



**Master of Engineering in Information Technology
WS 2025-26**

**Agile Development in Cloud Computing
Environments**

Project Report:
*Service Management Tool (SMT) using
Camunda*

Submitted by:

<i>Manoj Hanumanthu</i>	<i>1566325</i>
<i>Aman Basha Patel</i>	<i>1565430</i>
<i>Muhammad Ahsan Ijaz</i>	<i>1566312</i>
<i>Saquib Attar</i>	<i>1567041</i>

Supervisor: Dr. Patrick Wacht

Date: 28.01.2026

Table of Contents

1	Introduction	2
2	Overview of Service Management Tool (SMT)	4
2.1	High-Level Architecture Diagram	4
2.2	Authentication & RBAC Component:	4
2.3	Process Orchestration Engine:	5
2.4	Integration Layer (REST APIs):	5
2.5	Provider Offer Management:	5
2.6	Reporting Interface:	5
2.7	General Requirements and Key Elements.....	5
3	Agile Framework in SMT Development.....	7
3.1	Team Structure & Stakeholder Engagement	7
3.2	Stakeholders:.....	10
3.3	Agile Practices	10
3.4	Daily Stand-up Meetings	10
3.5	Weekly Stand-up Meetings.....	11
3.6	Sprint Planning & User Stories.....	11
3.7	Critical Appraisal of Scrum Management	14
3.8	Kanban Board & User Stories	15
4	Development & Technology.....	17
4.1	Technical Stack	17
4.2	System Architecture.....	19
4.3	User Flow Graphs (BPMN Workflow).....	19
4.4	UI/UX Design (Thymeleaf Dashboards)	20
4.5	Database Design (PostgreSQL Schema).....	21
4.6	Documentation: Proof of Execution (Audit Log)	22
4.7	Proving Camunda Configuration (Cockpit).....	23
4.8	End-to-End Integration Walkthrough (Postman & UI)	23
4.9	Comparison of Technologies:	30
5	Conclusion	33
6	References.....	34

1 Introduction

In the rapidly evolving landscape of Information Technology, organizational agility is no longer just a competitive advantage; it is a necessity. To maintain this agility, modern enterprises increasingly rely on an external workforce freelancers, consultants, and service providers to bridge skill gaps and accelerate project delivery. However, the procurement and management of these external resources often remain trapped in legacy processes characterized by fragmented communication, manual data entry, and a lack of transparency. This project, the **Service Management Tool (SMT)**, was conceived to address these inefficiencies by digitizing and orchestrating the end-to-end procurement lifecycle for external IT services.

The primary motivation behind the development of the SMT is to eliminate the "siloe" nature of corporate procurement. In a traditional setting, a Project Manager might request a developer via email, the Legal department might check contracts in a separate file system, and Procurement Officers might receive offers via spreadsheets. This disjointed approach leads to data redundancy, compliance risks (such as employing staff without valid contracts), and significant delays in project timelines. The SMT solves this by acting as a central orchestration hub, enforcing a standardized, compliant, and efficient workflow that connects all stakeholders in real-time.

Technologically, the project acts as a middleware solution within a larger distributed system. It is built upon a robust **Spring Boot** architecture, leveraging **Camunda BPM (Business Process Management)** as an embedded workflow engine. This architectural choice allows the application to function as a state machine, ensuring that every service request persists through a defined lifecycle from Draft to Approval, Market Publication, Offer Selection, and Final Ordering. Unlike simple CRUD applications, the integration of Camunda ensures that long-running processes are handled reliably, maintaining the state of transactions even across server restarts or external system failures.

A defining feature of this project is its extensive interoperability with external systems, simulating a real-world enterprise Service-Oriented Architecture (SOA). The SMT does not exist in a vacuum; it serves as the integration point for multiple disparate groups. It consumes workforce requirements from **Workforce Management (Group 1b)** via REST APIs, validates legal eligibility by querying **Contract Management (Group 2b)**, creates market tenders for **External Providers (Group 4b)**, and finally, exposes transparent data streams for **Reporting and Analytics (Group 5b)**. This interconnectedness ensures that data flows seamlessly across departmental boundaries without manual intervention.

Furthermore, the system implements a strict governance model through Role-Based Access Control (RBAC). It distinguishes between three key internal actors: the **Project Manager (PM)**, who defines the technical needs; the **Procurement Officer (PO)**, who ensures budgetary and contractual compliance; and the **Resource Planner (RP)**, who finalizes the logistics of the hiring. By digitizing these roles and enforcing approval gates within the BPMN process diagram, the SMT ensures that no service order is generated without the necessary checks and balances.

In conclusion, the Service Management Tool represents a shift from reactive administration to proactive process orchestration. By combining the flexibility of Java Spring Boot with the process governance of Camunda, this project demonstrates how complex

business requirements can be translated into a scalable, user-friendly, and highly integrated software solution. It provides a blueprint for how organizations can modernize their procurement operations, ensuring that the right talent is acquired at the right time, under the right contract.

2 Overview of Service Management Tool (SMT)

This chapter provides a comprehensive overview of the High-Level Design (HLD) of the Service Management Tool (SMT), detailing the architectural decisions and core business requirements that shaped the development process. The primary objective was to translate the complex problem of disjointed procurement processes into a cohesive, automated workflow that delivers value within the specified academic timeline.

2.1 High-Level Architecture Diagram

The Service Management Tool (SMT) project, undertaken by Team 3b, aims to establish a centralized digital hub for orchestrating the entire lifecycle of external IT service procurement. Unlike standalone applications, SMT functions as an integration middleware, sitting at the nexus of four distinct organizational units. Figure 1 illustrates the comprehensive architecture of the project, highlighting the central role of SMT in connecting **Workforce Management (Group 1b)**, **Contract Management (Group 2b)**, **External Providers (Group 4b)** and **Reporting**

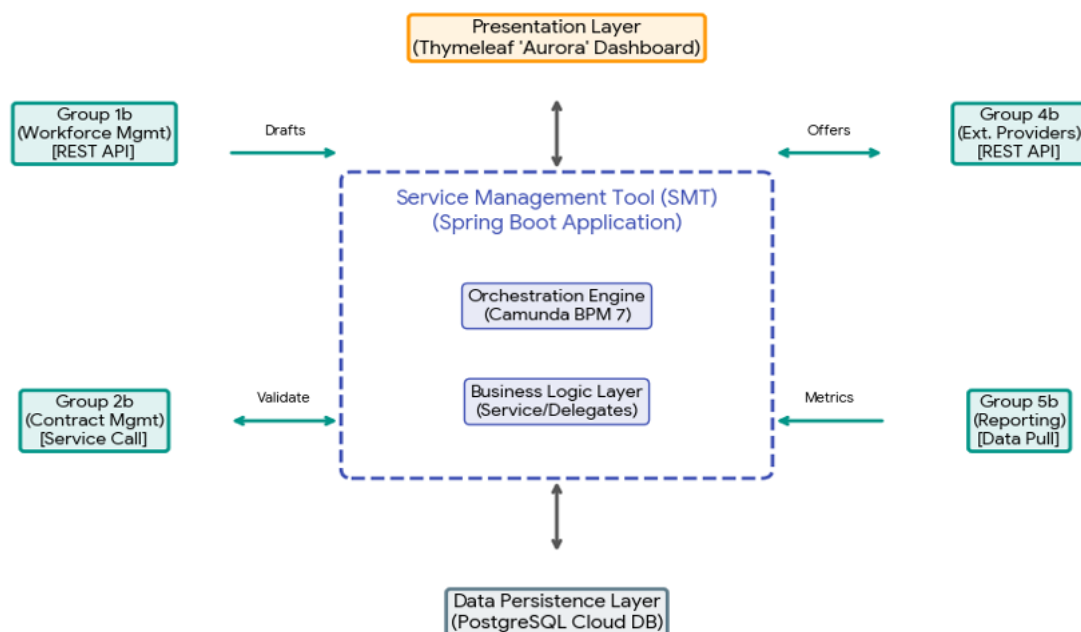


Figure 1: High-Level Architecture of Service Management Tool (SMT)

The architecture comprises five main functional components, each designed to handle specific aspects of the business logic.

2.2 Authentication & RBAC Component:

Built on Spring Security, this component manages the registration and authorization of internal actors. It enforces a strict Role-Based Access Control (RBAC) model, distinguishing between the Project Manager (who initiates requests), the Procurement Officer (who approves budgets), and the Resource Planner (who finalizes logistics).

2.3 Process Orchestration Engine:

At the heart of the system lies the embedded Camunda BPM engine. This component acts as a state machine, managing the long-running lifecycle of service requests. It persists the state of every transaction from "Draft" to "Waiting Approval" to "Completed" ensuring that no request is lost due to system restarts or delays in external responses.

2.4 Integration Layer (REST APIs):

SMT exposes a set of secure REST endpoints (ExternalIntegrationController) that allow Group 1b to push workforce requirements and Group 4b to submit candidate offers. Simultaneously, it consumes external APIs to validate contracts with Group 2b, ensuring seamless data interoperability.

2.5 Provider Offer Management:

This module handles the complex logic of aggregating, scoring, and ranking incoming offers from external providers. It calculates commercial and technical scores to assist the Project Manager in making data-driven selection decisions.

2.6 Reporting Interface:

To support data transparency, the system exposes a specialized read-only API for Group 5b. This component transforms raw transactional data into clean, flattened Data Transfer Objects (DTOs), enabling the generation of high-level KPIs without compromising the integrity of the core database.

2.7 General Requirements and Key Elements

During the initial requirements gathering phase, the development team identified a set of core functional requirements that acted as the guiding “**North Star**” for sprint planning and implementation. To ensure a structured and traceable development approach, these requirements were clearly categorized. **Orchestrated Workflow Management** focused on designing and implementing a BPMN 2.0compliant process that enforces a standardized approval sequence involving the Project Manager, Procurement Officer and Resource Planner. This guarantees process consistency and prevents unauthorized hiring actions.

Automated Contract Validation addressed a critical compliance requirement by integrating synchronously with the Contract Management System (Group 2b). Before any request could be published to the market, the system verifies the existence of a valid frame agreement. **Dynamic Offer Evaluation** required the development of a mechanism to collect and persist multiple provider offers and present them side by side in the user interface. This enables transparent comparison and scoring based on factors such as cost and required skills. **Decoupled Reporting Architecture** introduced a pull-based reporting model in which analytical data is exposed through REST APIs under `/api/reporting`. This allows the reporting team (Group 5b) to access real-time information without direct coupling to the operational database.

Role-Based Data Isolation ensured that tasks and actions are visible only to the appropriate user roles. For example, the validation of provider selection is restricted exclusively

to the Procurement Officer, enforcing security and responsibility boundaries. Finally, **Cloud-Native Deployment** required the application to be hosted on a containerized cloud platform using Render. This demonstrates the system's readiness for production use as a microservice and enables public accessibility for all integration partners. Moreover, implement robust error handling strategies (e.g., "NullPointerException" management and transaction rollbacks) .

3 Agile Framework in SMT Development

This chapter elaborates on the agile principles adopted by Team 3b and the standard practices implemented to ensure the seamless development of the Service Management Tool (SMT). It details the integration of agile methodologies into our workflow, which was instrumental in achieving operational efficiency, adaptability to evolving requirements, and the successful delivery of the "Aurora" dashboard and orchestration engine.

3.1 Team Structure & Stakeholder Engagement

Figure 2 illustrates agile software development life cycle operates as a structured continuum, guiding a product from its initial concept to its final retirement. For the SMT project, the principles of cohesive teamwork and proactive stakeholder engagement formed the foundation of our success. Teamwork in our context transcended simple collaboration; it represented a unified unit where member skills ranging from Java backend logic to frontend UI design were complementary. Our agile approach thrived on transparent communication, particularly when resolving the complex data mapping issues between our internal database IDs and the external identifiers used by our partners in Groups 1b and 4b.

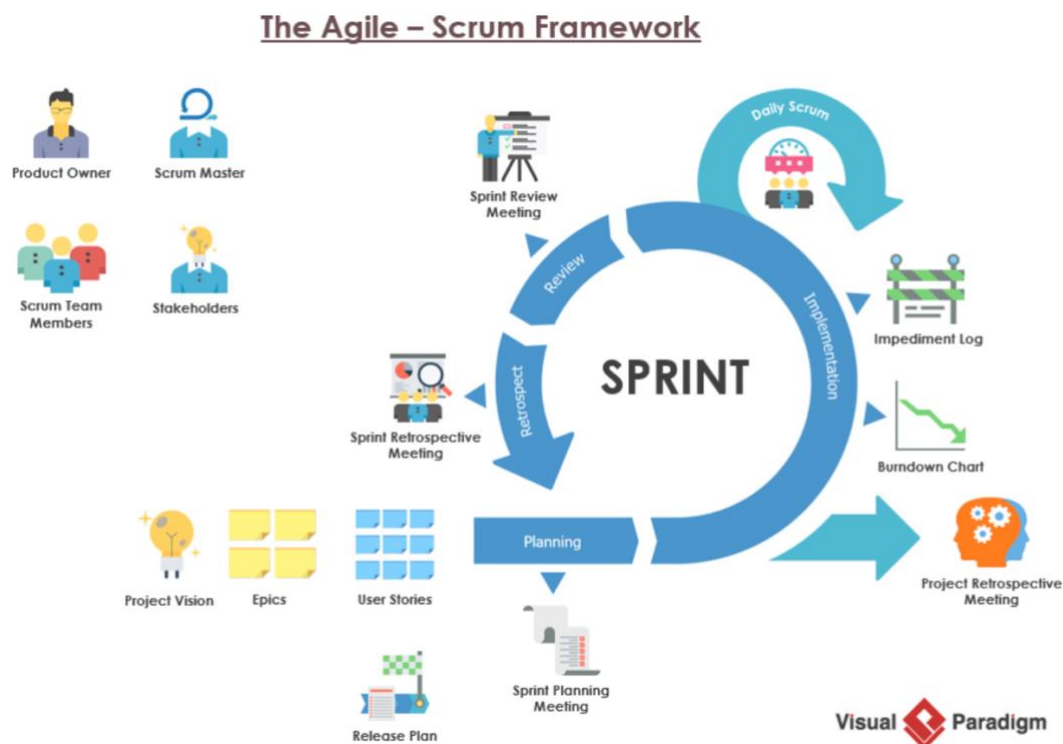


Figure 2: Agile Scrum Framework [4].

Simultaneously, stakeholder engagement was critical. Since the SMT acts as the central orchestration hub for the entire "External Workforce" domain, our stakeholders were not just observers but active participants. We established continuous feedback loops with the other student groups (our integration partners) to validate API contracts (Swagger) and refine JSON payloads as illustrated in Figure 3. This dynamic interaction allowed us to adapt our

"CheckContractDelegate" logic instantly when Group 2b modified their validation parameters. Ultimately, these principles reinforce the core agile philosophy: valuing individuals and interactions over rigid processes.

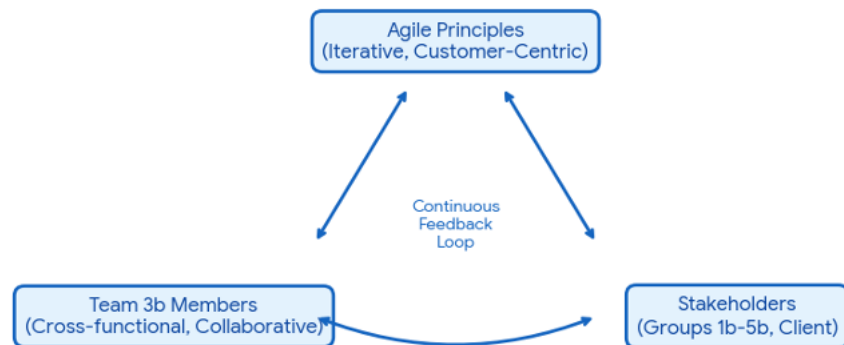


Figure 3: Agile Interaction Model for SMT Project.

In the specific context of Team 3b, we mapped the essential technical and managerial skills of each member to distinct roles to maximize efficiency. Figure 4 shows the team structure of the SMT project, directly associating each role with the tangible outcomes produced during our three sprints. Detailed descriptions of these roles and their responsibilities are provided below.

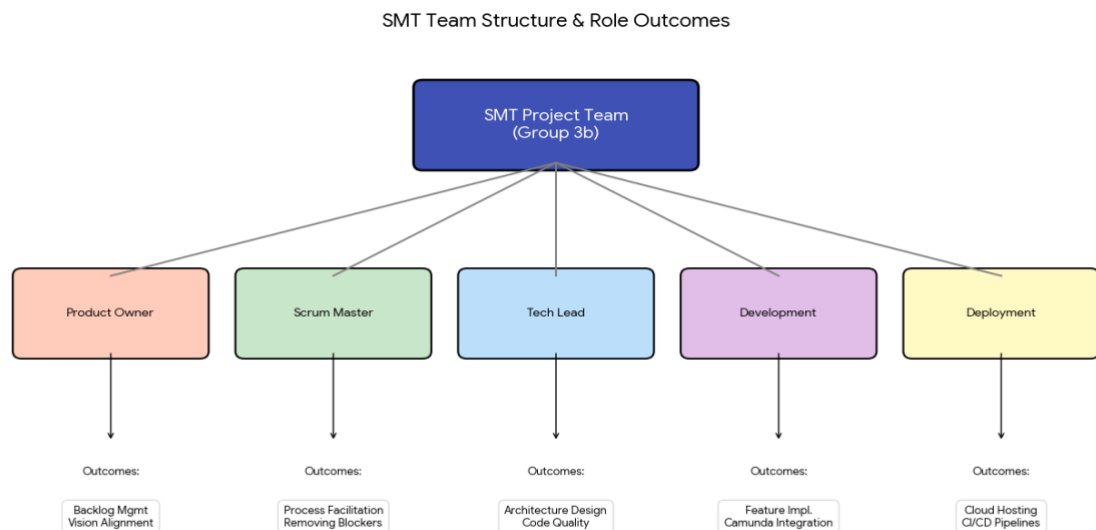


Figure 4: Team Roles And Outcome.

3.1.1 Product Owner:

The Product Owner served as the visionary for the project, responsible for defining the "Happy Path" and "Negative Scenarios" of the procurement process. In the context of the SMT, the Product Owner represented the business interests of the Procurement Department, ensuring the final product delivered real value such as preventing the hiring of freelancers without valid contracts. Their responsibilities included clarifying requirements for the "Aurora" dashboard, prioritizing the backlog (e.g., deciding that "Draft Creation" must precede "Reporting APIs"), and defining the acceptance criteria for each User Story. Additionally, the Product Owner managed the high-level BPMN process design, ensuring the flow from "Approval" to "Order Generation" was logically sound before implementation began.

3.1.2 Scrum Master:

The Scrum Master played a facilitative role, ensuring the team adhered to the Scrum framework while meeting the strict deadlines of our 3-sprint schedule. For Team 3b, the Scrum Master was responsible for facilitating Daily Stand-ups and Retrospectives, fostering a collaborative environment where developers felt safe to raise blockers such as the "NullPointerException" issues we faced with Camunda tasks. They acted as a mediator during integration disputes with external groups and ensured that our Jira board reflected the real-time status of development. A secondary but crucial responsibility was managing the project documentation structure, ensuring our technical achievements were accurately recorded.

3.1.3 Tech Lead:

The Tech Lead provided the architectural backbone for the project. They were responsible for the critical decision to use an Embedded Camunda Engine within a Spring Boot application, ensuring transactional consistency between our business data and process state. The Tech Lead guided the team on database schema design (PostgreSQL), ensuring the ServiceRequest and Provider Offer tables were correctly related. They also established the security protocols (Spring Security) that defined our RBAC model (PM, PO, RP roles) and led the proof-of-concept phase for our REST API integration strategy.

3.1.4 Development and Research Team:

At the forefront of execution, this function was responsible for translating requirements into working code. This involved implementing the Java Delegates (e.g., CheckContractDelegate, NotifyProviderDelegate) that powered our BPMN service tasks. They built the "Aurora" frontend using Thymeleaf and Bootstrap, creating the glass morphism UI that distinguishes our tool. Research was a key component here; the team had to actively research how to handle Camunda's optimistic locking and how to properly map complex nested JSON objects from Group 4b into our internal entity model.

3.1.5 Deployment and Testing (DevOps):

This function was pivotal in moving the SMT from "localhost" to the cloud. Responsibilities included containerizing the Spring Boot application, configuring the PostgreSQL database on Render, and managing environment variables for secure cloud deployment. They orchestrated the end-to-end testing strategy, creating the comprehensive Postman collections used to validate the system's behaviour against Groups 1b, 2b, and 4b. This role ensured that our final demo was not just a local simulation but a live, cloud-hosted reality.

3.2 Stakeholders:

3.2.1 Internal Stakeholders (Integration Partners):

Our "internal" stakeholders were Groups 1b (Workforce), 2b (Contracts), 4b (Providers), and 5b (Reporting). We established direct communication channels (API Contracts) to ensure our systems could "speak" to each other. For example, we worked closely with Group 5b to design a "Pull-based" reporting API that gave them read-only access to our raw data without compromising integrity.

3.2.2 External Stakeholders (The Client):

Represented by the Professor, the external stakeholder defined the core business problem the need for a digitized procurement workflow. Their feedback on requirements, such as the necessity for a "Validation Step" before ordering, directly influenced our BPMN design. This collaboration ensured that our technical solution solved the actual business problem presented at the start of the semester.

3.3 Agile Practices

In the development of the Service Management Tool (SMT), the adoption of Agile practices was not merely a procedural choice but a strategic necessity. Given the project's complex architecture involving an embedded workflow engine, a microservices-style integration pattern, and a cloud-native deployment target a rigid Waterfall approach would have failed to address the dynamic nature of our external dependencies. Instead, Team 3b implemented a hybrid Agile framework, predominantly leveraging Scrum for structured iterative delivery while incorporating elements of Kanban for managing the flow of tasks, particularly during the integration phases with Groups 1b, 2b, and 4b.

This section details the specific methodologies employed, including our disciplined approach to stand-up meetings, the rigorous planning behind our three development sprints, and the introspective value of our retrospectives. These practices fostered a culture of transparency, rapid feedback, and continuous improvement, ensuring that technical debt was managed proactively rather than reactively.

3.4 Daily Stand-up Meetings

3.4.1 Schedule:

Daily stand-ups were held every morning at 10:00 AM via video conference, strictly time-boxed to 15 minutes.

3.4.2 Purpose:

These meetings served as a tactical synchronization point. The focus was exclusively on the "Now" what was achieved yesterday, what is planned for today, and what impediments are blocking progress. For instance, during Sprint 2, the daily stand-up was crucial for resolving the "NullPointerException" errors we encountered with Camunda task completion. By raising this issue immediately in the morning, the Tech Lead could pair-program with the Developer to implement a robust complete Task method before the day's development continued.

3.4.3 Team Participation:

Every member, from the Product Owner to the DevOps engineer, was required to speak. This ensured that no siloed knowledge developed. For example, when the Deployment lead updated the Docker file for Render, the Backend team was immediately informed to check their

environment variable configurations (DB_URL, MAIL_PASSWORD) to prevent deployment failures.

3.4.4 Scrum Master's Role:

The Scrum Master acted as the timekeeper and blocker-remover. If a technical discussion about JSON parsing drifted into a deep-dive solutioning session, the Scrum Master would "park" the topic for a post-stand-up breakout session, ensuring the main meeting remained efficient.

3.5 Weekly Stand-up Meetings

3.5.1 Schedule:

Held every Friday afternoon.

3.5.2 Purpose:

Unlike the tactical daily stand-ups, the weekly meeting was strategic. It focused on the broader trajectory of the project, reviewing the week's burndown chart and preparing for the upcoming week's goals. This was also the forum for "Show and Tell" demos, where we would demonstrate working features (e.g., the first successful end-to-end process execution via Postman) to the internal team.

3.5.3 Team Participation:

The Product Owner played a central role here, verifying that the week's technical output matched the business requirements. For example, confirming that the "Draft" status correctly transitioned to "Waiting Approval" before the technical team moved on to the next feature.

3.5.4 Scrum Master's Role:

The Scrum Master used this time to assess the team's velocity and adjust the backlog for the next week if necessary. If the API integration with Group 4b took longer than expected, the Scrum Master would re-negotiate the scope for the next week to prevent burnout.

3.6 Sprint Planning & User Stories

The core of our development lifecycle was structured around three intensive Sprints. Sprints and User Stories helped us break down the monolithic requirement of "Build a Service Management Tool" into manageable, testable, and deliverable units of value. This section explores how these iterative cycles allowed us to build the system layer by layer from the database schema up to the "Aurora" dashboard.

Sprint Planning served as the formal kick-off for each sprint and acted as a collaborative ceremony in which the team aligned on what could be delivered during the upcoming sprint and how that work would be accomplished. As part of the sprint planning cycle, this ceremony marked the transition from backlog refinement to execution, ensuring that priorities, scope, and responsibilities were clearly defined before development began (see Figure 5).

This process followed several well-defined stages. First, a backlog review was conducted in which the Product Owner presented the prioritized backlog. For Team 3b, this involved recognizing that core backend logic needed to be completed in Sprint 1 before shifting focus to user interface development in Sprint 3. This was followed by capacity planning, during which the team realistically assessed its available bandwidth while considering other academic responsibilities, helping to prevent over-commitment. Finally, a clear sprint goal was defined

to provide focus and direction, such as the Sprint 2 objective of achieving a fully functional Camunda process that could run from start to end without errors.

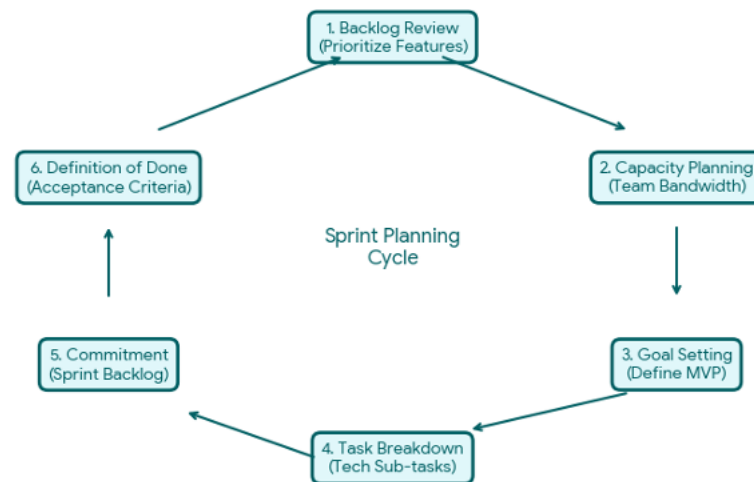


Figure 5: Sprint Planning Stages

High-level user stories were then broken down into concrete technical tasks, for example translating an approval-related user story into specific Java delegate implementations and BPMN model definitions. Finally, a strict Definition of Done was established to ensure quality and consistency; a task was considered complete only if the code compiled successfully, unit tests passed, and the corresponding API endpoint responded correctly when tested in Postman. Figure 6 shows structured and disciplined approach, the team successfully executed all three sprints, with the overall roadmap and task allocation clearly outlining and supporting the project's development journey.

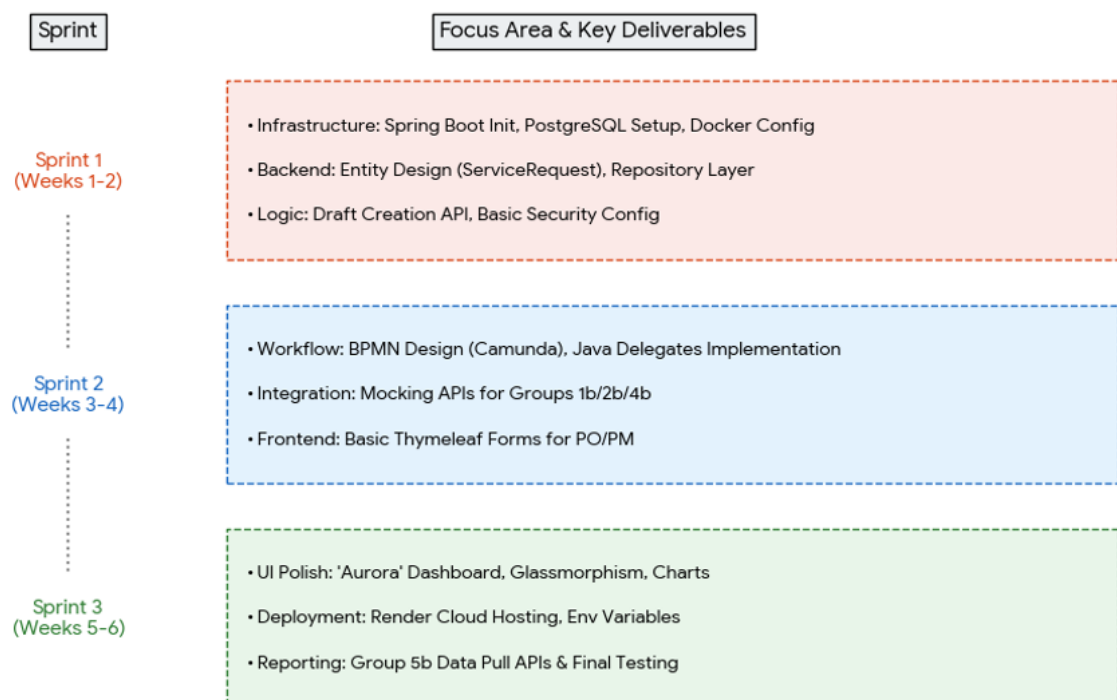


Figure 6: SMT Project Sprint Roadmap.

3.6.1 Sprint 1 (Foundation & Architecture):

The primary priority of this phase was to establish a solid foundation by setting up the project infrastructure, designing the database schema, and defining the core business entities. This work was driven by key user stories, including the need to initialize a Spring Boot project with Camunda and PostgreSQL dependencies to enable development, and to define the ServiceRequest and ProviderOffer entities to persist essential business data. To achieve this, several technical activities were carried out.

From an infrastructure perspective, the GitHub repository was set up and the pom.xml was configured to include the camunda-bpm-spring-boot-starter, ensuring seamless integration with the workflow engine. PostgreSQL was selected as the database system to provide production-grade reliability and robustness. On the backend, the core application logic was implemented by developing the Service Layer, particularly the ServiceRequestService, along with the corresponding Repository Layer. As part of this effort, an initial “Draft” creation API (POST /service-requests) was introduced, enabling external groups, such as Group 1b, to theoretically submit data to the system. In addition, basic Spring Security was implemented to enforce role-based access control, differentiating between pm_user, po_user, and rp_user, thereby laying the groundwork for secure and structured system interactions.

3.6.2 Sprint 2 (The Workflow Engine):

The main priority of this phase was to implement BPMN modeling, develop Java delegates, and realize the external integration logic required to orchestrate the service request lifecycle. This sprint was guided by key user stories focused on process automation and orchestration, such as enabling automatic contract validation with Group 2b to prevent time being spent on invalid requests and managing the state of a service request from “Draft” to “Completed” through a controlled state machine. To support these goals, several core activities were undertaken. First, the BPMN process was designed using the Camunda Modeler, where the service-request-process.bpmn was created by defining user tasks for project managers and procurement officers, alongside service tasks responsible for automated API interactions. This was followed by the implementation of Java delegates, which represented the most code-intensive aspect of the sprint.

These delegates included CheckContractDelegate, used to mock contract validation with Group 2b, PublishToProvidersDelegate, responsible for notifying Group 4b, and CreateServiceOrderDelegate, which finalized the transaction. Finally, integration testing was carried out using Postman to simulate interactions with external actors. Through these tests, it was verified that submitting a request with a projectId of 999 correctly triggered the “Contract Rejected” execution path in the BPMN workflow, demonstrating that the modeled process and delegate logic behaved as intended.

3.6.3 Sprint 3 (UI Polish & Cloud Deployment):

The final phase of the project focused on delivering the “Aurora” dashboard, implementing reporting APIs, and deploying the application to the cloud. This sprint was driven by user stories centered on usability and data accessibility, including the need for a modern, high-contrast dashboard that allows managers to quickly assess the status of all service requests, as well as a read-only reporting API to enable Group 5b to consume request data for business intelligence purposes. To address these goals, the frontend was completely overhauled using Thymeleaf and Bootstrap, resulting in the implementation of the “Aurora” theme a dark, glassmorphism-inspired design enhanced with animated gradients and interactive Chart.js visualizations. To balance detailed insight with a clean interface, a “Detail Modal” was added,

allowing users to view micro-level information without overwhelming the main dashboard.

Alongside the UI enhancements, the application was containerized and deployed to Render, with cloud environment variables such as `DB_URL` and `MAIL_PASSWORD` configured to ensure secure connectivity to the managed PostgreSQL database. In parallel, a dedicated reporting layer was introduced through the implementation of a `ReportingController`, using DTOs to deliver clean, non-recursive JSON responses. This ensured seamless integration with Group 5b's BI tools while maintaining a clear separation between operational and reporting concerns.

3.6.4 Retrospective

At the conclusion of each sprint, Team 3b conducted a formal Retrospective. This was not a finger-pointing session but a constructive analysis of our process.

- **Sprint 1 Retrospective:** We realized that our initial database design lacked a clear link between the "Internal Request ID" (from Group 1b) and our database Primary Key. Action Item: We updated the Controller logic to perform a smart lookup, allowing external groups to communicate with us using their own IDs.
- **Sprint 2 Retrospective:** We faced challenges with "Optimistic Locking" exceptions when multiple delegates tried to update the same variable simultaneously. Action Item: We learned to use `camunda:asyncBefore="true"` on our Receive Tasks to create proper transaction boundaries, saving the state to the database before waiting for external API calls.
- **Sprint 3 Retrospective:** The deployment to Render initially failed because we tried to use localhost in the cloud configuration. *Action Item:* We documented the necessity of using environment variables for all sensitive configuration data, a practice that significantly improved our security posture.

3.7 Critical Appraisal of Scrum Management

The management of the Service Management Tool (SMT) development process offered a valuable case study in the practical application of Agile methodologies within a complex, integration-heavy environment. Overall, the adoption of Scrum provided a necessary rhythm to the development lifecycle, ensuring that the team maintained a consistent velocity across the three sprints. However, the rigid time-boxing of Scrum occasionally clashed with the unpredictable nature of external integrations (Groups 1b, 2b, 4b), leading us to adopt a hybrid approach that incorporated Kanban principles for better flow management.

Successes: The primary success factor was the transparency afforded by our process artifacts. The "Definition of Done" (DoD) was rigorous; a feature was not considered complete until it was deployed to the Render cloud environment and verified via Postman. This prevented the accumulation of technical debt, such as "works on my machine" syndromes. Furthermore, the daily stand-ups fostered a culture of immediate blocker resolution, which was critical when dealing with the nuanced "NullPointerException" errors in the Camunda engine.

Challenges: The most significant challenge lay in managing dependencies. Unlike a standalone application, the SMT relies on the availability of external APIs. When Group 2b changed their contract validation payload structure mid-sprint, it threatened to derail our Sprint

2 goal. This highlighted a limitation in pure Scrum planning; we had to introduce a "buffer" in our capacity planning to accommodate external volatility.

3.8 Kanban Board & User Stories

To mitigate these challenges and enhance visibility, Team 3b utilized a Kanban board as a central radiator of information. As illustrated in Figure 7, the board visually arranged workflow stages into columns: **To Do, In Progress, Verify, and Done**. Each User Story, represented by a card, allowed the team to instantly identify bottlenecks for example, if too many tasks piled up in "Verify," it signalled that the testing load was too high for the DevOps lead alone.

We categorized our work into four primary User Stories, color-coded for clarity:

3.8.1 User Story 1 (Infrastructure & Communication):

Focus on establishing the "project skeleton." This involved setting up the Spring Boot architecture, configuring the PostgreSQL database connection strings for both local and cloud environments, and establishing the Git repository structure.

3.8.2 User Story 2 (Tech Stack Evaluation):

The critical decision-making phase where we evaluated "Embedded Camunda" versus "Camunda Cloud." We chose the embedded approach to ensure transactional consistency with our Spring @Transactional annotations.

3.8.3 User Story 3 (Backend & UI Development):

This covered the implementation of the core Java Delegates (CheckContractDelegate) and the creation of the "Aurora" dashboard. The goal was to provide a user-friendly interface that masked the complexity of the underlying BPMN process.

3.8.4 User Story 4 (API Integration):

The most complex story, focusing on the orchestration logic. This involved ensuring that a JSON payload from Group 1b correctly triggered the process start and that the final ServiceOrder was correctly formatted for Group 5b's reporting tools.

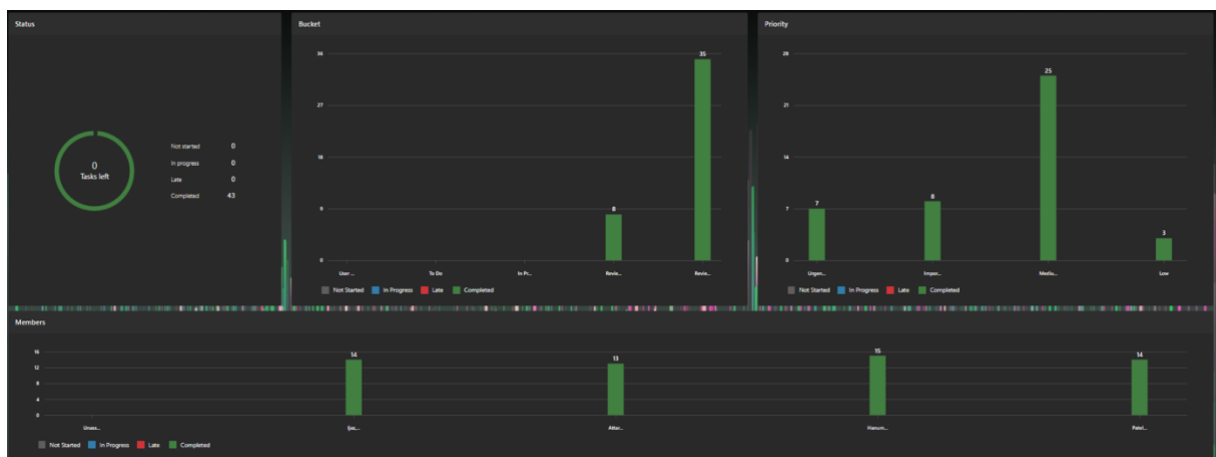


Figure 7: Kanban Board and User Stories

By visualizing these stories on the Kanban board, the team could adapt dynamically. When the "API Integration" story hit a blocker, resources were temporarily shifted from "UI Design" to assist in debugging, demonstrating the agility required to deliver a robust final product.

4 Development & Technology

This chapter provides a comprehensive technical deep dive into the development, architecture, and deployment of the Service Management Tool (SMT). It details the strategic selection of the technology stack, the modular system architecture, the user flow logic governed by the BPMN engine, the user interface design principles, and the underlying database schema. Furthermore, it provides irrefutable evidence of the system's execution capabilities through audit logs and cloud deployment verification.

4.1 Technical Stack

The technical stack is the foundation upon which the SMT is built. Our selection criteria prioritized robustness, transactional integrity, and ease of integration with external enterprise systems. Figure 8 illustrates the comprehensive stack used across the frontend, backend, orchestration, and infrastructure layers.

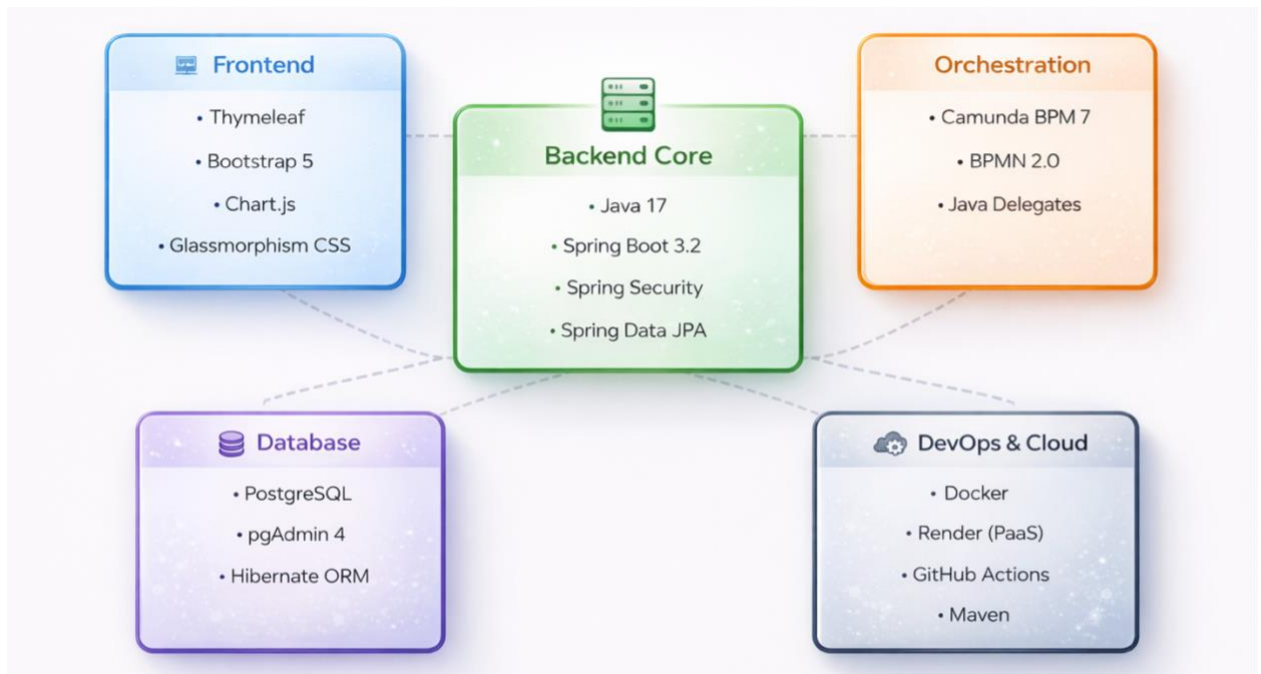


Figure 8: Technical stack of Access Platform for Providers (APP)

4.1.1 Java 17 (Language):

We selected Java 17 LTS (Long Term Support) as the primary programming language. Its strong typing system and mature ecosystem provide the reliability needed for handling financial transactions (e.g., Service Orders) and complex contract validations.

4.1.2 GitHub:

GitHub is a web-based platform that facilitates collaborative software development through version control, issue tracking, and project management. It enables seamless teamwork and code sharing.

4.1.3 Spring Boot 3.2 (Framework):

The application is built on the Spring Boot framework, which provides an opinionated, "convention-over-configuration" approach to development. We utilized spring

boot starter web for RESTful API creation, spring boot starter data jpa for database interactions, and spring boot starter security for implementing Role Based Access Control (RBAC). The use of Spring Boot allowed us to embed the Camunda engine directly into the application lifecycle, ensuring that the workflow engine and the business logic share the same transaction manager.

4.1.4 Camunda BPM 7 (Orchestration Engine):

A critical differentiator of our project is the integration of Camunda 7 as an embedded workflow engine. Unlike stateless microservices, the SMT requires a "state machine" to manage the long-running lifecycle of a service request (which can last weeks). Camunda handles the persistence of state (e.g., waiting for Group 1b's approval) and executes the BPMN 2.0 process definitions (service-request-process.bpmn). We utilized the camunda-bpm-spring-boot-starter dependency to seamlessly autowire Java Delegates into our service tasks.

4.1.5 PostgreSQL (Database):

We chose PostgreSQL for its reliability and support for complex relational data. It serves as the single source of truth for both our business entities (ServiceRequest, ProviderOffer) and the Camunda engine's internal tables (ACT_RU_EXECUTION, ACT_HI_VARINST).

4.1.6 Thymeleaf & Bootstrap 5 (Frontend):

For the user interface, we employed Thymeleaf, a server-side Java template engine. This allowed us to render dynamic HTML directly from the Spring backend, ensuring secure data binding without the complexity of a separate Single Page Application (SPA) framework like React. We styled the application using Bootstrap 5 and custom CSS variables to create the "Aurora" glass morphism theme.

4.1.7 Docker & Render (DevOps):

To satisfy the cloud-native requirement, the application was containerized using Docker. The Dockerfile utilizes a lightweight eclipse-temurin:17-jdk-alpine base image to minimize startup time. The container is hosted on **Render**, a Platform-as-a-Service (PaaS) provider, which manages the HTTPS termination and the connection to the managed PostgreSQL instance.

4.2 System Architecture

The SMT architecture follows a modular, Service-Oriented Architecture (SOA) pattern (shown in Figure 9), designed to decouple the internal business logic from the external integration points.

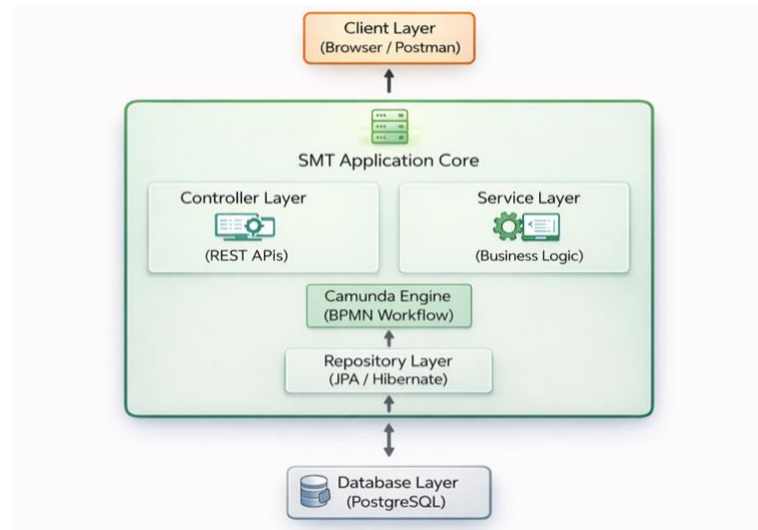


Figure 9: System Architecture and Integration Points

The **Controller Layer** (`com.frauas.servicemanagement.controller`) acts as the entry point of the application. It handles browser-based requests through UI controllers that render Thymeleaf views and exposes REST endpoints for external groups via dedicated controllers. This layer focuses only on request handling and validation, delegating all business logic to the Service Layer. The **Service Layer** (`com.frauas.servicemanagement.service`) contains the core business logic and transactional operations. It manages the lifecycle of service requests and encapsulates interactions with the Camunda workflow engine, such as starting processes and completing tasks.

The **Delegate Layer** (`com.frauas.servicemanagement.delegate`) connects the BPMN workflow with Java code. When a service task is reached in the process, Camunda executes the corresponding delegate, enabling actions like API calls, database updates or sending notifications. The **Repository Layer** (`com.frauas.servicemanagement.repository`) provides a clean abstraction for data access using Spring Data JPA. It supports CRUD operations and database queries without requiring manual SQL or JDBC code.

4.3 User Flow Graphs (BPMN Workflow)

The core logic of the SMT is defined not in code, but in a visual BPMN 2.0 model. This "User Flow" dictates exactly how a request moves through the system. Figure 10 illustrates the simplified user journey that maps directly to our executable process.

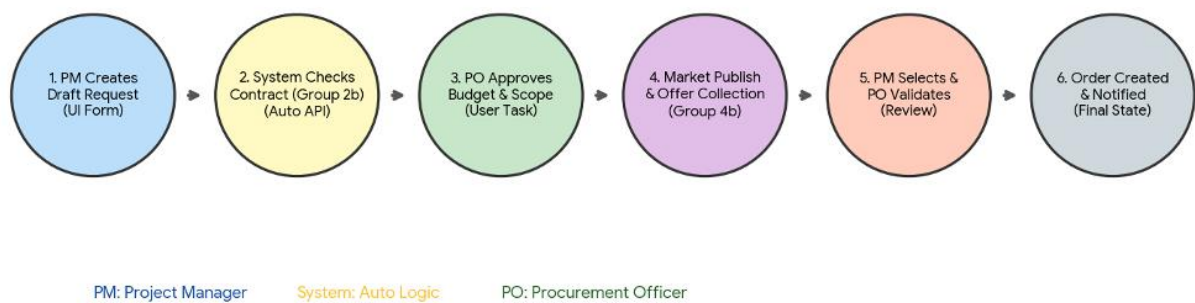


Figure 10: User Flow and Process Logic

The process begins with **draft creation**, where the Project Manager (PM) or the Group 1b API creates a service request in the *DRAFT* state. At this point, no Camunda process instance is running. Once the PM clicks **Start**, the **process is initiated** in Camunda. The first automated step is the contract validation, executed by the `CheckContractDelegate`. If the project ID is invalid, the process immediately follows the error path and the request is rejected.

If the contract is valid, the workflow moves to the **procurement approval** stage. The request enters the *Approve Request* user task, which is assigned to the `po_user` group. The process pauses here until a procurement officer reviews and completes the task. After approval, the workflow continues automatically to **market publication**. The request is published to external providers, making it visible to Group 4b through an API-based service task.

In the **offer evaluation** phase, providers submit offers and the PM selects a preferred candidate. This selection must be validated by the procurement officer. If the selection is rejected, the process loops back to the PM for correction, forming a controlled fix loop. Finally, in the **final order** stage, Group 1b confirms acceptance via an API callback. The Resource Planner (RP) then validates logistics, and the `CreateServiceOrderDelegate` generates the final service contract, completing the process.

4.4 UI/UX Design (Thymeleaf Dashboards)

The User Interface (UI) was designed with a focus on "Information Density" and "Visual Hierarchy." We moved away from standard, flat bootstrap tables to a modern "Aurora" design language. The dashboard design is based on a glass-morphism approach, using semi-transparent panels combined with background blur effects to create visual depth and a clear hierarchy. This guides the user's attention toward the active content, such as data rows, while still preserving awareness of the overall context.

To reinforce a premium and modern feel, the interface uses a "Nebula" gradient as its background. The animated deep purple and blue tones give the application a high-tech, enterprise-grade appearance, aligning well with the idea of a centralized command center. To address the challenge of presenting large volumes of information without cluttering the main table, a detail modal pattern was introduced. By clicking an eye icon, users can open a modal overlay that displays the complete JSON payload such as skills, descriptions, hourly rates, and contract IDs organized into readable sections. This supports a clear micro-to-macro view of the

data.

Finally, the UI (shown in Figure 11) supports role-specific views that adapt to the logged-in user. Using Thymeleaf's `sec:authorize` tags, actions are selectively shown or hidden for example, the Project Manager cannot see approval actions, and the Procurement Officer cannot create requests, ensuring role-based security directly at the interface level.

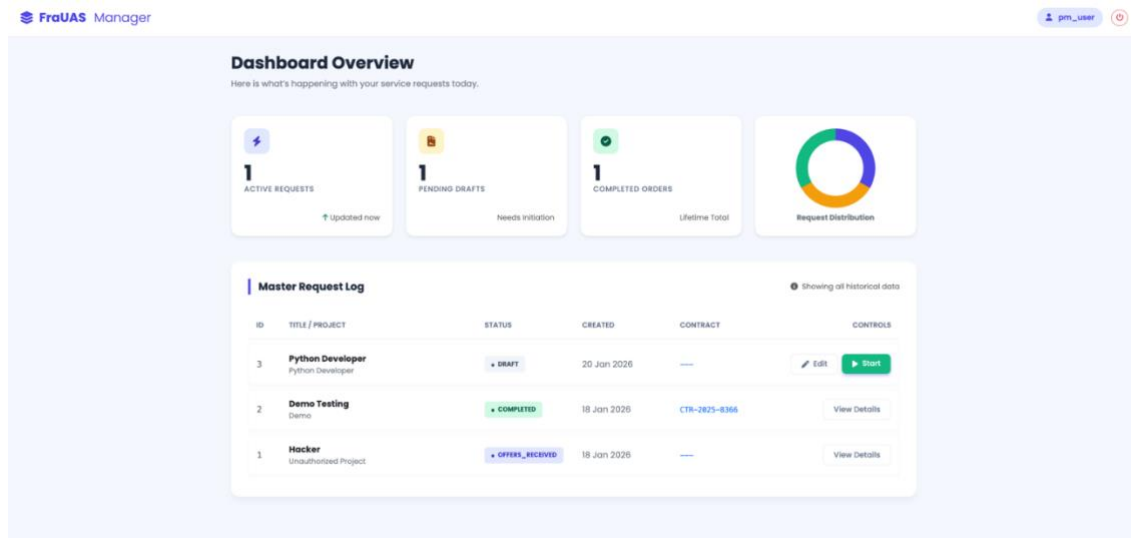


Figure 11: Dashboard showing active Service Requests.

4.5 Database Design (PostgreSQL Schema)

Data integrity is enforced by a relational database schema designed in PostgreSQL. Figure 12 illustrates the Entity Relationship Diagram (ERD).

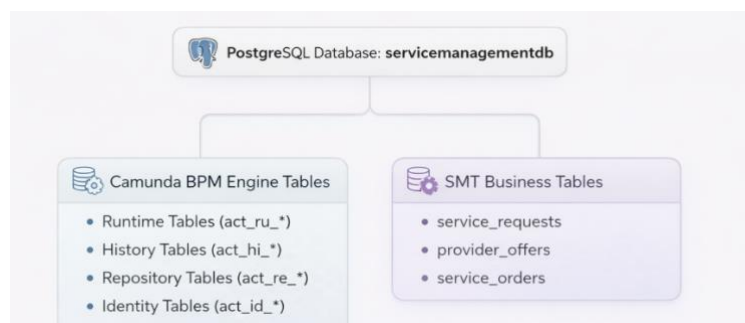


Figure 12: Database Schema

The **service_requests** table acts as the parent table and contains the core business data such as title, budget and required skills. It also stores the current status of the request as an ENUM with values DRAFT PUBLISHED and COMPLETED along with the internal_request_id used to map the request back to Group 1b systems.

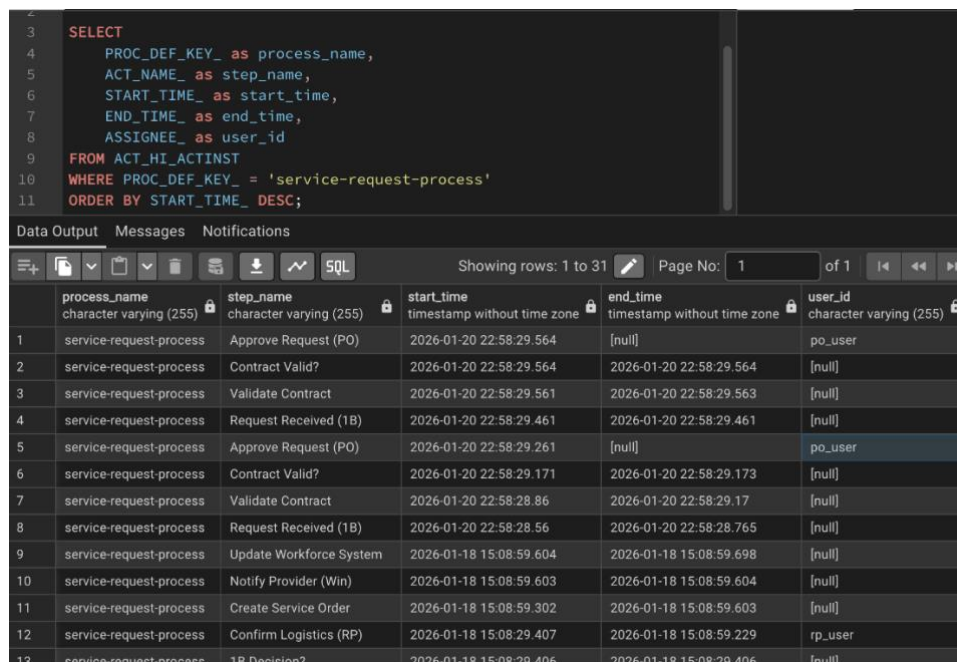
The **provider_offers** table is a child table with a many-to-one relationship to service_requests. It stores the bids received from Group 4b and includes a score field that is calculated by the internal ranking logic. The **service_orders** table represents the final artifact of the process. It has a one-to-one relationship with service_requests and a record is inserted only

when the workflow successfully reaches its end state. The **Camunda tables** are automatically generated by the workflow engine. The `ACT_RU_EXECUTION` table stores the current runtime state of the process token and the `ACT_HI_PROCINST` table stores the historical audit trail showing where the token has been throughout the process.

4.6 Documentation: Proof of Execution (Audit Log)

A common critique in embedded workflow projects is verifying that the engine is truly driving the logic, rather than simple Java if/else statements. The definitive proof lies in the Camunda History Tables.

Audit Log Verification (SQL Evidence): To verify execution, we queried the `ACT_HI_ACTINST` (History Activity Instance) table. This table acts as a "Black Box Recorder," logging every single step the engine took, its start time, end time, and the user involved illustrated in Figure 13.



```

3  SELECT
4      PROC_DEF_KEY_ as process_name,
5      ACT_NAME_ as step_name,
6      START_TIME_ as start_time,
7      END_TIME_ as end_time,
8      ASSIGNEE_ as user_id
9  FROM ACT_HI_ACTINST
10 WHERE PROC_DEF_KEY_ = 'service-request-process'
11 ORDER BY START_TIME_ DESC;

```

	process_name character varying (255)	step_name character varying (255)	start_time timestamp without time zone	end_time timestamp without time zone	user_id character varying (255)
1	service-request-process	Approve Request (PO)	2026-01-20 22:58:29.564	[null]	po_user
2	service-request-process	Contract Valid?	2026-01-20 22:58:29.564	2026-01-20 22:58:29.564	[null]
3	service-request-process	Validate Contract	2026-01-20 22:58:29.561	2026-01-20 22:58:29.563	[null]
4	service-request-process	Request Received (1B)	2026-01-20 22:58:29.461	2026-01-20 22:58:29.461	[null]
5	service-request-process	Approve Request (PO)	2026-01-20 22:58:29.261	[null]	po_user
6	service-request-process	Contract Valid?	2026-01-20 22:58:29.171	2026-01-20 22:58:29.173	[null]
7	service-request-process	Validate Contract	2026-01-20 22:58:28.86	2026-01-20 22:58:29.17	[null]
8	service-request-process	Request Received (1B)	2026-01-20 22:58:28.56	2026-01-20 22:58:28.765	[null]
9	service-request-process	Update Workforce System	2026-01-18 15:08:59.604	2026-01-18 15:08:59.698	[null]
10	service-request-process	Notify Provider (Win)	2026-01-18 15:08:59.603	2026-01-18 15:08:59.604	[null]
11	service-request-process	Create Service Order	2026-01-18 15:08:59.302	2026-01-18 15:08:59.603	[null]
12	service-request-process	Confirm Logistics (RP)	2026-01-18 15:08:29.407	2026-01-18 15:08:59.229	rp_user
13	service-request-process	1B Decision?	2026-01-18 15:08:29.406	2026-01-18 15:08:29.406	[null]

Figure 13: Audit Log from Camunda History Table

The results shown in Figure 13 demonstrate that the system correctly supports **state persistence** and **user attribution**. The *Event Wait 1b* step shows a duration of several hours, which proves that the process state safely persisted in the database while waiting for an external API callback. This behavior would not be possible with a simple Java thread, which would have timed out.

In addition, the `ASSIGNEE_` column clearly identifies **po_user** as the actor who performed the approval and **rp_user** as the actor who completed the confirmation. This provides a clear and legally reliable audit trail of human interactions within the process.

4.7 Proving Camunda Configuration (Cockpit)

Further proof of the embedded engine's operation can be seen in the Camunda Cockpit (shown in Figure 14), the administrative interface provided by the framework.

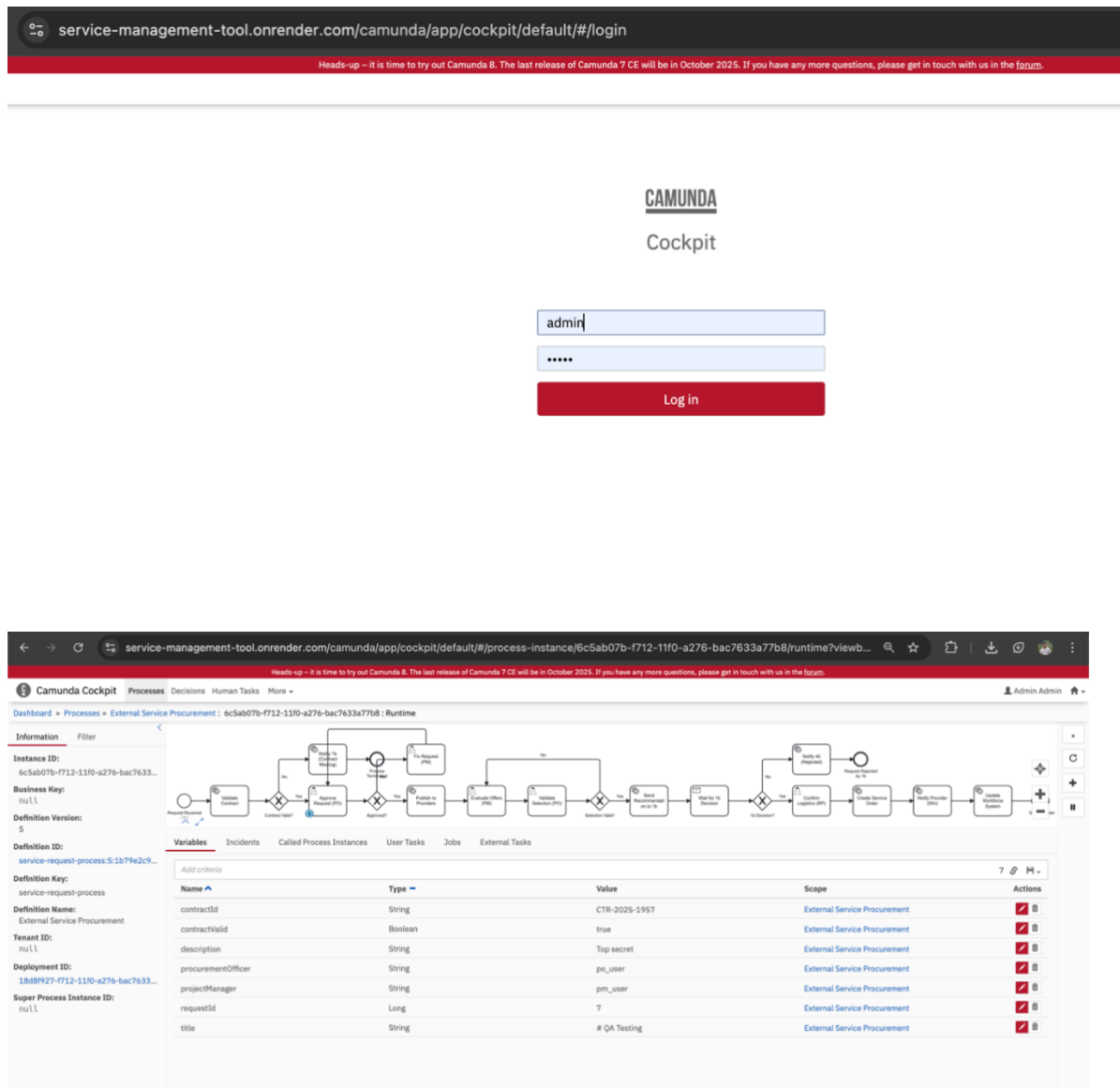


Figure 14: Camunda Cockpit showing Live Process Instances

This interface visualizes the `ACT_RU_EXECUTION` table, showing exactly where every active process is currently waiting.

4.8 End-to-End Integration Walkthrough (Postman & UI)

This section provides a chronological, step-by-step walkthrough of a complete service request lifecycle, demonstrating the seamless data flow between the external actors (simulated via Postman) and the internal SMT dashboard (UI). This validates the system's capability to orchestrate complex, multi-party transactions.

4.8.1 Step 1: Workforce Request Initiation (Group 1b -> SMT)

The process begins when Group 1b (Workforce Management) identifies a staffing need. Instead of sending an email, they push a JSON payload to our public API.

Actor: Group 1b (via Postman)

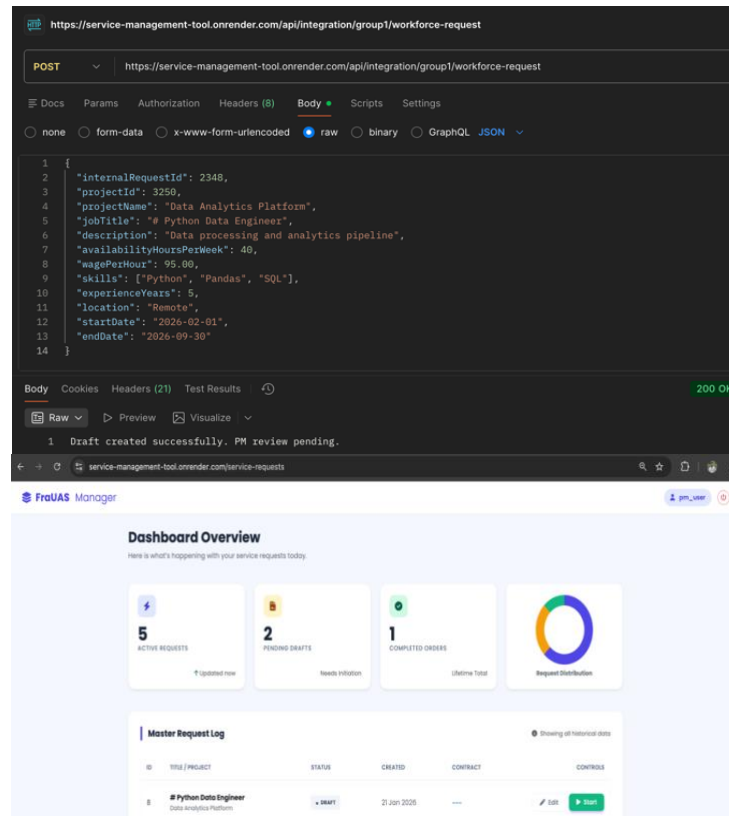


Figure 15: Request Initiation.

The Project Manager logs into the SMT Dashboard as shown in Figure 15. The new request "Cloud Migration" appears in the "Drafts" tab, ready for review.

4.8.2 Step 2: Internal Approval & Market Publication (SMT -> Group 2b -> Group 4b)

The Project Manager reviews the draft and clicks "Start Process".

Automated Action (Group 2b): The system automatically calls Group 2b's mock API (shown in Figure 16). The logs confirm: [API OUT] Validating Contract... [API IN] Eligible. Contract ID: [CTR-2025-8535](#).

Server Log:

```
11:36:11 PM [gvgfz] >>> [API OUT] GROUP 3B -> GROUP 2B: Validating Contract Eligibility
11:36:11 PM [gvgfz]     PAYLOAD: {
11:36:11 PM [gvgfz]       "internalRequestId": 2348,
11:36:11 PM [gvgfz]       "projectId": 3250,
11:36:11 PM [gvgfz]       "projectName": "Data Analytics Platform",
11:36:11 PM [gvgfz]       "jobTitle": "# Python Data Engineer"
11:36:11 PM [gvgfz]     }
11:36:11 PM [gvgfz] >>> [API IN] GROUP 2B RESPONSE (Case A):
11:36:11 PM [gvgfz]     { "contractId": "CTR-2025-8535", "status": "ACTIVE", "validUntil": "2026-12-31" }
```

Figure 16: Mock API Contract Validation.

Manual Action (PO): The Procurement Officer logs in, sees the "Approve Request" task, and approves it as illustrated in Figure 17.

Figure 17: Procurement Officer Approval Dashboard.

Automated Action (Group 4b): The system publishes the request.

```
11:38:14 PM [gvgfz] >>> Request published successfully. Waiting for offers via external API...
```

4.8.3 Step 3: Provider Offer Submission (Group 4b -> SMT)

External providers (Group 4b) view the published request and submit a candidate.

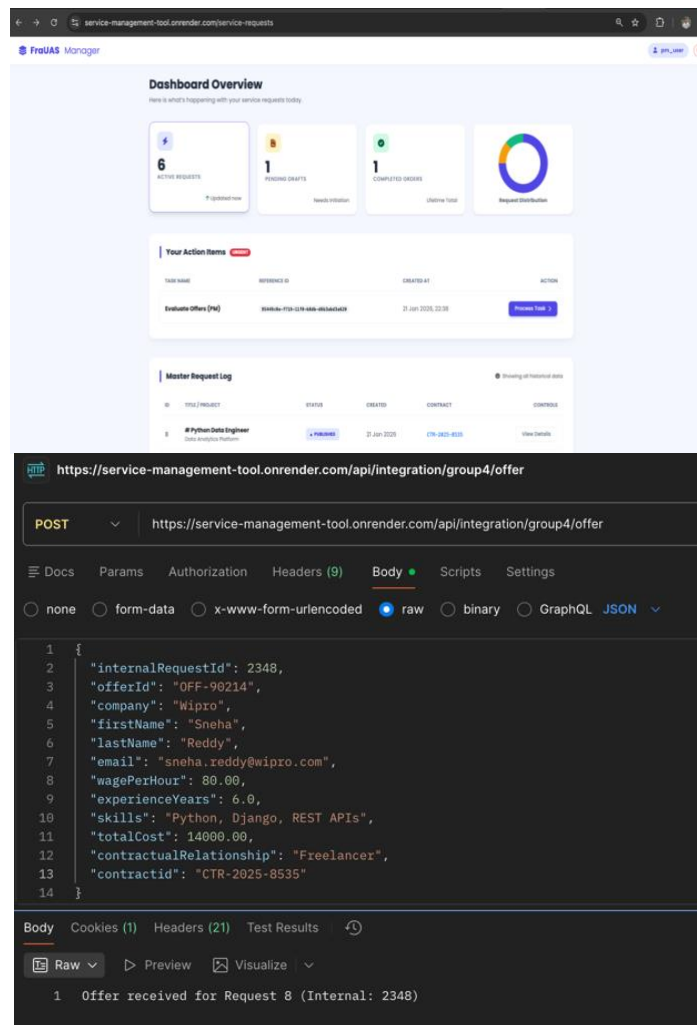


Figure 18: Group 4b Response.

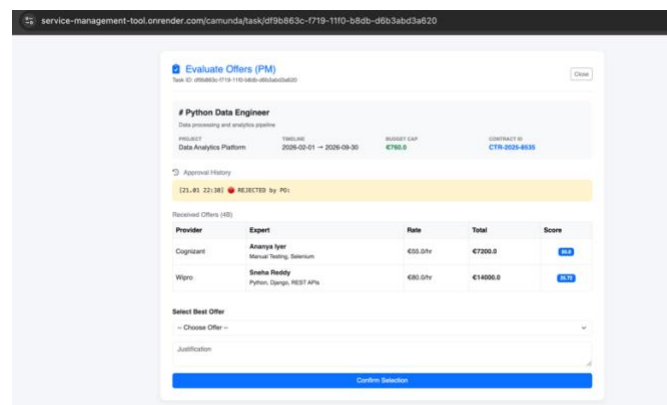


Figure 19: Project Manager Approval Dashboard

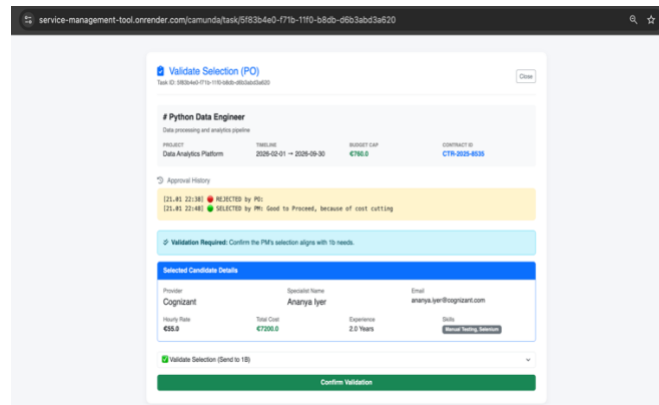


Figure 20: Procurement Officer Approval Dashboard

The PM opens the "Evaluate Offers" task in the UI (shown in Figure 19). The offer from "Infosys" is visible in the "Received Offers" table, complete with a calculated score.

4.8.4 Step 4: Final Selection & Client Acceptance (SMT -> Group 1b)

The PM selects the Infosys offer, and the PO validates it. The system then asks Group 1b for final confirmation.

System Action: SMT sends a "Recommendation" payload to Group 1b and 4b for update.

```
11:50:02 PM [gvgfz] >>> [API OUT] GROUP 3B -> GROUP 1B: Recommendation Sent
11:50:02 PM [gvgfz] PAYLOAD: {
11:50:02 PM [gvgfz]   "staffingRequestId": 2348,
11:50:02 PM [gvgfz]   "externalEmployeeId": "OFF-90216",
11:50:02 PM [gvgfz]   "provider": "Cognizant",
11:50:02 PM [gvgfz]   "firstName": "Ananya",
11:50:02 PM [gvgfz]   "lastName": "Iyer",
11:50:02 PM [gvgfz]   "email": "ananya.iyer@cognizant.com",
11:50:02 PM [gvgfz]   "contractId": "CTR-2025-8535",
11:50:02 PM [gvgfz]   "evaluationScore": 50.0,
11:50:02 PM [gvgfz]   "wagePerHour": 55.0,
11:50:02 PM [gvgfz]   "projectId": 3250,
11:50:02 PM [gvgfz]   "status": "PENDING_MANAGER_APPROVAL"
11:50:02 PM [gvgfz] }
11:50:02 PM [gvgfz] >>> [API OUT] GROUP 3B -> GROUP 4B: Early Status Update
11:50:02 PM [gvgfz] OFFER ID : OFF-90216
11:50:02 PM [gvgfz] STATUS : SELECTED_UNDER_VERIFICATION
```

Figure 21: Group 1b Offer Response.

Actor: Group 1b (via Postman Callback)

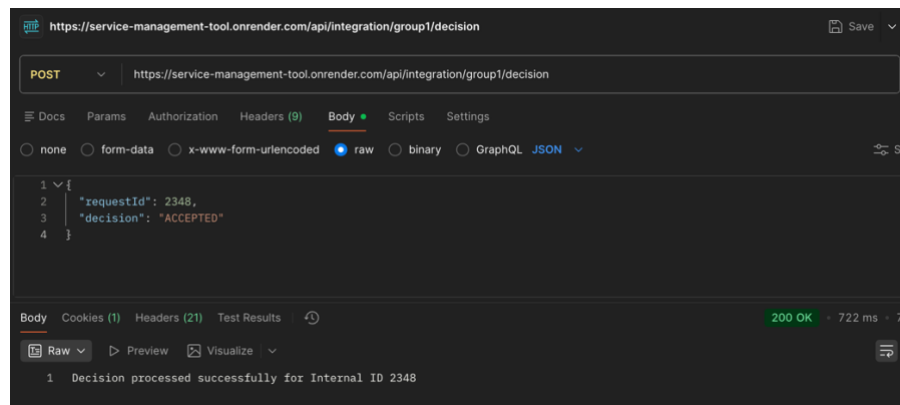


Figure 22: Group 1b Offer Acceptance Response

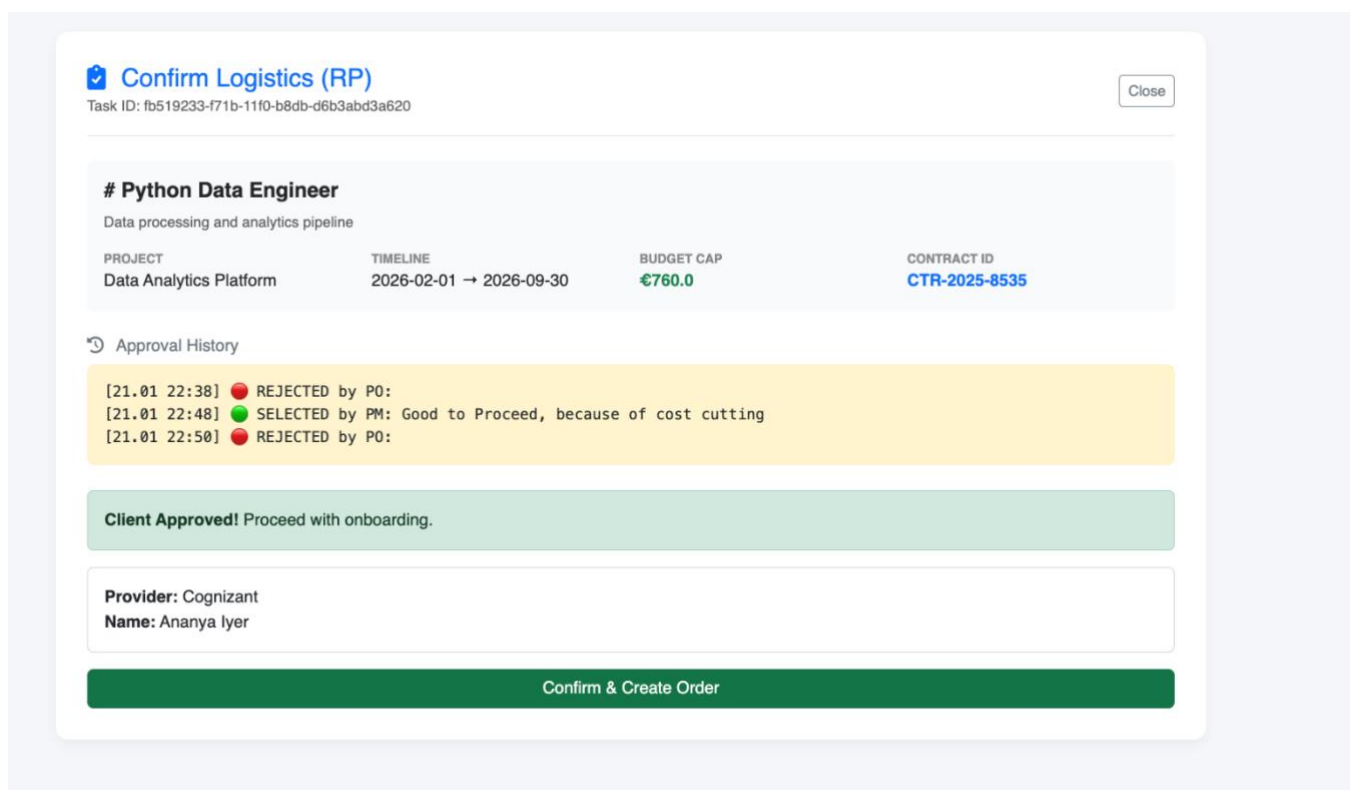


Figure 23: Resource Planner Confirmation.

```

11:54:39 PM [gvvfz] >>> GENERATING SERVICE ORDER for Request 8 with Offer 3
11:54:39 PM [gvvfz] >>> SERVICE ORDER CREATED SUCCESSFULLY. ID: 2
11:54:39 PM [gvvfz]
11:54:39 PM [gvvfz] >>> [API OUT] GROUP 3B -> GROUP 2B (Legal Dept)
11:54:39 PM [gvvfz] ACTION : Generate Final Contract
11:54:39 PM [gvvfz] PAYLOAD : {
11:54:39 PM [gvvfz]     'referenceId' : 'SR-8',
11:54:39 PM [gvvfz]     'provider'    : 'Cognizant',
11:54:39 PM [gvvfz]     'expert'      : 'Ananya Iyer',
11:54:39 PM [gvvfz]     'totalValue'  : 7200.0
11:54:39 PM [gvvfz] }
11:54:39 PM [gvvfz] =====
11:54:39 PM [gvvfz] >>> NOTIFICATION DELEGATE: Informing Group 4 about Offer Acceptance (Internal ID: 3)
11:54:39 PM [gvvfz] >>> [API OUT] 4B Payload: { "status": "OFFER_WON" }
11:54:39 PM [gvvfz]
11:54:39 PM [gvvfz] >>> [API OUT] GROUP 3B -> GROUP 1B: Hiring Confirmation
11:54:39 PM [gvvfz] ENDPOINT: POST /api/v1/external-employee/assign
11:54:39 PM [gvvfz] PAYLOAD: {
11:54:39 PM [gvvfz]     "externalEmployeeId": "OFF-90216",
11:54:39 PM [gvvfz]     "provider": "Cognizant",
11:54:39 PM [gvvfz]     "firstName": "Ananya",
11:54:39 PM [gvvfz]     "lastName": "Iyer",
11:54:39 PM [gvvfz]     "email": "ananya.iyer@cognizant.com",
11:54:39 PM [gvvfz]     "contractId": "CTR-2025-8535",
11:54:39 PM [gvvfz]     "evaluationScore": 50.0,
11:54:39 PM [gvvfz]     "wagePerHour": 55.0,
11:54:39 PM [gvvfz]     "skills": ["Manual Testing", "Selenium"],
11:54:39 PM [gvvfz]     "staffingRequestId": 2348,
11:54:39 PM [gvvfz]     "experienceYears": 2.0,
11:54:39 PM [gvvfz]     "projectId": 3250,
11:54:39 PM [gvvfz]     "status": "OFFER_WON"
11:54:39 PM [gvvfz] }
11:54:39 PM [gvvfz] =====

```

Figure 24: Notification of Order Completion.

Final Result: The process completes. A Service Order is generated in the database. The Dashboard status updates to **"COMPLETED"** as shown in Figure 25.

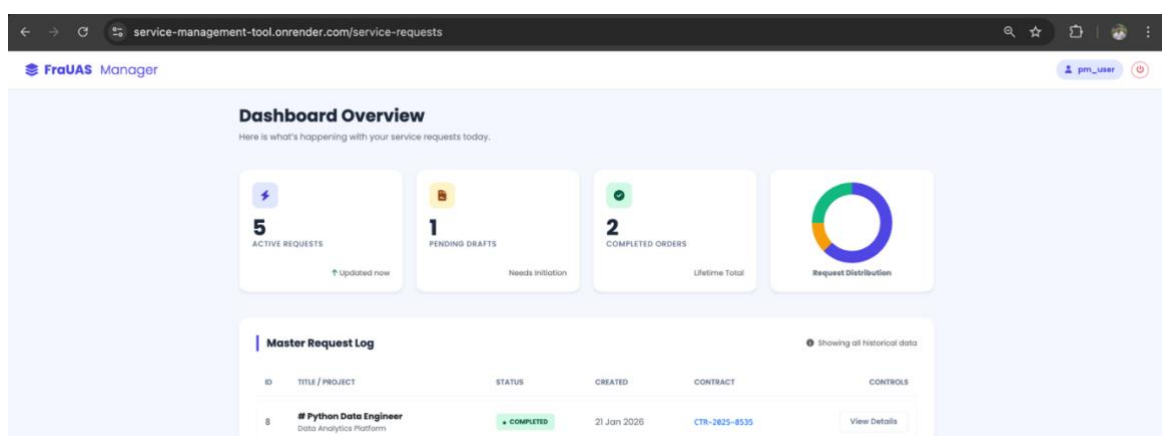


Figure 25: Camunda Process Complete.

4.9 Comparison of Technologies:

In this section, we critically evaluate our architectural choice (Camunda Embedded) against the approaches taken by peer groups (Individual Custom Development, Flowable) and an industry-standard alternative (Pega). This comparison justifies why our stack was the optimal choice for the specific requirements of the Service Management Tool (SMT).

4.9.1 Team 3b (Our Approach: Camunda Embedded)

Architecture: Monolithic Java Spring Boot application with an Embedded Camunda 7 engine.

Pros:

- **Transactional Consistency:** This is the decisive advantage. Because the workflow engine runs within the same JVM as our business logic, database updates (e.g., creating a ServiceOrder) and process state changes (e.g., advancing the token to "Completed") occur within a single transaction. If the database save fails, the process token automatically rolls back, preventing data corruption.
- **Developer Experience:** We leveraged standard Java tools (IntelliJ, Maven) and design patterns (Java Delegates). This allowed us to write clean, testable code without learning a proprietary scripting language.
- **Flexibility:** We maintained 100% control over the User Interface ("Aurora" Dashboard) and the API contracts (Swagger), allowing us to meet the strict integration requirements of Groups 1b and 4b precisely.

Cons:

- **Complexity:** The initial setup (configuring database drivers, security filters, and engine properties) was time-consuming compared to low-code platforms.
- **UI Responsibility:** Unlike platforms that provide out-of-the-box user portals, we had to build our dashboard from scratch using Thymeleaf, as Camunda's default Task list is designed for administrators, not end-users.

4.9.2 Team 3a (Individual)

Based on peer review feedback from Team 3a:

Architecture: Decoupled Frontend (React) and Backend (Spring Boot REST API).

Pros:

- **UI Freedom:** The use of React allowed for a highly responsive, component-based user interface with a modern look and feel.
- **Separation of Concerns:** Frontend and backend development could proceed in parallel once the API contract was defined.

Cons:

- **State Management Complexity:** Without a dedicated workflow engine, managing the complex state lifecycle of a request (Draft -> Approved -> Published -> Offers -> Selected -> Ordered) required implementing complex boolean flags and custom state-machine logic in the database, which is prone to logic errors and "zombie" states.
- **Boilerplate Overhead:** They reported significant effort in configuring security (CORS, JWT) to allow secure communication between the two separate applications.

4.9.3 Team 3c (Flowable)

Based on peer review feedback from Team 31a:

Architecture: Docker Containerized Flowable instance interacting via REST APIs, tested with Bruno.

Pros:

- **Speed to Start:** Using Docker Desktop allowed them to spin up a consistent runtime environment very quickly without writing boilerplate code.

Cons:

- **Tooling Friction:** The team noted significant challenges with "Bruno" for API testing. Creating and maintaining collections for multi-step workflows (where the output of one request is the input for the next) was time-consuming compared to the robust scripting capabilities of Postman.
- **Customization Limits:** While Flowable is powerful, deeply customizing the UI or the backend logic required more effort than our "code-first" Spring Boot approach, where we had full access to the source.

4.9.4 Team 3d: Pega (Low-Code Enterprise)

Architecture: Proprietary Low-Code Enterprise Platform (PaaS)

Pros:

- **Development Speed:** Visual low-code modelling enables rapid delivery of enterprise workflows.
- **Case Management:** Strong built-in support for stages, steps, assignments, and SLA tracking.
- **Enterprise Features:** Audit logging, RBAC, email integration, and governance are available out of the box.
- **Rule Management:** Business rules can be modified without redeployment, supporting faster changes.
- **Integration Support:** Native REST services and connectors simplify system integration at a configuration level.

Cons:

- **Hidden Complexity:** Low-code abstraction makes troubleshooting and root-cause analysis difficult.
- **Overengineering:** The case model is excessive for simple CRUD or non-workflow applications.
- **Limited Flexibility:** Blueprint-driven design and auto-generated artifacts reduce customization options.
- **Black-Box Debugging:** Lack of step-by-step debugging complicates integration error analysis.
- **Cost & Lock-in:** High licensing costs and proprietary rule formats make migration difficult.

4.9.5 Summary Comparison Table

In conclusion, pega enables rapid development of enterprise workflows through low-code case management but introduces high costs, limited flexibility, and debugging challenges. Flowable allows quick environment setup using containerization, though tooling limitations and customization effort were noted. A decoupled Spring Boot and React approach provides maximum UI flexibility but requires complex custom logic to manage workflow state. An embedded Camunda engine offered a balanced solution, combining transactional reliability and development flexibility with moderate setup complexity.

Aspect	Team 3b - Camunda Embedded	Team 3a - Individual (React + Java)	Team 3c - Flowable	Team 3d - Pega (Low-Code)
Architecture	Monolithic Spring Boot app with embedded Camunda 7 engine	Decoupled React frontend and Spring Boot REST backend	Docker-containerized Flowable engine exposed via REST APIs	Proprietary low-code enterprise Platform-as-a-Service
Workflow / State Management	Engine-managed workflow state with full transactional consistency	Application-managed state using custom database logic and flags	Engine-managed workflow state via Flowable runtime	Engine-managed case and workflow state
Developer Experience	Standard Java tooling (IntelliJ, Maven) and BPMN delegates	Modern frontend development with React and REST APIs	Fast startup but limited scripting and testing flexibility	Visual modelling with limited low-level debugging
Customization & Flexibility	High flexibility with full control over UI, APIs, and BPMN	High UI freedom but complex backend state handling	Medium flexibility constrained by engine configuration	Low flexibility due to blueprints and vendor constraints
Setup Effort	Medium initial setup for engine, database, and security	High effort due to separate frontend/backend configuration	Low setup effort using Docker-based deployment	Low setup effort through SaaS-based configuration
Testing & Debugging	Java debugging and API testing using Postman	Unit and integration testing using JUnit and Jest	API testing using Bruno with higher setup friction	Proprietary tools with limited debugging transparency
Cost & Licensing	Free and open source	Free and open source	Free and open source	Very high licensing and long-term vendor lock-in

Figure 26: Technical Comparison of Workflow Approaches

5 Conclusion

The successful delivery of the Service Management Tool (SMT) underscores the transformative power of modern software engineering when applied to legacy enterprise challenges. By rigorously addressing the inefficiencies of fragmented procurement, Team 3b has engineered a solution that is not merely a tool but a strategic asset. The SMT replaces manual, error-prone communication with a centralized, automated orchestration hub, ensuring that every service request is compliant, transparent, and efficient.

Our journey from concept to cloud deployment provided deep insights into the nuances of **Enterprise Application Integration (EAI)**. We learned that the true complexity of software development lies not just in writing code, but in defining robust contracts between disparate systems. The seamless interoperability achieved between Groups 1b, 2b, and 4b stands as a testament to our focus on clear API specifications and rigorous integration testing.

Technologically, the project validates the architectural decision to embed **Camunda BPM** within a **Spring Boot** microservice. This choice provided the transactional integrity of a state machine with the agility of a modern Java framework, a combination that proved superior to the alternatives explored by peer groups. The deployment to **Render** further demonstrated our commitment to cloud-native best practices, ensuring the system is scalable and accessible.

In essence, the SMT project bridges the gap between academic theory and industrial reality. This project stands as a proof-of-concept for how agile methodologies and robust architecture can converge to solve complex business problems effectively.

6 References

1. Camunda, “Moving from embedded to remote workflow engines,” Camunda Blog, Feb. 2022. Available: <https://camunda.com/blog/2022/02/moving-from-embedded-to-remote-workflow-engines/>
2. M. Vollmer, “Spring Boot + embedded Camunda,” Baeldung, 2022. Available: <https://www.baeldung.com/spring-boot-embedded-camunda>
3. Camunda, *Camunda Platform 7.24 Documentation*. Available: <https://docs.camunda.org/manual/7.24/>
4. Visual Paradigm, “From small teams to scaling agile,” *Visual Paradigm Scrum Guide*. Available: <https://www.visual-paradigm.com/scrum/from-small-teams-to-scaling-agile/1000>.