345. doi: 10.1109/SOFTCOM.2008.4669507.

[77] D. Q. Birkmeier, A. Gehlert, S. Overhage, and S. Schlauderer, “Alignment of Business and IT Architectures in the German Federal Government: A Systematic Method to Identify Services from Business Processes,” in *2013 46th Hawaii International Conference on System Sciences*, Jan. 2013, pp. 3848–3857. doi: 10.1109/HICSS.2013.77.

[78] A. Santana, K. Fischbach, and H. Moura, “Enterprise Architecture Analysis and Network Thinking: A Literature Review,” in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, IEEE, Jan. 2016, pp. 4566–4575. doi: 10.1109/HICSS.2016.567.

[79] R. Klimek, Ł. Faber, and M. Kisiel-Dorohinicki, “Verifying data integration agents with deduction-based models,” in *2013 Federated Conference on Computer Science and Information Systems*, Sep. 2013, pp. 1029–1035.

[80] G. Camposo, “Welcome to Apache Camel,” *Cloud Native Integration with Apache Camel*, pp. 1–37, 2021, doi: 10.1007/978-1-4842-7211-4_1.

[81] “Apache Kafka.” Accessed: Dec. 21, 2024. [Online]. Available: https://kafka.apache.org/

[82] “RabbitMQ: One broker to queue them all | RabbitMQ.” Accessed: Dec. 21, 2024. [Online]. Available: https://www.rabbitmq.com/

[83] “Prometheus - Monitoring system & time series database.” Accessed: Dec. 21, 2024. [Online]. Available: https://prometheus.io/

[84] “Vault by HashiCorp.” Accessed: Dec. 21, 2024. [Online]. Available: https://www.vaultproject.io/

[85] “What is a REST API?” Accessed: Dec. 21, 2024. [Online]. Available: https://www.redhat.com/en/topics/api/what-is-a-rest-api

[86] “SOAP- Documentação da IBM.” Accessed: Dec. 21, 2024. [Online]. Available: https://www.ibm.com/docs/pt-br/integration-bus/10.0?topic=services-what-is-soap

[87] “Grafana Observability platform overview.” Accessed: Dec. 21, 2024. [Online]. Available: https://grafana.com/products/cloud/?src=ggl-s&mdm=cpc&camp=b-grafana-exac-emea&cnt=118483912276&trm=grafana&device=c&gad_source=1&gclid=CjwKCAiA65m7BhAwEiwAAgu4JK_DyI7B7t-Grbv3mifQ_CWL5DVcHg_IT4DIPFjspzmfYkhkuzOSoxoCfIMQAvD_BwE

[88] “Home | C4 model.” Accessed: Sep. 23, 2025. [Online]. Available: https://c4model.com/

[89] “What is FURPS+? – Business Analyst Training in Hyderabad – COEPD.” Accessed: Sep. 27, 2025. [Online]. Available: https://businessanalysttraininghyderabad.wordpress.com/2014/08/05/what-is-furps/

[90] “Goal-Question-Metrics (GQMs) | Definition and Overview.” Accessed: Sep. 26, 2025. [Online]. Available: https://www.productplan.com/glossary/goal-question-metrics/

[91] “Quality, functional and non-functional requirements in software development - News, tips & guidance for agile, development, Atlassian-Software (JIRA, Confluence, Bitbucket, ...) and Google Cloud.” Accessed: Sep. 26, 2025. [Online]. Available: https://seibert.group/blog/en/quality-functional-and-non-functional-requirements-in-software-development/

[92] “How do I score Risk in a Risk Log? - Business Best Practice.” Accessed: Dec. 23, 2024. [Online]. Available: https://business-docs.co.uk/scenario/how-to-score-risk-in-a-risk-log/

[93] G. David, D. R. Zmaranda, R. S. Gyorodi, and C. A. Gyorodi, “Exploring the Impact of Workflow Engines on Business Process Management in Enterprise Applications. A case-study: Camunda,” *2023 17th International Conference on Engineering of Modern Electric Systems, EMES 2023*, 2023, doi: 10.1109/EMES58375.2023.10171706.

[94] K. Kluza, K. Kaczor, G. J. Nalepa, and M˙lazyński, “Opportunities for business process semantization in open-source process execution environments,” *Proceedings of the 2015 Federated Conference on Computer Science and Information Systems, FedCSIS 2015*, pp. 1307–1314, 2015, doi: 10.15439/2015F250.

[95] N. Bartmann, S. Hill, C. Corea, C. Drodt, and P. Delfmann, “Applied Predictive Process Monitoring and Hyper Parameter Optimization in Camunda,” *Lecture Notes in Business Information Processing*, vol. 424 LNBIP, pp. 129–136, 2021, doi: 10.1007/978-3-030-79108-7_15.

[96] J. Recker, M. Indulska, P. Green, A. Burton-Jones, and R. Weber, “Information Systems as Representations: A Review of the Theory and Evidence,” *J Assoc Inf Syst*, vol. 20, no. 6, pp. 735–786, 2019, doi: 10.17705/1jais.00550.

[97] N. MOMTSELIDZE and A. TSITSAGI, “Apache Kafka - Real-time Data Processing,” *Journal of Technical Science and Technologies*, vol. 4, no. 2, pp. 31–34, May 2015, doi: 10.31578/JTST.V4I2.80.

[98] C. Corea and P. Delfmann, “A Tool to Monitor Consistent Decision-Making in Business Process Execution,” *International Conference on Business Process Management*, 2018.

Appendix A

This appendix provides an evaluation of the author's current skills, emphasizing strengths and identifying areas for development, critical to the successful execution of this project. This analysis focuses on technical, analytical, communication, and project management skills, all of which are essential for achieving the implementation of the proposed framework. By integrating insights from academic accomplishments, professional experiences, and recent feedback, this evaluation provides a holistic view of the author's capabilities and readiness for the project's demands.

Table 8 Skills Matrix

Skill	**Required Proficiency**	**Current Proficiency**	**Gap**	**Comments**
BPMN (Business Process Modeling and Notation)	5	4	Low	Solid knowledge of BPMN. Needs advanced expertise in designing executable workflows and integrating BPMN models with tools like Camunda.
Enterprise Integration Patterns (EIPs)	5	3	High	Basic understanding of core patterns (e.g., message routing). Needs hands-on experience applying EIPs in middleware like Apache Camel.
Data Governance and Security (GDPR Compliance)	4	3	Medium	Familiar with basic security principles; needs practical application in integrated systems.
Middleware Technologies	5	3	High	Minimal exposure to tools like Kafka and apache camel.
Integration Testing and Validation	5	3	Medium	Experience with testing tools like Postman; needs validation of complex integrated workflows.
Problem-Solving	5	5	None	Excellent ability to tackle complex technical challenges like system fragmentation and redundancy. Leverage this strength to identify and resolve project bottlenecks.
Collaboration and Stakeholder Engagement	4	3	Low	Familiar with engaging stakeholders; needs improvement in managing expectations and collaboration.
Communication (Oral and Written)	4	3	Medium	Can articulate technical ideas; needs improvement in simplifying technical content for non-technical audiences.
Project Management	4	3	Low	Experience managing small-scale projects; needs formalization in tracking milestones and risks.

The results of the evaluation are summarized in Table 8, which outlines the required proficiency levels, current skills, identified gaps, and detailed observations on areas needing improvement. This structured analysis serves as the foundation for a targeted plan to enhance the skills necessary for the effective completion of the project.

Identification of Required Skills

The successful execution of this project requires a diverse and well-developed skills set encompassing technical expertise, analytical abilities, effective communication, and project management. BPMN is essential for designing workflows within the proposed framework, enabling structured representation of processes that ensure efficiency and adaptability. Without a high level of BPMN expertise, the research might not be as clear and precise as required for the framwork's practical application, which could lead to ambiguities.

To achieve interoperability across heterogeneous systems, Enterprise Integration Patterns (EIPs) are also essential. These patterns offer the means to create and carry out reliable integration solutions, and they are frequently implemented with the assistance of middleware such as Apache Camel. The suggested framework's scalability and ability to manage the complexity of real-world information systems are guaranteed by mastery of EIPs.

For integrated systems to be implemented in practice, Camunda, a widely used open-source platform, is necessary. It is crucial for streamlining processes, ensuring operational efficiency, and automating workflows created with BPMN. Mastery of this tool provides the foundation required to convert process models into executable workflows, closing the gap between theoretical frameworks and practical implementation. By doing so, Camunda certifies that the functionality and design of the suggested system can be successfull implemented within the organizational environment.

Presenting research findings to both technical and non-technical audiences requires strong oral and written communication skills. A wide range of readers, including stakeholders who are unfamiliar with the field, can understand the dissertation thanks to clear technical writing. Similarly, oral communication is essential for presentations because it enables the writer to convey complex ideas with assurance and precision.

Middleware technologies like Kafka and Apache Camel, which enable distributed processing and the smooth transfer of data between components, are also essential for effective system integration. These tools guarantee robust interoperability by enhancing the framework's scalability and dependability. Assuring the accuracy and dependability of the system also requires integration testing and validation with Postman and other tools to find and fix possible problems in intricate workflows. Particularly with regard to GDPR compliance, data governance and security ensure that the framework complies with ethical and legal standards while safeguarding sensitive data. Finally, while project management and stakeholder engagement ensure the research is carried out efficiently, balancing timelines, resources and expectations, strong communication skills allow the findings to be clearly articulated to a variety of audiences. Together, these skills form the foundation for delivering a practical, scalable, and legally compliant solution aligned with both academic and professional objectives.

Assessment of Current Skills

An evaluation of the author's current skills reveals a solid foundation in key areas, alongside several opportunities for growth. A solid grasp of theoretical concepts acquired through academic training underpins established proficiency in BPMN. To guarantee flexibility and accuracy in real-world situations, more work must be done to apply these concepts in dynamic and useful contexts, such as workflow modeling for various and intricate business processes.

The author shows that they understand the fundamentals of Enterprise Integration Patterns (EIPs), but they have no real-world experience putting these patterns into practice with middleware tools like Apache Camel. This gap limits the ability to efficiently design and implement robust integration solutions, which is necessary to achieve the framework's goal of seamless interoperability across heterogeneous systems.

The author demonstrates a strong foundational readiness to use this technology by being familiar with open-source tools such as Camunda. Even though Camunda's core characteristics have been studied, more incorporation into the experimental environment is necessary to confirm competence and strengthen operational knowledge. In addition to ensuring a more sophisticated use of the platform, this progression would bring theoretical design principles and the actual orchestration of enterprise processes closer together.

Other technical areas that require attention are middleware technologies, such as Apache Kafka and Camel, which are essential for reliable distributed processing and seamless data transfer between system components. Due to the author's limited exposure to these tools, practical practice is necessary to reach the necessary level of proficiency. Likewise, additional experience is needed for integration testing and validation, a crucial step in confirming the framework's resilience. Although the author is aware of Postman and other similar tools, there is still room for improvement in terms of deeper validation methods for complex workflows.. Familiarity with data governance and security principles, particularly GDPR compliance, is evident but requires practical application to ensure legal and ethical adherence in the implemented framework.

Another area that needs work is communication skills. Although the author exhibits technical writing proficiency, more work is required to improve coherence and clarity, especially when speaking to a variety of audiences. When presenting to both technical and non-technical stakeholders, oral communication is adequate, but it still requires improvement in terms of accuracy and confidence.

Finally, although project management skills are sufficient for smaller-scale projects, they must be formalized to handle the complexity of a large-scale dissertation. This includes developing structured methodologies for setting and tracking milestones, assessing risks, and managing collaborative workflows effectively. Another crucial element is stakeholder engagement, where the author needs to improve their capacity to control expectations and promote fruitful cooperation.

Strategies for Skill Development

Addressing the identified skill gaps requires a structured approach that combines theoretical learning with hands-on practical application. The author will concentrate on developing intricate workflow models that mimic actual business situations in order to improve BPMN proficiency.This will involve iterative modeling exercises, peer reviews, and critical evaluation to refine techniques and improve the adaptability of workflows for diverse and complex systems. Converting theoretical knowledge into workflows that satisfy realistic, practical needs is the aim.

Enterprise Integration Patterns (EIPs) will be developed through hands-on projects involving middleware tools such as Apache Camel. Through these projects, which will focus on developing and implementing integration solutions, the author will be able to obtain hands-on experience managing intricate system interoperability issues.

This technology will be incorporated into the framework's experimental phase to enhance competence with open-source tools like Camunda. Camunda will be employed to automate workflows modeled in BPMN, ensuring seamless process automation and operational efficiency. The author's proficiency in workflow orchestration will be strengthened by iterative testing and improvement throughout the experimental phase, which will also strengthen the link between theoretical design principles and their application in real-world integration scenarios.

The ability to use middleware technologies like Apache Kafka and Camel will be strengthened through targeted hands-on exercises. To guarantee scalability and dependability, these exercises will entail configuring distributed processing environments and modeling data flow among system components. Additionally, integration testing and validation will be given top priority. Complex workflows will be validated and areas that need improvement will be highlighted using Postman and other tools. This will guarantee the scalability and resilience of the solutions put into place.

To fill in the gaps in data governance and security, the framework's development will take into account the real-world implementation of GDPR compliance principles. This will entail assessing how the system handles private information and making sure that moral and legal requirements are met. Data integrity and privacy will be protected while the framework satisfies important compliance requirements thanks to this focused attention to data governance.

Weekly review sessions centered on editing important technical documents for impact, coherence, and clarity will enhance communication skills. Constructive feedback from advisors and peers will serve as the basis for these adjustments, which will improve the author's work's effectiveness and accessibility. Regular mock presentations given to a variety of audiences will improve oral communication by highlighting the simplification of technical material and the growth of delivery confidence. Based on feedback from these sessions, the presentation's content and style will be refined.

The use of cutting-edge techniques and tools will formalize project management capabilities. Gantt charts will be used for timeline tracking in order to monitor progress and ensure that deadlines are fulfilled. Effective resource allocation, dependency management, and milestone tracking will all be accomplished with the use of project management software. Priority will also be given to stakeholder engagement and collaboration, with an emphasis on managing expectations and encouraging open communication to ensure successful partnerships.

Summary – How skills lead to project success

The successful development of the identified skills is essential to the quality and overall success of this work, as they form the backbone of the proposed framework and its contributions to the fields of business process modeling and integrated information systems. Each ability is essential to achieve the framework's objectives, bridging the gap between theory and practice, and meeting the demands of both academic rigor and practical applicability.

Developing scalable, flexible, and structured workflows requires the use of BPMN. These workflows give the framework a solid foundation on which to address important problems like system fragmentation and operational inefficiencies. Without a solid command of BPMN, the framework may not be as precise and clear as it needs to be in order to meet real-world problems. A deep comprehension of BPMN ensures that the framework's process designs are not only conceptually sound but also practical and directly applicable to the complex environments it seeks to enhance.

Building upon this framework, Enterprise Integration Patterns (EIPs) are essential to attaining smooth interoperability. EIPs allow the system to work together in spite of different structures and technologies by offering ways to connect various components.. This ability is critical for ensuring that the framework is reliable and scalable, making it suitable for addressing the complexities of integrated information systems. EIPs make it easier for structured workflows created with BPMN to run smoothly across a range of technological environments.

Camunda strengthens the framework by transforming its theoretical models into functional systems. Workflows created with BPMN are automated, which improves operational dependability and efficiency while maintaining process transparency and executability. By integrating, the framework goes beyond conceptualization and provides useful, implementable solutions that tackle the intricacies of contemporary business process problems.

Strong communication skills guarantee that the framework's conclusions are successfully communicated to the target audiences, even though technical skills facilitate the framework's design and implementation. Both academic and professional stakeholders can understand the framework's intricate ideas thanks to written communication, which makes them easy to understand and precise. Similarly, oral communication facilitates the dissemination of these findings through assured, captivating presentations that promote cooperation and seek feedback.

Project management skills, which offer the framework required to carry out the research efficiently, are equally important. The author guarantees that the work stays on track and in

line with its goals by using techniques like risk management and milestone tracking. These abilities enhance the research and its conclusions by promoting stakeholder collaboration and participation, which enables the inclusion of various viewpoints and contributions.

In conclusion, the abilities listed in this section provide a cohesive basis that guarantees the effectiveness of the suggested framework. Each skill plays a distinct and complementary role in addressing the research objectives and transforming theoretical concepts into practical outcomes.

Appendix B

This appendix outlines the project planning and control processes for developing the Business Process Modeling Framework for Integrated Information Systems**.** A detailed project plan enhances the likelihood of success by ensuring a structured, methodical approach to achieving research objectives and managing associated challenges.

Stakeholders

The active participation and influence of important stakeholders, each of whom makes a unique contribution to the research, are essential to the success of this dissertation.

The primary leaders of the project are the supervisor and the researcher. In order to ensure that the dissertation's goals are achieved with accuracy and rigor, the researcher bears the primary responsibility for defining its scope, methodology, and execution. On the other hand, the supervisor, provides indispensable guidance, ensuring alignment with academic standards, steering the research toward meaningful contributions, and offering critical feedback throughout the project.

Organizations and enterprises constitute another important stakeholder group. They stand to benefit directly from the proposed framework, which addresses the urgent need for seamless system integration and operational efficiency. These stakeholders are especially interested in the practical applications of the research, as it provides solutions to existing challenges related to workflow automation and system interoperability.

Software engineers and developers are an important audience for the project outcomes, even though they are not directly involved in the research process. They are interested because the suggested framework may be useful to them in their line of work, especially when it comes to automating procedures and combining various technologies. Software architects are also interested in this research. It is anticipated that the dissertation's insights will guide their approaches to creating scalable and effective systems that tackle the challenges of integration in contemporary technological environments.

The academic community also holds a prominent position among the stakeholders. The dissertation aims to contribute to the body of knowledge in the field, offering theoretical advancements in system integration methodologies. Academics and researchers are anticipated to engage with the findings, using them as a foundation for additional research or to improve current procedures.

Open-source communities, while more peripheral, stand to benefit from the outcomes of the research. By enhancing modularity and interoperability, the dissertation supports the collaborative nature of open-source development, which may encourage wider adoption and innovation within these communities.

Finally, regulatory and compliance bodies indirectly influence the project by shaping the ethical and procedural considerations underlying the research. Following their instructions guarantees that the project complies with ethical standards and best practices for data handling.

Through understanding of the roles and expectations of these diverse stakeholders, the dissertation bridges academic inquiry with practical, real-world applications, ensuring its relevance and utility across multiple domains.

Costs

Since the main goal of this dissertation is research, direct costs are unlikely to occur. However, there may be some indirect costs, such as software licenses for development or analytical tools and access to scholarly literature or journals. Even though access to a multitude of resources is made possible by institutional access, there may be extra costs associated with buying specialized software or other materials that are not part of the institution's collection.

The study makes extensive use of institutional resources and open-source technologies to reduce these expenses, guaranteeing that the project stays affordable while upholding high standards of relevance and quality.

Assumptions and Restrictions

The execution of this dissertation is framed by critical assumptions and operational restrictions, which define the project's scope and feasibility.

Assumptions

The research is built on the following key assumptions, which guide its methodology and objectives:

- **Adequacy of Open-Source Tools**: The chosen open-source tools, including Camunda, and Apache Camel, are assumed to possess the necessary functionality, scalability, and reliability to support the development and testing of the framework. These tools are expected to align with industry standards and academic research requirements.
- **Relevance of Simulated Data**: Simulated datasets are presumed to accurately replicate real-world conditions and behaviours, enabling effective validation of the framework. This assumption ensures that the absence of proprietary or live datasets does not hinder the reliability of the findings.
- **Feasibility of the Project Timeline**: The project timeline is expected to remain achievable within the constraints of the academic calendar. It is assumed that unforeseen technical challenges or resource limitations will not significantly disrupt progress.
- **Technological Stability**: The core technologies and frameworks used in the project are assumed to remain relevant and stable throughout the research period, minimizing the risk of obsolescence.

- **Ethical Compliance**: To ensure that the research complies with institutional and legal requirements, it is assumed that all data sources and tools used in the project adhere to ethical standards and data protection regulations.

Restrictions

The research is subject to the following restrictions, which delineate its boundaries and impact its scope:

- **Strict Deadlines**: The dissertation must be submitted by the specified dates for reports, drafts, and the final presentation. These timelines impose constraints on the flexibility of extending specific phases or tasks.
- **Budget Constraints**: Financial limitations restrict access to certain premium tools, advanced infrastructure, or external expertise. The research primarily relies on open-source tools and institutional resources to mitigate these constraints.
- **Data Accessibility**: The study is confined to the use of publicly available or simulated datasets due to limited access to proprietary or real-world organizational data. This restricts the ability to validate the framework against live operational scenarios.
- **Institutional Requirements**: The project must comply with institutional guidelines regarding the format, structure, and submission of the dissertation. These requirements influence how the research is documented and presented.
- **Dependency on Open-Source Ecosystem**: The success of the framework depends on the capabilities and limitations of the selected open-source tools. Issues such as compatibility, lack of features, or unsupported updates in these tools may restrict the framework's development.
- **Supervisor Availability and Feedback**: The progress of the research is tied to the timely feedback and availability of the supervisor. Delays in receiving feedback could impact the refinement and delivery of key outputs.
- **Technology Scope**: The research is limited to exploring technologies within the predefined scope of the study, including BPMN, Apache Camel, and related frameworks. The adoption of alternative tools or methodologies outside this scope is restricted due to time and resource constraints.

Acknowledging these assumptions and restrictions ensures transparency in the project's planning, allowing for realistic expectations and effective risk management.

Risk Assessment

Effective risk assessment and management are essential to navigating uncertainties and ensuring project success. Identified risks are evaluated based on their probability of occurrence and potential impact, forming the basis of the risk management strategy.

Figure 80 Risk matrix Source:[92]

A key tool in risk management is the Risk Matrix, illustrated in Figure 80. This matrix evaluates the seriousness of risks based on two main dimensions:

- **Probability**: The likelihood of a risk occurring, rated from 1 (low probability) to 5 (high probability).
- **Impact**: The potential consequences if a risk materializes, rated from 1 (minor impact) to 5 (major impact).

The combination of these dimensions yields a Probability-Impact (PI) Score, categorizing risks into four levels:

- **Low Risk (Green)**: Minimal likelihood or impact, requiring minimal oversight.
- **Medium Risk (Yellow)**: Calls for monitoring and moderate mitigation efforts.
- **Medium High Risk (Orange)**: Needs active management and mitigation measures to reduce impact or probability.
- **High Risk (Red)**: Critical risks demanding immediate attention and comprehensive response strategies.

Appendix G contains the comprehensive risk identification and corresponding action plans.

The risk table systematically identifies potential risks that are pertinent to this dissertation project, detailing their descriptions, root causes, and potential effects on project timelines and deliverables. Additionally, it designates a risk owner, calculates the PI Score to prioritize risks, and specifies tailored response strategies to address each one. Risks such as missed deadlines, tool incompatibility, and data scarcity are particularly emphasized due to their potential to disrupt progress and reduce the quality of findings.

The table guarantees that high-priority issues, like integration difficulties and stakeholder delays, are addressed with workable solutions by classifying risks based on their severity. To

proactively mitigate integration challenges, such as those brought on by workflow inconsistencies or architectural mismatches, thorough compatibility testing is carried out during the integration phase. Flexible review sessions are planned in the interim to keep things moving forward and decision-making effective despite stakeholder delays caused by supervisors' limited availability.

All things considered, the risk table is a dynamic framework for ensuring the dissertation's success rather than just a static tool. The project becomes resilient even in the face of difficulties by promoting a proactive approach to identifying, prioritizing, and resolving uncertainties. The risk assessment framework guarantees that the project keeps its momentum, maintains the caliber of the research, and accomplishes its goals within the allotted time by methodically addressing high-impact risks and putting flexible strategies for medium and low-priority issues into place.

.Project Planning

The project planning for this dissertation ensures a structured and systematic approach to accomplishing the research objectives. By segmenting the dissertation into precisely defined phases, tasks, and milestones, it guarantees clarity, progress monitoring, and timeline adherence. This structured approach helps address the scope, resources, and deliverables efficiently. A thorough explanation of each stage is provided below, along with pertinent information and planning document notes.

Work Breakdown Structure

The Work Breakdown Structure (WBS) is a crucial step in the project planning process, offering a detailed framework for organizing the dissertation into smaller, actionable components. Through precise task and deliverable definitions, the WBS offers a well-organized plan that makes it easier to allocate resources and execute projects efficiently, all the while keeping the project on course to meet its goals.

The WBS facilitates smooth progress tracking and the identification of task interdependencies while acting as a guide for managing complexity. It enables stakeholders to pinpoint potential challenges early and make necessary adjustments without disrupting the overall flow. Table 9 outlines the WBS, highlighting the full scope of work and providing a clear representation of how the project is systematically structured to meet its goals.

Table 9 Work Breakdown Structure

WBS	Task Name	WBS Level
1	**Dissertation Project**	**Project**
1.1	**Dissertation Preparation**	**Phase**
1.1.1	**State-of-the-Art**	**Deliverable**
1.1.1.1	Background and Standards	Task
1.1.1.2	Technologies Identification	Task
1.1.2	**Research Phase**	**Phase**
1.1.2.1	Research Questions	Task
1.1.2.2	Discussion	Task
1.1.3	**Planning Phase**	**Phase**
1.1.3.1	Project Charter	Deliverable
1.1.3.2	Work Breakdown Structure	Deliverable
1.1.3.3	Detailed Project Plan	Deliverable
1.1.3.4	Skills Management	Deliverable
1.1.4	Dissertation Preparation Report	Deliverable
1.1.5	Dissertation Preparation Report Validation	Task
1.1.6	Final Dissertation Preparation Report	Deliverable
1.1.7	Dissertation Preparation Presentation	Deliverable
1.2	**Framework Proposal Phase**	**Phase**
1.2.1	Framework Conception/Design	Task
1.2.2	Define Selection Criteria and Evaluate Tools	Task
1.2.3	Establish Testing Strategy	Task
1.2.4	Proposal for Construction Document	Deliverable
1.2.5	BPMN Improvement	Task
1.2.6	Enterprise Integration Patterns Improvement	Task
1.2.7	Apache Kafka and Camel Integration Prototype	Task
1.3	**Framework Construction Phase**	**Phase**
1.3.1	Environments and Tools Setup	Task
1.3.2	Implement Integration Modules	Task
1.3.3	Conduct Initial Prototype Testing	Task
1.3.4	Consolidate the Modular Framework	Task
1.3.5	Modular Framework Core	Deliverable
1.3.6	Camunda Improvement	Task
1.3.7	GDPR compliance principles Improvement	Task
1.4	**Evaluation Phase**	**Phase**
1.4.1	Test Scenarios Specification	Task
1.4.2	Perform Test Scenarios	Task
1.4.3	Collect Analyzed Results	Task
1.4.4	Evaluation Report	Deliverable
1.5	**Conclusions**	**Phase**
1.5.1	Report Validation	Task
1.5.2	Dissertation Report	Deliverable
1.5.3	Final presentation	Deliverable
1.5.4	Presentation Validation	Task
1.5.5	Presentation Preparation	Task

Project Schedule

This section details the project schedule based on academic and dissertation requirements, ensuring alignment with research goals and deliverable timelines. The schedule includes a clear structure of tasks, dependencies, and milestones. A full export of the Microsoft Project schedule is available in Appendix E.

The Dissertation Preparation Phase (16/09/24 – 06/02/25**)** forms the cornerstone of the entire project, laying a robust foundation through extensive research, planning, and preparation. This phase comprises several key tasks and deliverables designed to provide clarity and direction for the dissertation:

- State-of-the-Art (16/09/24 – 03/01/25): In order to create a thorough and critical assessment of the state-of-the-art in the selected dissertation topic, this task compiles all relevant data.. The deliverable highlights the current research landscape, identifying gaps and opportunities for the project. It is further broken down into the following sub-tasks:
    - Background and Standards (16/09/24 – 03/01/25): To fully comprehend the context of the dissertation, this task entails gathering and synthesizing foundational knowledge, including pertinent standards and protocols. The background serves as the theoretical underpinning for the entire project.
    - Technologies Identification (16/09/24 – 03/01/25): Identifying and evaluating the tools and technologies necessary for framework development. The analysis provides a roadmap for selecting and integrating these tools.
- Research Phase (04/11/24 – 03/01/25): This phase focuses on defining the research direction and exploring existing knowledge to lay a solid foundation for the dissertation. It involves critical analysis of the current academic landscape, ensuring that the research is well-aligned with gaps in knowledge and opportunities for contribution. The following sub-tasks are included in this phase:
    - Research Questions (04/11/24 – 30/11/24): This sub-task involves formulating clear and precise research questions that will guide the dissertation. These questions are designed to fill in gaps in the literature and support the goals of the dissertation.
    - Discussion (01/12/24 – 03/01/25): This assignment involves a careful examination of the research findings in order to offer well-reasoned responses to the questions presented.
- Planning Phase (04/11/24 – 03/01/25): This phase includes three essential planning documents, each serving a specific function in ensuring the smooth progression of the project. These documents set out a roadmap that incorporates deadlines, resources, and important deliverables in addition to clarifying the project's goals and scope.
    - Project Charter (04/11/24 – 03/01/25): The project's goals, objectives, scope, and important stakeholders are all outlined in the charter. It establishes the

boundaries of what the project aims to achieve while coordinating these goals with academic requirements and expectations. The charter also specifies the primary responsibilities of each participant, ensuring alignment and accountability from the outset. This document serves as the project's official blueprint, providing a shared understanding among all parties involved.

  - Work Breakdown Structure (WBS) (04/11/24 – 03/01/25): The WBS is a hierarchical breakdown of the project into smaller, easier-to-manage parts. It offers a thorough breakdown of all the tasks and subtasks needed to finish the dissertation. The WBS guarantees that no task is missed and permits effective use of time and resources by segmenting the project into smaller components. This deliverable also aids in tracking progress throughout the project, as each component can be monitored individually to ensure adherence to the planned timeline.

  - Detailed Project Plan (04/11/24 – 03/01/25): This plan includes specific timelines, task dependencies, required resources, and methodologies to be used. In order to guarantee that the project remains on course even in the event of unforeseen difficulties, it identifies possible risks and provides mitigation strategies. The plan also includes milestones and checkpoints to evaluate progress at critical junctures, allowing for adjustments and refinements as necessary.

- Dissertation Preparation Report (04/11/24 – 13/12/24): To make it possible to create a coherent document, this deliverable combines the important discoveries and results from the planning and state-of-the-art stages. The report establishes the theoretical foundation of the dissertation by presenting the insights derived from the literature review, the research questions, and the contextual background. These elements are complemented by the structured roadmap developed during the planning phase, outlining how the research will be executed and the skills management.

- Dissertation Preparation Report Validation (16/12/24 – 23/12/24): During this phase, the Dissertation Preparation Report undergoes a thorough review by the academic supervisor. This validation ensures the document meets academic rigor, aligns with research standards, and addresses any gaps or inconsistencies in its content. Feedback is given on areas that need to be improved, such as structure, clarity, depth of analysis, and alignment with the project’s goals. In addition to enhancing the work’s quality, this review provides the researcher with valuable guidance to refine their work and make sure it is prepared for final submission.

- Final Dissertation Preparation Report (24/12/24 – 31/12/24): The supervisor's comments are incorporated into the Final Dissertation Preparation Report, which produces a polished, comprehensive document that satisfies the highest requirements for academic rigor. This final iteration refines the theoretical framework, strengthens the narrative, and addresses any gaps identified during the validation phase. It acts as a definitive representation of the preparation phase, aligning the dissertation’s

objectives, methodologies, and plans while ensuring readiness for the next stages of the research process.

- Dissertation Preparation Presentation (01/01/25 – 03/01/25): This phase is concluded by the Dissertation Preparation Presentation, which provides an engaging and concise summary of the project's progress, findings, and preparation efforts. Important elements such as the research questions, theoretical foundation, methodologies, and project roadmap are highlighted in this presentation. It is a crucial step in getting feedback and guaranteeing alignment prior to moving on to later stages of the dissertation. It is intended to successfully communicate the results of the preparation phase to an academic audience.

By conceiving a modular framework and identifying the tools and methodologies necessary for its development, the Framework Proposal Phase (07/02/25 – 20/03/25) establishes the theoretical groundwork for the dissertation. This phase ensures that the project's design aligns with the research objectives, addressing both academic and practical considerations. The tasks in this phase provide a comprehensive basis for the subsequent framework construction.

- Framework Conception/Design (07/02/25 – 20/02/25): The conceptual design forms the foundation of the modular framework. This task involves defining the architecture, specifying core components, and identifying interactions between modules. A strong and future-proof framework is ensured by the design's emphasis on scalability, flexibility, and adherence to industry standards. In order to direct the construction phase, the output consists of architectural diagrams and illustrative documentation..

- Define Selection Criteria and Evaluate Tools (21/02/25 – 06/03/25): This task establishes clear selection criteria and ensures the tools and technologies chosen align with the project's technical requirements and constraints. This task involves identifying tools, evaluating their capabilities, and justifying their selection based on performance, compatibility, and scalability. The deliverable is a detailed evaluation report summarizing the rationale behind each tool's inclusion in the project.

- Establish Testing Strategy (07/03/25 – 13/03/25**):** To validate the framework, a robust testing strategy is developed. This task includes defining test scenarios, success metrics, and methodologies. The strategy ensures that the framework meets both functional and non-functional requirements. The deliverable is a testing plan outlining detailed procedures for validating the framework in subsequent phases.

- Proposal for Construction Document (07/03/25 – 20/03/25): This deliverable documents the conceptual framework and provides detailed guidelines for its construction. It serves as a bridge between theory and implementation, ensuring a seamless transition to the construction phase. The document includes design justifications, tool selection, and procedural outlines.

- BPMN Improvement (07/02/25 – 13/03/25): Enhancing skills with BPMN (Business Process Model and Notation) ensures the framework's processes align with

established modeling standards. This task involves hands-on practice and integration of BPMN tools, refining workflows for greater efficiency and clarity.

- Enterprise Integration Patterns Improvement (07/02/25 – 13/03/25): This task focuses on mastering enterprise integration patterns to design robust, modular architectures. Through practical applications and studies, this improvement ensures the framework incorporates best practices in software integration.

- Apache Kafka and Camel Integration Prototype (07/02/25 – 13/03/25**):** Developing a prototype for integrating Apache Kafka and Camel demonstrates the feasibility of the framework's design. This task involves hands-on experimentation and validation of integration strategies, ensuring the components interact as intended.

The Framework Construction Phase (21/03/25 – 08/05/25) represents a crucial transition from the theoretical foundation established in earlier phases to the practical implementation of the modular framework. This stage is dedicated to building, testing, and consolidating the framework to ensure it aligns with the project's technical and functional requirements. By the end of this phase, the framework will be operational and ready for rigorous evaluation in subsequent stages.

- Environments and Tools Setup (21/03/25 – 17/04/25): This task establishes the technical infrastructure needed for the framework's development and testing. It involves configuring development environments, installing necessary tools, and setting up virtual environments to facilitate seamless integration of components. Ensuring compatibility across all tools and systems is a priority, as it lays the groundwork for efficient and error-free framework construction.

- Implement Integration Modules (21/04/25 – 02/05/25): The implementation of integration modules is the core of the construction phase. Each module is designed and developed to adhere to the principles of modularity, scalability, and functionality. These modules are integrated into the larger framework, ensuring that they align with the theoretical design while meeting the practical needs of the project. This task is critical for transforming the conceptual framework into a functional system.

- Conduct Initial Prototype Testing (05/05/25 – 09/05/25): Initial testing of the prototype validates the integration and functionality of the modular components. Through carefully designed testing scenarios, this task identifies potential issues related to performance, compliance, or functionality. The feedback from these tests informs adjustments and refinements, ensuring the framework meets the design specifications and performs as intended.

- Consolidate the Modular Framework (10/04/25 – 24/04/25): The consolidation of the modular framework ensures all components are fully integrated and operate cohesively as a unified system. This task focuses on resolving dependencies, optimizing performance, and addressing any compatibility issues between modules. The result is a fully functional framework, ready to handle the demands of further testing and evaluation.

- Modular Framework Core (19/05/25 – 23/05/25): As the final deliverable of this phase, the Modular Framework Core represents the culmination of all design and construction efforts. This fully operational and adaptable framework encapsulates the project's modularization principles and serves as a robust solution for modular applications. It is prepared for rigorous evaluation, marking a significant milestone in the dissertation project.

The Evaluation Phase (09/05/25 – 20/06/25) is a crucial step that thoroughly evaluates the developed framework to make sure it satisfies the technical requirements. During this phase, the framework undergoes systematic testing and analysis to validate its functionality, performance, and adherence to design objectives. The results of this phase guarantee that the framework is prepared for academic presentation and real-world applications by confirming its efficacy and pointing out possible areas for improvement.

- Test Scenarios Specification (09/05/25 – 20/05/25):The process begins with defining detailed test scenarios that align with the testing strategy established in earlier phases. These scenarios outline the conditions under which the framework will be assessed, specifying success metrics to evaluate key attributes such as functionality, scalability, and performance. This task ensures a structured and systematic approach to validation, laying a strong foundation for reliable and consistent testing.

- Perform Test Scenarios (21/05/25 – 13/06/25): Once the test scenarios are specified, they are executed to assess the framework's capabilities in real-world-like conditions. The focus during this task is on evaluating the framework's functionality, integration between modular components, and its ability to handle varying workloads. Detailed documentation of the results ensures that each test is thoroughly analyzed and linked back to the success metrics defined earlier.

- Collect Analyzed Results (16/06/25 – 18/06/25): The collected test results are compiled and systematically analyzed to provide insights into the framework's performance. This task identifies strengths, such as seamless integration or robust functionality, and highlights areas requiring further optimization. The analysis offers a comprehensive view of the framework's readiness, guiding the refinement process where necessary.

- Evaluation Report (19/06/25 – 20/06/25): The phase concludes with the preparation of the Evaluation Report, a formal document that consolidates all findings from the testing and analysis process. This report summarizes the framework's overall performance, highlights key achievements, and provides actionable recommendations for future improvements. As a definitive assessment of the framework, the Evaluation Report ensures that the project is ready for final presentation and further development if required.

The Conclusions Phase (04/07/25 – 22/07/25) marks the final stage of the dissertation, where the entire project is synthesized, validated, and prepared for presentation. This phase focuses on refining the final deliverables and ensuring they meet the highest academic standards while clearly communicating the research findings and their significance. Through rigorous review

and presentation preparation, this phase ensures that the dissertation is ready for formal submission and defense.

- Report Validation (04/07/25 – 10/07/25): The Conclusions Phase begins with a thorough validation of the final Dissertation Report. This task involves a detailed review by supervisors and peers to ensure the report adheres to academic standards, is well-structured, and communicates its findings clearly and concisely. Feedback received during this process is used to refine the report further, ensuring it accurately reflects the research's depth and significance.
- Dissertation Report (11/07/25 – 17/07/25): The finalized Dissertation Report represents the culmination of all research, design, and evaluation efforts throughout the project. This comprehensive deliverable consolidates the project's theoretical foundation, methodologies, results, and conclusions. It is set up to highlight how the project answers the research questions and accomplishes its goals, as well as the researcher's contributions to the academic community.
- Final Presentation (04/07/25 – 10/07/25): The dissertation's core ideas are condensed into a clear and engaging format in the Final Presentation. It provides an overview of the research objectives, methodologies, findings, and their implications. Designed for an academic audience, this presentation is a critical opportunity to demonstrate the significance of the research and its potential contributions to the field.
- Presentation Validation (11/07/25 – 17/07/25): Once the presentation is prepared, it undergoes a validation process with supervisors to ensure clarity, coherence, and alignment with the research objectives. To make sure the presentation successfully conveys the project's main points and holds the attention of the audience, this review focuses on improving the presentation's content, visual aids, and delivery style.
- Presentation Preparation (18/07/25 – 22/07/25): The final task in this phase is preparing for the presentation delivery. This involves rehearsing the presentation, refining the delivery based on feedback, and ensuring confidence in addressing potential questions. This task ensures that the researcher is fully prepared to present their work with clarity and professionalism, marking the successful conclusion of the dissertation journey.

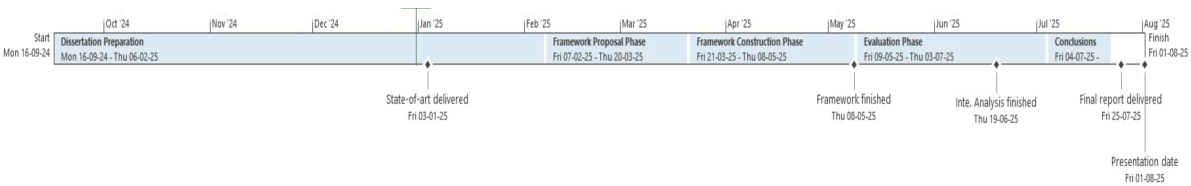


Figure 81 Project Schedule Baseline

Figure 81 illustrates the baseline project schedule and its milestones. This schedule is properly planned and detailed to align with the specific requirements of the experiment outlined in this dissertation. The Microsoft Porject Chart is present in Appendix F.

Appendix C

This appendix explores the evolution of Camunda [40], its functionalities, and its role in modern enterprises while integrating insights from leading research and practical applications.

Camunda

In today's fast-evolving digital landscape, businesses are under constant pressure to improve efficiency and streamline operations to stay competitive. The growing reliance on technology has made the integration of information systems and the automation of intricate workflows a top priority for organizations. To address these needs, Camunda, an open-source Business Process Management System (BPMS), has become a top option. Comunda, which is well-known for its flexibility, developer-oriented features, and adherence to widely used standards such as Business Process Model and Notation (BPMN), Decision Model and Notation (DMN), and Case Management Model and Notation (CMMN), is transforming the way organizations design and manage their processes.

Camunda has made a name for itself as a game-changing tool for decision management and workflow automation. Its ability to address the growing complexity of enterprise operations stems from its capacity to execute standardized process models efficiently, maintain scalability, and seamlessly integrate diverse systems. As highlighted by Ghiurau et al., workflow engines like Camunda enable enterprises to automate business processes effectively, lowering manual intervention and increasing flexibility [93]. This adaptability promotes innovation and operational efficiency by enabling businesses to react quickly to changing market demands.

According to research by Kluza et al. incorporating semantization techniques into open-source workflow engines like Camunda has the potential to further improve them [94]. Improved knowledge representation and collaboration between organizations may be made possible by such developments. Meanwhile, Berti et al. showed how Camunda'ss data extraction capabilities are improved by incorporating process mining features, opening the door to better operational and decision-making outcomes [10].

Furthermore, Bartmann et al. investigated predictive process monitoring in Camunda, showcasing how advanced analytics and hyper-parameter optimization can improve business processes. Their results demonstrate the value of combining machine learning techniques with workflow automation, offering organizations a competitive edge in data-driven environments [95].

Core Functionalities of Camunda

Figure 82, illustrates the platform's architecture, which is lightweight and modular and facilitates integration with a range of enterprise systems. Key tools, such as the Modeler for process design, Tasklist for user interaction, and Cockpit for monitoring, work in tandem with the core engine, subdivided into (BPMN, DMN and CMMN engines), to deliver seamless

automation. Camunda easily integrates with databases, third-party systems, and custom applications via REST and Java APIs, guaranteeing operational efficiency and transparency.

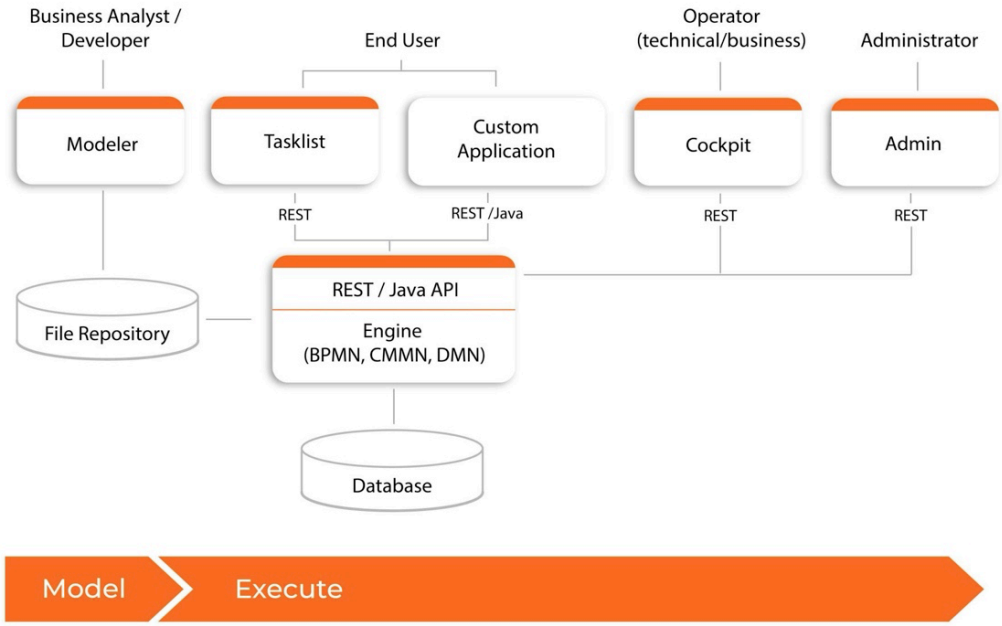


Figure 82 Camunda Core Components

Camunda's extensive set of features meets the various of modern enterprises by offering tools to automate processes, manage decisions, and handle dynamic workflows. Its core functionalities are driven, as shown in Figure 82, by three engines:

BPMN Workflow Engine

At the core of Camunda's architecture is its BPMN-based workflow engine, facilitating the design and execution of intricate business processes. BPMN, as noted by White and Miers , offers a standardized visual language that fosters collaboration between business and technical teams [35]. Camunda extends this standard by seamlessly integrating with both microservices and legacy systems, providing organizations with the flexibility to automate end-to-end workflows in diverse operational contexts.

The BPMN engine is closely tied to Tasklist interfaces, such as those provided by Camunda, offering a user-friendly bridge between process models and their execution. These interfaces empower business users to interact directly with workflows designed in tools like Camunda Modeler, enabling the launch, monitoring, and management of executable process instances with minimal technical overhead. By aligning BPMN automation with application-specific UIs, organizations can embed workflow control directly into existing enterprise applications, providing a more intuitive and efficient user experience. This approach not only enhances operational transparency but also increases adoption by ensuring workflows remain accessible within familiar business environments.

DMN Decision Engine

By encoding intricate business rules into decision tables, Camunda's DMN engine facilitates decision automation. Especially in regulated industries where error reduction and compliance are crucial, this feature guarantees consistency and transparency in decision-making processes. For instance, financial institutions can increase accuracy and efficiency by automating eligibility checks for loan approvals.

The Cockpit interface, which gives technical experts the ability to audit decision execution and make real-time adjustments, is used to manage the integration of DMN into backend systems. This feature guarantees that business rules continue to be in line with changing operational needs.

CMMN Case Management Engine

For workflows requiring human intervention or handling unpredictable scenarios, Camunda's CMMN engine offers the flexibility to adjust processes in real-time. Recker emphasizes the critical role of such adaptability in industries like healthcare, where workflows must accommodate evolving patient needs[96]. The case management tools from Camunda enable businesses to oversee changing procedures while preserving operational stability.

The Cockpit also supports case management by offering insights into task execution, enabling administrators to suspend or modify workflows as needed to address real-time issues.

Integration and Applications

Camunda's user-friendly web interface, which is divided into three separate modules to accommodate various user roles, improves usability. As the main interface for end users, the Tasklist provides a simple method of starting and managing workflows without requiring extensive technical knowledge. It guarantees that operational workflows are accessible and easy to use by streamlining communication with BPMN-based processes.

The Cockpit gives technical experts a thorough backend view of processes, allowing real-time monitoring and optimization. In order to keep workflows effective and flexible enough to meet changing operational requirements, this module is crucial for monitoring execution logs, evaluating process variables, and resolving incidents. The Admin Section also helps administrators by making it easier to manage users and permissions while adhering to organizational policies. Through this module, access can be controlled, roles configured, and security reinforced to ensure system integrity.

Beyond its interface, Camunda's modular architecture and adherence to industry standards, such as BPMN, DMN, and CMMN, make it an optimal solution for integration within complex enterprise ecosystems. Its compatibility with technologies like Apache Camel [80], and Kafka [97] enables seamless communication across heterogeneous systems, streamlining workflows and improving interoperability. Because of the platform's adaptability, businesses can centralize process management while still integrating it with current technologies.

Additionally, Camunda's adaptability and compatibility across diverse automation environments are ensured by its flexible architecture, which enables users to integrate it with a variety of BPMN engines. Because of its adaptability, businesses can use the execution engines of their choice while using Camunda's powerful modeling, decision-making, and monitoring features. Similarly, tools introduced by Corea and Delfmann demonstrate Camunda's ability to support consistent decision-making within workflows, aligning operational processes with strategic objectives [98]. This feature emphasizes how it serves as both a workflow engine and a strategic enabler.

Camunda combines user-centric design, strong integration potential, and support for data-driven decision-making to give organizations a comprehensive tool for navigating the complexities of contemporary enterprise operations. It is an essential tool for digital transformation projects since it not only automates procedures but also promotes operational transparency and strategic alignment.

Summary: Integrating Business Processes with Camunda

Camunda has established itself as a leader in business process automation, providing organizations with the tools to streamline operations, foster innovation, and adapt to the demands of digital transformation. It is an adaptable solution for companies navigating the complexities of modern workflows because of its open-source nature and compliance with industry standards like BPMN and DMN. Camunda empowers businesses to enhance operational transparency and decision-making capabilities by enabling seamless integration with heterogeneous systems and supporting process mining and predictive monitoring.

However, Camunda's adoption is not without difficulties. Organizations without technical expertise may be put off by the high learning curve involved in becoming proficient in BPMN and DMN modeling. Leopold, Mendling, and Gunther address the value of organized training and the development of user-friendly tools to mitigate these barriers [45]. Additionally, while the platform's open-source nature fosters innovation and adaptability, it also demands a level of customization and continuous upkeep that some organizations may not be able to provide.

Despite these challenges, Camunda keeps evolving as a pivotal enabler of operational excellence and strategic agility. Organizations can take full advantage of its potential by removing adoption barriers through tool improvements and training. In a business environment that is changing quickly, Camunda's distinctive blend of flexibility, strong integration skills, and support for data-driven frameworks guarantees its continued relevance.

Appendix D

Zeebe, a distributed workflow engine built for scalability and resilience, is at the heart of Camunda's architecture, which is shown in Figure 83. Zeebe operates as a replicated cluster that persists state via an event log and synchronizes consistency through the RAFT consensus algorithm. Client interactions with Zeebe are mediated by the Zeebe Gateway, which exposes APIs for deploying process models, starting instances, and interacting with running workflows. Developers typically use the Camunda Modeler to author BPMN, DMN, or form artifacts, which are then deployed via the gateway, while external workers connect through the Zeebe Client API to fetch and complete service tasks in accordance with the External Task pattern.

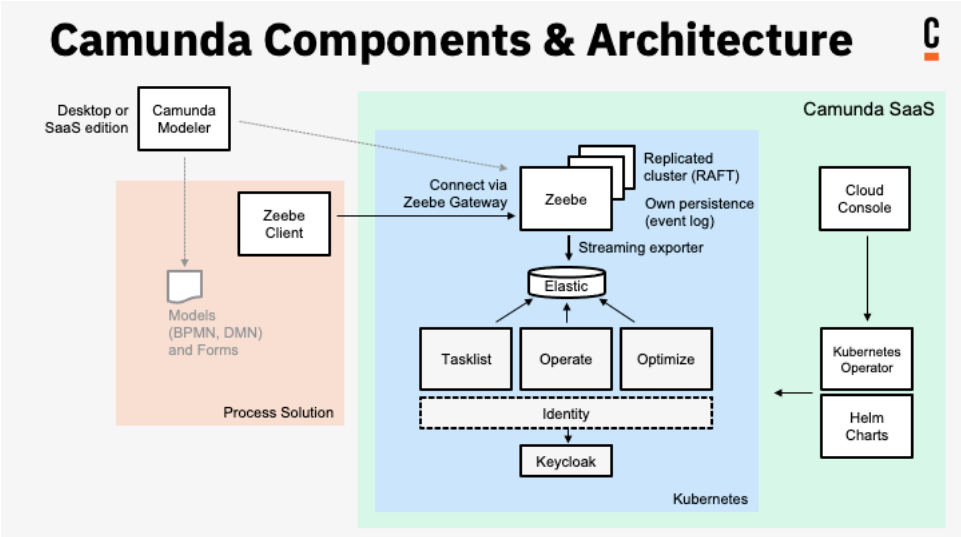


Figure 83 Camunda Architecture

Surrounding Zeebe is an operational toolchain that supports execution monitoring and continuous improvement. Operate enables operators to visualize process flows and resolve incidents, while Tasklist provides an interface for end-users to interact with human tasks. Optimize aggregates data for performance and compliance analytics. Elasticsearch is used by these components to export and index execution data for monitoring, querying, and inspection.

The Identity component, which controls authorization and authentication throughout the platform, is a crucial part of Camunda's contemporary architecture. In Camunda 8, this identity service is not monolithic or encapsulated, but designed to be coupled with external identity providers such as Keycloak. The designed framework, which incorporates Camunda into an existing Keycloak realm, relied heavily on this capability. The framework uses its central IAM to provide unified authentication, guaranteeing that authorization rules are uniformly applied across all containers, rather than keeping a second, separate identity service. This contrasts with earlier versions such as ,where identity management was tightly embedded and could not be decoupled from the engine's internal Keycloak instance. By integrating Camunda with the framework's existing Keycloak realm, we eliminate redundancy, simplify governance, and

ensure that role assignments are defined and enforced only once, upstream at the Core Gateway.

Appendix E

The Serverless Services container embodies the framework's function execution layer, and it is entirely built on top of Nuclio, a high-performance serverless platform. While the Integration Layer provides the logic to scaffold, package, and deploy Ballerina-based services, and the Core Backend triggers their execution, it is Nuclio that supplies the runtime environment that allows those services to run elastically, at scale, and on demand.

At its core, Nuclio's architecture is divided into three conceptual planes — control, runtime, and observability — each with distinct components and responsibilities as shown in Figure 84.

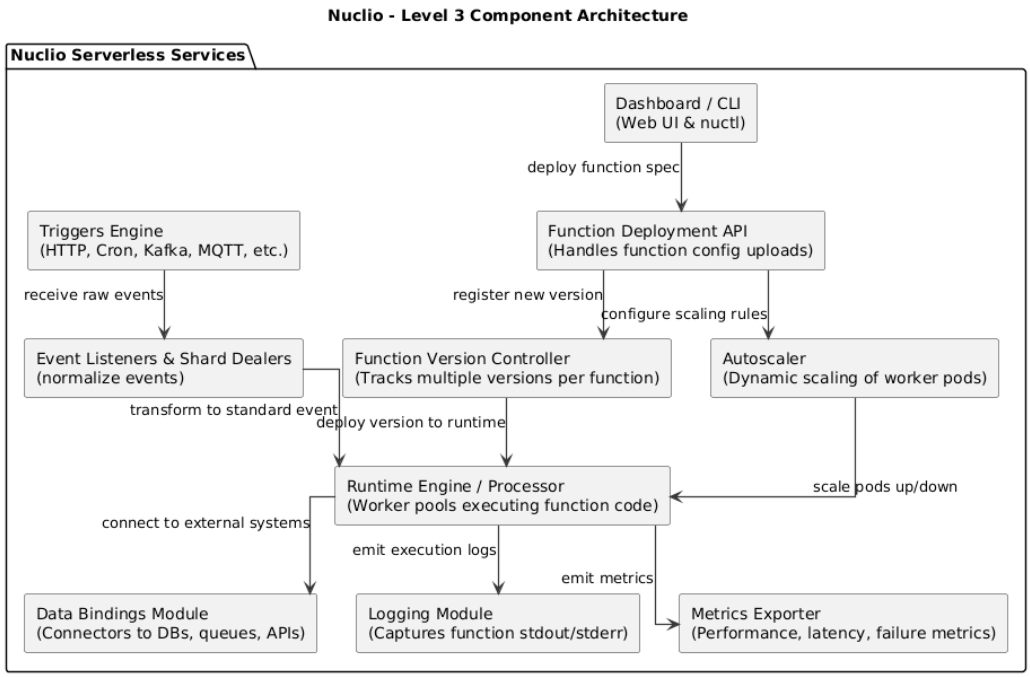


Figure 84 Serverless Service - Component View of Nuclio

The control plane governs the lifecycle of functions. The Nuclio Dashboard and CLI (through the nuctl utility) act as the external interfaces for function management. Every integration that is deployed by the Integration Layer ultimately reaches the Function Deployment API, which processes incoming function specifications, container images, and deployment configurations. Once received, the Deployment API registers the new function with the Function Version Controller, which tracks multiple versions of the same function, enabling upgrades, rollbacks, and safe iterative development. The control plane also includes the Autoscaler, which applies scaling policies by monitoring demand and automatically adjusting the number of function replicas in the runtime environment.

Beneath the control plane lies the runtime plane, which is where the actual execution of functions takes place. All invocation requests — whether triggered by HTTP calls from the Core Backend, Kafka messages, cron schedules, or other supported protocols — enter the system through the Triggers Engine. This engine supports a variety of trigger types, abstracting their

differences into a uniform entry point. Events are then normalized by Event Listeners and Shard Dealers, which distribute incoming events across available workers, ensuring parallelism and efficient load distribution. Once normalized, events are handed off to the Runtime Processor, which is essentially the worker pool responsible for executing the deployed functions. When a Ballerina-based integration service is invoked, it is the Runtime Processor that instantiates and executes that code, applying the function logic to the provided input. The Data Bindings module, which encapsulates connectors for a range of external systems and protocols, is used by the Runtime Processor to carry out tasks outside of its immediate scope, such as contacting a database, using a message queue, or contacting an external API.

Nuclio's observability plane, which supports both control and runtime planes, guarantees operational insight and transparency into each deployed function. Execution logs are gathered by the Logging module, which records the stdout and stderr streams of every function instance and provides crucial data for auditing and debugging. Performance metrics, including error rates, latency measurements, and invocation counts, are collected by the Metrics exporter and made available to the Monitoring Layer for visualization and alerting.

These components intentionally depend on one another and flow together. The Dashboard and Deployment API register a new function that is deployed by the Integration Layer, the Version Controller monitors its status, and the Autoscaler gets ready to handle its scaling. When the Core Backend later triggers that function via an HTTP request, the Triggers Engine routes the call, the Event Listeners normalize it, and the Runtime Processor executes it. Throughout, the Logging and Metrics modules ensure that every step is visible and measurable.

By basing the Serverless Services container entirely on Nuclio, the framework inherits a proven, production-grade serverless architecture without re-engineering the fundamentals of function execution. Nuclio’s modular design aligns perfectly with the framework’s architectural goals: the control plane is cleanly separated from the runtime plane, allowing functions to be deployed and updated independently of how they are invoked, and observability is built-in rather than bolted on. Every Ballerina-based integration is now an on-demand Nuclio function thanks to this design, and the framework can now execute those integrations with millisecond latency, automatic scaling, and the adaptability to support a variety of triggers and external connectors.

Comparison between Serverless Platforms: Fission vs Nuclio

When selecting the execution backbone for a Kubernetes-native integration framework, one must weigh the trade-offs between simplicity, configurability, and operational maturity. Among the ecosystem of open-source serverless platforms, Fission and Nuclio stand out for their relatively lightweight design and strong alignment with Kubernetes. Yet their trajectories diverge sharply in how they balance developer convenience and production readiness, making the comparison between the two both necessary and consequential.

Fission was designed as a lean serverless layer for Kubernetes. Developers can quickly push small functions into a cluster thanks to its command-line interface, which abstracts away

deployment complexity. The platform is a great option for rapid prototyping because its pool manager enables functions to be written in multiple languages, triggered by common events like HTTP requests or message queues, and deployed with low cold-start latency. Fission, however, purposefully limits its reach. It requires additional configuration work to integrate with monitoring stacks, has low observability, and lacks a native user interface. More importantly, exposing functions securely to external consumers often demands manual setup through ingress controllers or service meshes. When functions need to be scaled, debugged, or audited in production settings, this lean design causes friction, even though it lowers the entry barrier in academic experiments.

Nuclio, on the other hand, markets itself as a production-quality serverless platform. Its architecture is far more comprehensive, combining a comprehensive management layer with Kubernetes-native deployment. The Nuclio dashboard, which enables developers to deploy functions, set up triggers, view live logs, and monitor metrics in real time—all from a single interface—is one of the main differentiators. Developers and operators can handle scaling rules, runtime parameters, and security constraints without using manual YAML editing thanks to this visibility, which significantly lessens their cognitive load.

Another characteristic that distinguishes Nuclio is the depth of its trigger ecosystem. Nuclio supports a large number of event sources, such as Kafka, NATS, MQTT, cron jobs, V3IO streams, and cloud-native storage events, while Fission is restricted to HTTP, timers, and simple messaging. In integration scenarios, where various enterprise systems must communicate via heterogeneous channels, this breadth is especially useful. This eliminates the need for intermediate adapters by enabling Nuclio functions to be directly bound to event streams or message brokers.

These findings are supported by a comparative performance analysis conductedby Manner et al. . The study revealed that Nuclio consistently achieved higher throughput and lower latency under load compared to Fission, while also demonstrating stronger resilience in concurrent execution scenarios. The authors came to the conclusion that Nuclio's performance optimizations and operational tooling make it a more compelling choice for production-grade workloads, whereas Fission's strength lies in experimentation and prototyping. These empirical findings directly informed the architectural decision: the integration framework required a platform that could not only run lightweight functions but also sustain them under enterprise-grade traffic and offer full observability.

Figure 85 llustrates the internal architecture of Nuclio function pods. Requests from clients are first routed through a third-party ingress controller, then passed into the Nuclio event listener, which dispatches them to the appropriate worker processes. Each worker executes the user-defined function logic while the listener ensures efficient load distribution and scaling. This architectural separation between ingress, event listeners, and workers underscores Nuclio's design principle of maintaining low latency while allowing horizontal scalability.

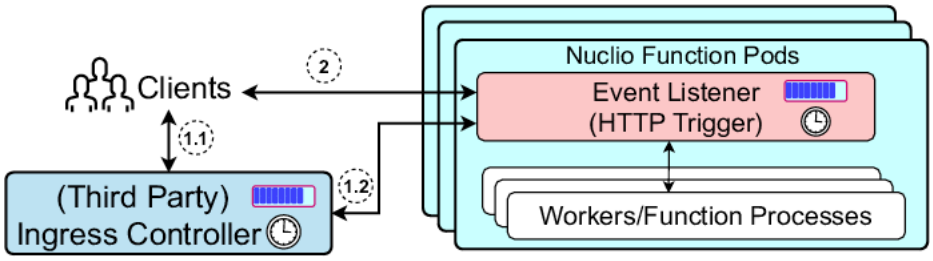


Figure 85 Fission vs Nuclio - Component View of Nuclio Pod Architecture

To complement this architectural view, Figure 86 depicts the dynamic flow of a request as a sequence of interactions.

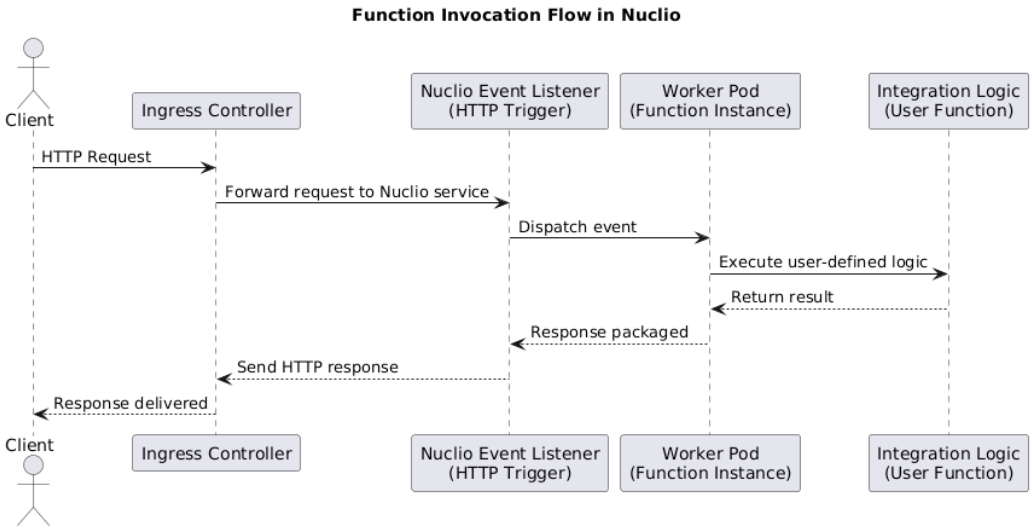


Figure 86 Fission vs Nuclio - Sequence View of Request Traversal

The diagram shows how a client request traverses the ingress controller, reaches the Nuclio event listener, and is dispatched to worker pods for execution. The user-defined integration logic is executed within the worker, and the result is propagated back to the client through the same pathway.

Together, the figures above provide both a structural and a behavioral understanding of Nuclio's execution model. Where the former highlights the modularity of Nuclio's architecture, the latter captures its runtime efficiency. These perspectives demonstrate why Nuclio was selected as the execution backbone of the integration framework: it enables not only scalable and manageable function orchestration but also the observability and configurability that Fission does not yet fully provide.

Appendix F

This appendix shows all the pages of the exported document based on the Microsoft Project of this academic paper.

ID	Task Mode	WBS	Task Name	WBS Level	Duration	Start	Finish	Septem 02
1		**1**	**Dissertation Project**	**Project**	**230 days**	**Mon 16-09-24**	**Fri 01-08-25**	
2		**1.1**	**Dissertation Preparation**	**Phase**	**104 days**	**Mon 16-09-24**	**Thu 06-02-25**	
3		**1.1.1**	**State-of-the-Art**	**Deliverable**	**80 days**	**Mon 16-09-24**	**Fri 03-01-25**	
4		1.1.1.1	Background and Standards	Task	80 days	Mon 16-09-24	Fri 03-01-25	
5		1.1.1.2	Technologies Identification	Task	45 days	Mon 04-11-24	Fri 03-01-25	
6		**1.1.2**	**Research Phase**	**Phase**	**40 days**	**Mon 11-11-24**	**Fri 03-01-25**	
7		1.1.2.1	Research Questions	Task	40 days	Mon 11-11-24	Fri 03-01-25	
8		1.1.2.2	Discussion	Task	20 days	Mon 09-12-24	Fri 03-01-25	
9		**1.1.3**	**Planning Phase**	**Phase**	**45 days**	**Mon 04-11-24**	**Fri 03-01-25**	
10		1.1.3.1	Project Charter	Deliverable	45 days	Mon 04-11-24	Fri 03-01-25	
11		1.1.3.2	Work Breakdown Structure	Deliverable	45 days	Mon 04-11-24	Fri 03-01-25	
12		1.1.3.3	Detailed Project Plan	Deliverable	45 days	Mon 04-11-24	Fri 03-01-25	
13		1.1.3.4	Skills Management	Deliverable	45 days	Mon 04-11-24	Fri 03-01-25	
14		1.1.4	Dissertation Preparation Report	Deliverable	30 days	Mon 04-11-24	Fri 13-12-24	
15		1.1.5	Dissertation Preparation Report Validation	Task	6 days	Mon 16-12-24	Mon 23-12-24	
16		1.1.6	Final Dissertation Preparation Report	Deliverable	6 days	Tue 24-12-24	Tue 31-12-24	
17		1.1.7	Dissertation Preparation Presentation	Deliverable	3 days	Wed 01-01-25	Fri 03-01-25	
18		**1.2**	**Framework Proposal Phase**	**Phase**	**30 days**	**Fri 07-02-25**	**Thu 20-03-25**	
19		1.2.1	Framework Conception/Design	Task	10 days	Fri 07-02-25	Thu 20-02-25	
20		1.2.2	Define Selection Criteria and Evaluate Tools	Task	10 days	Fri 21-02-25	Thu 06-03-25	

Projeto: Dissertation Project Pla
Data: Tue 31-12-24

Baseline Milestone	Resumo Inativo	Prazo
Baseline Summary	Tarefa Manual	Linha Base
Tarefa	Apenas-duração	Marco da Linha Base
Dividir	Resumo da Agregação Manual	Resumo da Linha Base
Marco	Resumo Manual	Progresso
Sumário	Apenas início	Progresso Manual
Resumo de Projeto	Apenas-conclusão	Baseline
Tarefa Inativa	Tarefas Externas	
Marco Inativo	Marco Externo	

Página 1

ID	Task Mode	WBS	Task Name	WBS Level	Duration	Start	Finish	Sep '2 01 | 0
21		1.2.3	Establish Testing Strategy	Task	5 days	Fri 07-03-25	Thu 13-03-25	
22		1.2.4	Proposal for Construction Document	Deliverable	10 days	Fri 07-03-25	Thu 20-03-25	
23		1.2.5	BPMN Improvement	Task	25 days	Fri 07-02-25	Thu 13-03-25	
24		1.2.6	Enterprise Integration Patterns Improvement	Task	25 days	Fri 07-02-25	Thu 13-03-25	
25		1.2.7	Apache Kafka and Camel Integration Prototype	Task	25 days	Fri 07-02-25	Thu 13-03-25	
26		**1.3**	**Framework Construction Phase**	**Phase**	**35 days**	**Fri 21-03-25**	**Thu 08-05-25**	
27		1.3.1	Environments and Tools Setup	Task	35 days	Fri 21-03-25	Thu 08-05-25	
28		1.3.2	Implement Integration Modules	Task	20 days	Fri 21-03-25	Thu 17-04-25	
29		1.3.3	Conduct Initial Prototype Testing	Task	15 days	Fri 21-03-25	Thu 10-04-25	
30		1.3.4	Consolidate the Modular Framework	Task	15 days	Fri 21-03-25	Thu 10-04-25	
31		1.3.5	Modular Framework Core	Deliverable	21 days	Thu 10-04-25	Thu 08-05-25	
32		1.3.6	Camunda and GraphQL Improvement	Task	25 days	Fri 21-03-25	Thu 24-04-25	
33		1.3.7	GDPR compliance principles Improvement	Task	25 days	Fri 21-03-25	Thu 24-04-25	
34		**1.4**	**Evaluation Phase**	**Phase**	**40 days**	**Fri 09-05-25**	**Thu 03-07-25**	
35		1.4.1	Test Scenarios Specification	Task	10 days	Fri 09-05-25	Thu 22-05-25	
36		1.4.2	Perform Test Scenarios	Task	15 days	Fri 23-05-25	Thu 12-06-25	
37		1.4.3	Collect Analyzed Results	Task	10 days	Fri 13-06-25	Thu 26-06-25	
38		1.4.4	Evaluation Report	Deliverable	6 days	Thu 26-06-25	Thu 03-07-25	
39		**1.5**	**Conclusions Phase**	**Phase**	**13 days**	**Fri 04-07-25**	**Tue 22-07-25**	
40		1.5.1	Report Validation	Task	5 days	Fri 04-07-25	Thu 10-07-25	

Projeto: Dissertation Project Pla
Data: Tue 31-12-24

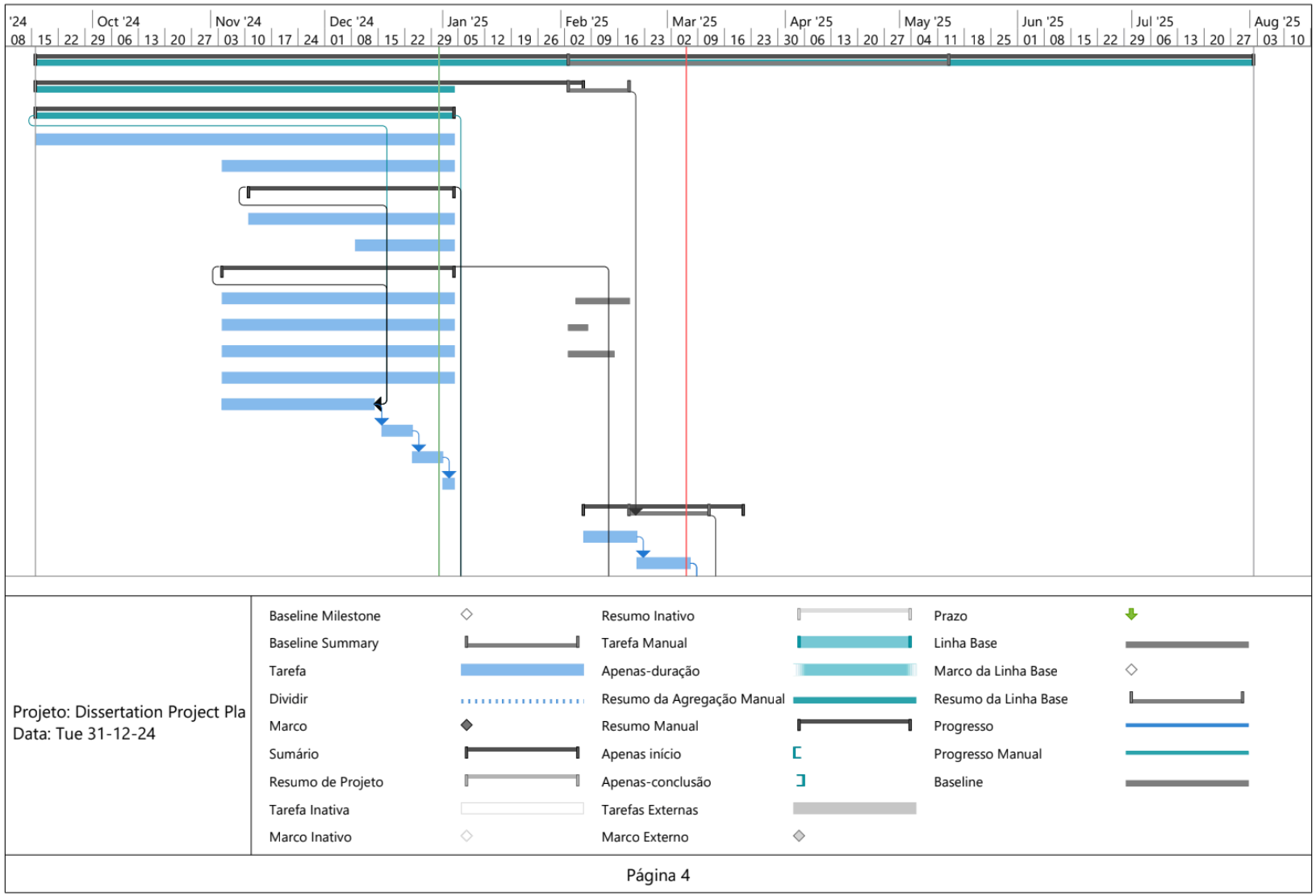
Baseline Milestone	Resumo Inativo	Prazo
Baseline Summary	Tarefa Manual	Linha Base
Tarefa	Apenas-duração	Marco da Linha Base
Dividir	Resumo da Agregação Manual	Resumo da Linha Base
Marco	Resumo Manual	Progresso
Sumário	Apenas início	Progresso Manual
Resumo de Projeto	Apenas-conclusão	Baseline
Tarefa Inativa	Tarefas Externas	
Marco Inativo	Marco Externo	

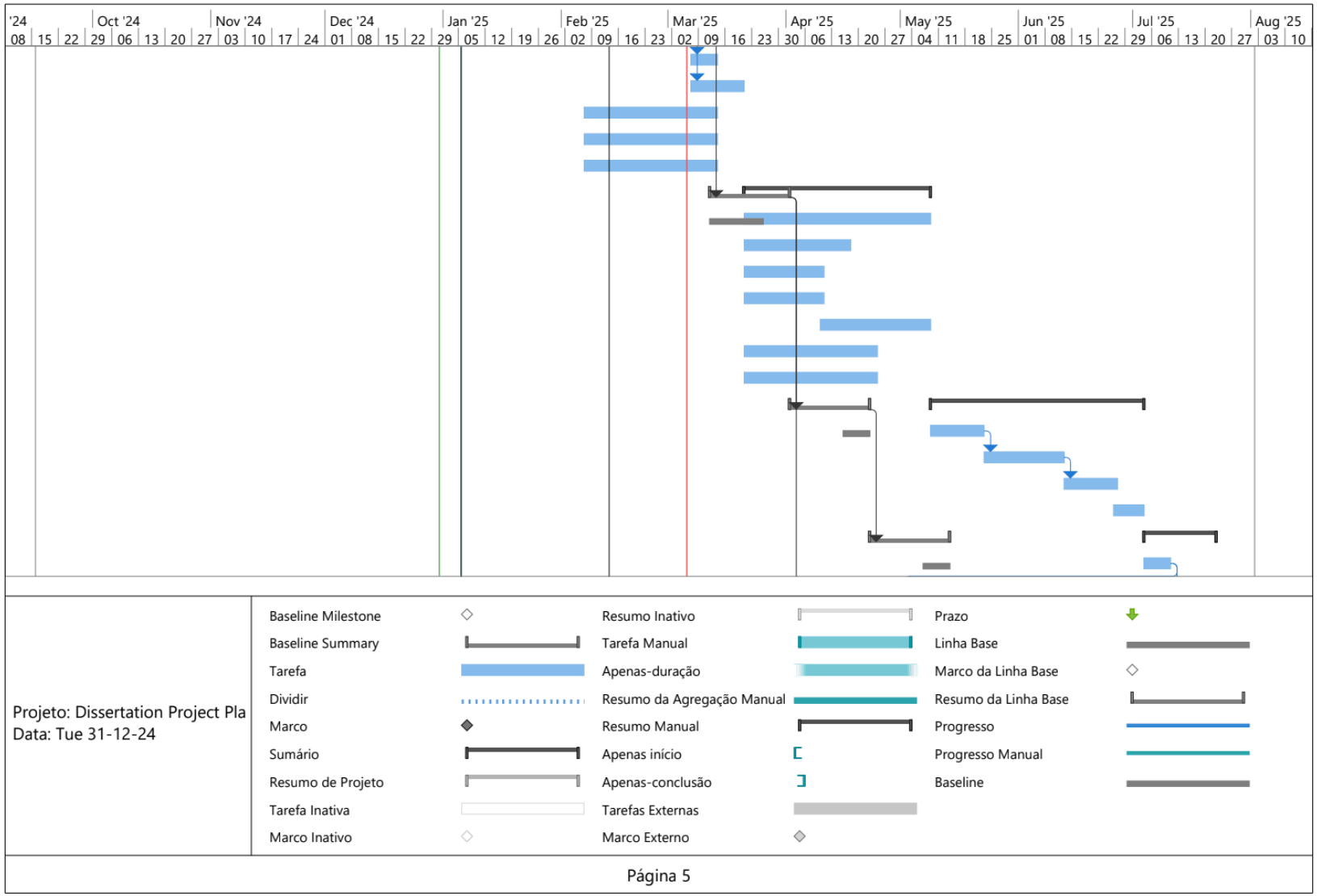
ID	Task Mode	WBS	Task Name	WBS Level	Duration	Start	Finish	Sep '2 01 0
41		1.5.2	Dissertation Report	Deliverable	5 days	Fri 11-07-25	Thu 17-07-25	
42		1.5.3	Final presentation	Deliverable	5 days	Fri 04-07-25	Thu 10-07-25	
43		1.5.4	Presentation Validation	Task	5 days	Fri 11-07-25	Thu 17-07-25	
44		1.5.5	Presentation Preparation	Task	3 days	Fri 18-07-25	Tue 22-07-25	
45		**1.6**	**Milestones**	**Milestones**	**151 days**	**Fri 03-01-25**	**Fri 01-08-25**	
46		1.6.1	State-of-art delivered	Milestones	0 days	Fri 03-01-25	Fri 03-01-25	
47		1.6.2	Planning delivered	Milestones	0 days	Fri 03-01-25	Fri 03-01-25	
48		1.6.3	Competencies Management delivered	Milestones	0 days	Fri 03-01-25	Fri 03-01-25	
49		1.6.4	Framework finished	Milestones	0 days	Thu 08-05-25	Thu 08-05-25	
50		1.6.5	Dissertation Report delivered	Milestones	0 days	Thu 19-06-25	Thu 19-06-25	
51		1.6.6	Inte. Analysis finished	Milestones	0 days	Thu 19-06-25	Thu 19-06-25	
52		1.6.7	Final report delivered	Milestones	0 days	Fri 25-07-25	Fri 25-07-25	
53		1.6.8	Presentation date	Milestones	0 days	Fri 01-08-25	Fri 01-08-25	

Projeto: Dissertation Project Pla
Data: Tue 31-12-24

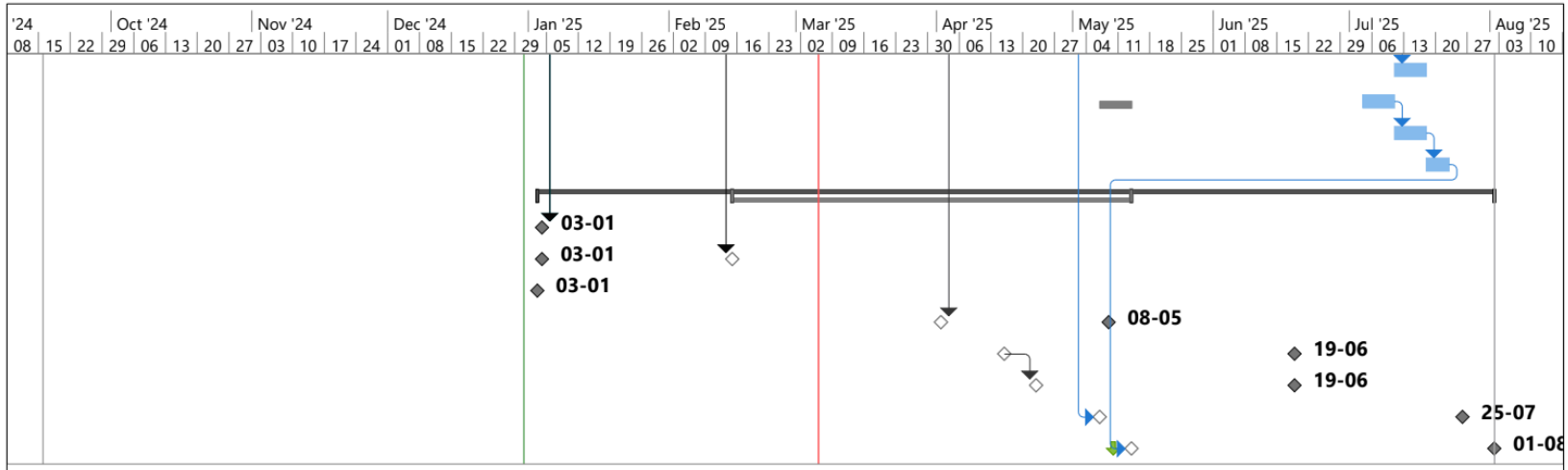
Baseline Milestone	Resumo Inativo	Prazo
Baseline Summary	Tarefa Manual	Linha Base
Tarefa	Apenas-duração	Marco da Linha Base
Dividir	Resumo da Agregação Manual	Resumo da Linha Base
Marco	Resumo Manual	Progresso
Sumário	Apenas início	Progresso Manual
Resumo de Projeto	Apenas-conclusão	Baseline
Tarefa Inativa	Tarefas Externas	
Marco Inativo	Marco Externo	

Página 3


'24 Oct '24 Nov '24 Dec '24 Jan '25 Feb '25 Mar '25 Apr '25 May '25 Jun '25 Jul '25 Aug '25
08 15 22 29 06 13 20 27 03 10 17 24 01 08 15 22 29 05 12 19 26 02 09 16 23 02 09 16 23 30 06 13 20 27 04 11 18 25 01 08 15 22 29 06 13 20 27 03 10
Projeto: Dissertation Project Pla
Data: Tue 31-12-24
Baseline Milestone
Baseline Summary
Tarefa
Dividir
Marco
Sumário
Resumo de Projeto
Tarefa Inativa
Marco Inativo
Resumo Inativo
Tarefa Manual
Apenas-duração
Resumo da Agregação Manual
Resumo Manual
Apenas início
Apenas-conclusão
Tarefas Externas
Marco Externo
Prazo
Linha Base
Marco da Linha Base
Resumo da Linha Base
Progresso
Progresso Manual
Baseline
Página 4


'24
Oct '24
Nov '24
Dec '24
Jan '25
Feb '25
Mar '25
Apr '25
May '25
Jun '25
Jul '25
Aug '25
08 15 22 29 06 13 20 27 03 10 17 24 01 08 15 22 29 05 12 19 26 02 09 16 23 02 09 16 23 30 06 13 20 27 04 11 18 25 01 08 15 22 29 06 13 20 27 03 10
Projeto: Dissertation Project Pla
Data: Tue 31-12-24
Baseline Milestone
Baseline Summary
Tarefa
Dividir
Marco
Sumário
Resumo de Projeto
Tarefa Inativa
Marco Inativo
Resumo Inativo
Tarefa Manual
Apenas-duração
Resumo da Agregação Manual
Resumo Manual
Apenas início
Apenas-conclusão
Tarefas Externas
Marco Externo
Prazo
Linha Base
Marco da Linha Base
Resumo da Linha Base
Progresso
Progresso Manual
Baseline

Página 5



Projeto: Dissertation Project Pla
Data: Tue 31-12-24

Baseline Milestone	Resumo Inativo	Prazo
Baseline Summary	Tarefa Manual	Linha Base
Tarefa	Apenas-duração	Marco da Linha Base
Dividir	Resumo da Agregação Manual	Resumo da Linha Base
Marco	Resumo Manual	Progresso
Sumário	Apenas início	Progresso Manual
Resumo de Projeto	Apenas-conclusão	Baseline
Tarefa Inativa	Tarefas Externas	
Marco Inativo	Marco Externo	

Appendix G

This appendix lists the table of Risk Assessment.

Risk ID	Description	Cause	Effect	Risk Owner	Probability (1-5)	Impact (1-5)	PI Score	Expected Result, No Action	Risk Response Type	Response Description
1	Delayed Timeline	Unforeseen technical challenges or resource limitations	Delayed project milestones, increased stress, and insufficient time for analysis	Researcher	3	4	12	Missed deadlines, reduced time for deliverables	Mitigate	Develop a contingency plan and allocate buffer periods for delays.
2	Tool Incompatibility	Mismatched integration capabilities of specific tools	Delays or errors in individual tool operation	Researcher	3	4	12	Individual tools may fail to work as expected	Avoid	Perform early testing of tools and their capabilities against project requirements.
3	Data Scarcity	Limited access to real-world datasets	Reduced validity of experimental results	Researcher	3	5	15	Reduced quality of findings	Mitigate	Use high-quality simulated datasets where real-world data is unavailable.
4	Stakeholder Feedback Delays	Supervisor availability	Slower project progress	Researcher	2	4	8	Delayed decision-making	Share	Schedule regular, flexible feedback sessions.
5	Technology Obsolescence	Rapid changes in tool landscapes	Framework becomes less relevant	Researcher	2	4	8	Obsolete framework with reduced applicability	Avoid	Adopt modular and extensible framework design to accommodate future advancements.
6	Inadequate Data Collection	Issues with experimental tools	Unreliable results and difficulty comparing applications	Researcher	3	5	15	Weak findings undermine project credibility	Avoid	Pre-test tools and processes thoroughly before the experimental phase begins.
7	Supervisor Availability and Feedback	Inconsistent availability of feedback	Delays in project progress and	Researcher	2	4	8	Uncoordinated guidance and slower progress	Contingency	Plan for alternative sources of

			misaligned research direction							feedback, such as peer reviews.
8	Limited Resources	Constraints in accessing necessary software tools	Limited analysis due to insufficient tools	Researcher	3	3	9	Reduced analysis capability	Share	Seek collaborations or institutional support for accessing required tools.
9	Lack of Clear Conclusion	Insufficient analysis or weak data patterns	Weak final report with no actionable findings	Researcher	2	5	10	Poor feedback and lack of meaningful contribution	Accept	Acknowledge limitations but focus on presenting insights within available data.
10	Changes in Scope of the Dissertation	Unclear objectives during project	Increased workload and reduced quality	Researcher	3	3	9	Unclear direction and increased scope creep	Avoid	Define clear objectives and maintain strict adherence to the project's scope.
11	Integration Challenges	Architectural or workflow mismatches between tools and systems	System inefficiencies, errors, and delays in combining tools	Researcher	4	5	20	Project delays and increased debugging efforts	Mitigate	Ensure cohesive system design and conduct thorough compatibility testing during integration planning.