UNIVERSIDADE
LUSÓFONA

Plataforma de onboarding de projetos AMS

Trabalho Final de curso

Relatório Intercalar 1º Semestre

Bernardo Ferreira, a22307944, Engenharia Informática

Duarte Cachata, a22302828, Engenharia Informática

Orientador: Prof. Luís Sousa

Co-orientador: Prof. José Cascais Brás

Entidade Externa: CGI

Departamento de Engenharia Informática da Universidade Lusófona

Centro Universitário de Lisboa

2/11/2025

www.ulusofona.pt

Direitos de cópia

2 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.

Agradecimentos

Resumo

O presente Trabalho Final de Curso consiste no núcleo *back-end* de uma plataforma AMS (*Application Management Services*), que tem como objetivo estruturar e centralizar a gestão operacional de aplicações empresariais. Este componente é responsável por garantir a consistência, normalização e governação dos dados essenciais ao serviço, abrangendo entidades como incidentes, problemas, mudanças, contratos de serviço, ambientes, artigos de conhecimento e informação de *discovery*. Toda a lógica de negócio é implementada em Java com Spring Boot, que expõe uma API REST robusta e normalizada para suportar integrações com os restantes módulos do sistema.

A API comunica com uma base de dados SQL, onde toda a informação é armazenada de forma relacional e auditável. O *back-end* inclui ainda mecanismos de autenticação e autorização, auditoria, *logging* estruturado e cálculo de métricas operacionais.

Com esta arquitetura modular, o *back-end* fornece uma base sólida para a evolução futura da plataforma, permitindo a integração com serviços inteligentes, motores analíticos e *pipelines* de automação, respondendo às necessidades operacionais da CGI e aproximando o protótipo de um ambiente empresarial real.

Palavras-chave: AMS, back-end, Spring Boot, REST API, governação de dados, operações de TI, discovery, métricas operacionais

4 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.

Abstract

Modern enterprise environments depend on a large number of applications whose operation must be continuously monitored, maintained, and improved. In the context of Application Management Services (AMS), operational data such as incidents, problems, changes, and service metrics is often fragmented across multiple tools, making it difficult to ensure consistency, traceability, and effective governance.

This project addresses this challenge by designing and developing the core back-end of an AMS platform aimed at centralizing and structuring operational management. The proposed solution provides a unified model to manage key entities such as applications, service contracts, environments, incidents, problems, changes, knowledge articles, and discovery data, enabling better control over service performance and operational processes.

The system is implemented using Java and Spring Boot, exposing a standardized REST API that supports integration with other platform components. A relational SQL database ensures reliable storage, auditing, and consistency of operational data. Additionally, the back-end incorporates authentication and authorization mechanisms, audit logging, structured observability, and the computation of key operational metrics such as MTTR and recurrence rate.

With a modular and extensible architecture, the solution establishes a solid foundation for future evolution, including integration with intelligent assistants, analytical systems, and automation pipelines. This work contributes to addressing real-world AMS operational challenges and brings the prototype closer to a production-ready enterprise solution.

Key-words: AMS, back-end, Spring Boot, REST API, data governance, IT operations, discovery, operational metrics

Índice

6 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.

8 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.

Lista de Figuras

 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.

Lista de Tabelas

Lista de Siglas

API	Interface de Programação de Aplicações
HTML	Linguagem de Marcação de Hipertexto
CMDB	Configuration Management Database
ITIL	Information Technology Infrastructure Library
AMS	Application Management Services
MTTR	Mean Time To Repair
SLO	Service Level Objective
SLA	Service Level Agreement
SRE	Site Reliability Engineering
DR	Disaster Recovery
BCP	Business Continuity Plan
RTO	Recovery Time Objective
RPO	Recovery Point Objective
KPI	Key Performance Indicator
JWT	JSON Web Token
RBAC	Role-Based Access Control
APM	Application Performance Monitoring
L1/L2/L3	Níveis de Suporte
CRUD	Create, Read, Update, Delete
SQL	Structured Query Language
IDE	Integrated Development Environment
CI	Continuous Integration
NBQ	Next-Best Question
IA	Inteligência Artificial
FTE	Full-Time Equivalent

 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.

BFSI	Banking, Financial Services & Insurance
ERP	Enterprise Resource Planning
CRM	Customer Relationship Management

1 Introdução

1.1 Enquadramento

As organizações modernas dependem cada vez mais de aplicações informáticas para suportar as suas operações diárias, desde sistemas de faturação e plataformas de *e-commerce* até ferramentas internas de gestão e comunicação. A indisponibilidade ou mau funcionamento destes sistemas pode ter impacto direto no negócio, como perdas financeiras, interrupções de serviço ou degradação da experiência do utilizador. Neste contexto, torna-se essencial garantir a manutenção contínua, monitorização e evolução destas aplicações, tarefa que é assegurada através de serviços de *Application Management Services* (AMS). Estes serviços incluem atividades como a resolução de incidentes (falhas inesperadas), a análise de problemas recorrentes, a gestão de mudanças e a monitorização do desempenho dos sistemas.

A literatura recente em gestão de serviços de TI demonstra que a eficácia operacional depende fortemente da estruturação adequada destes processos de suporte, incluindo a gestão de incidentes, problemas, mudanças, conhecimento e contratos de serviço [1] [2]. Adicionalmente, conceitos como observabilidade (a capacidade de monitorizar e compreender o estado interno dos sistemas através de métricas, logs e eventos) e a medição do desempenho através de indicadores como MTTR (*Mean Time to Resolution*) ou SLOs (*Service Level Objectives*) têm ganho relevância em abordagens modernas de engenharia de fiabilidade, nomeadamente no contexto do Google SRE [3]. Esta complexidade operacional verifica-se diretamente no contexto da CGI, que enfrenta o desafio recorrente de avaliar, estruturar e estabilizar projetos AMS com níveis distintos de maturidade, documentação e risco.

Neste contexto, o presente Trabalho Final de Curso insere-se como um exercício integrado de engenharia de software orientado ao desenvolvimento do núcleo operacional de uma plataforma AMS. O projeto é concebido de forma modular, envolvendo três áreas distintas: o *back-end* operacional (Grupo 1), a interface inteligente e automação de *intake* (Grupo 2), e os mecanismos analíticos e de apoio à decisão, incluindo modelos de priorização de questões, risco e continuidade (Grupo 3). Este relatório incide exclusivamente sobre o contributo do Grupo 1, responsável pela construção da camada de dados, APIs, segurança, auditoria e métricas que sustentam todo o sistema.

Do ponto de vista académico, o desenvolvimento deste tipo de solução representa um exercício completo e interdisciplinar de engenharia de *software*, combinando conhecimentos de modelação de dados, arquitetura de sistemas, segurança, integração e observabilidade. Ao seguir princípios amplamente utilizados na indústria, este trabalho aproxima o contexto formativo da prática profissional, permitindo aplicar conceitos teóricos em problemas reais de gestão de serviços tecnológicos. Deste modo, o projeto assume-se como uma resposta concreta às necessidades operacionais da CGI, integrando fundamentos

académicos com a implementação prática de soluções orientadas à gestão eficaz de serviços AMS.

1.2 Motivação e Identificação do Problema

A CGI, enquanto consultora multinacional com forte presença na gestão de serviços tecnológicos, enfrenta frequentemente o desafio de avaliar se deve ou não aceitar determinados projetos de *Application Management Services* (AMS). Cada projeto apresenta características próprias, níveis distintos de maturidade operacional, documentação incompleta ou inexistente e graus variáveis de risco. Esta variabilidade torna complexa a decisão de aceitar, recusar ou renegociar projetos, sobretudo quando não existe um processo estruturado de recolha e validação de informação que permita analisar, de forma objetiva, o estado real das aplicações a suportar.

Um dos principais problemas identificados é a ausência de informação consolidada e estruturada, essencial para qualquer transição AMS. A falta de dados fiáveis sobre incidentes, mudanças, ambientes, integrações, níveis de serviço e práticas de operação reduz significativamente a capacidade de prever o comportamento da aplicação após a transição. Em termos técnicos, muitas equipas iniciam projetos sem informações críticas como observabilidade (métricas, *logs*, *dashboards* ativos), continuidade de negócio (DR/BCP, RTO/RPO), mapa de integrações (proprietários, erros comuns, mecanismos de *retry/replay*), ou gestão de acessos (perfis e aprovadores por ambiente). Esta inexistência de evidência concreta leva a triagem lenta, longos períodos de estabilização e maior exposição a incidentes.

Adicionalmente, diferentes indústrias e tipos de solução exigem abordagens distintas, tornando insuficiente o uso de *checklists* genéricos. Setores como *healthcare*, BFSI ou *retail* possuem requisitos específicos de *compliance* e rastreabilidade, ao passo que aplicações ERP, CRM ou sistemas customizados apresentam padrões de risco e complexidade próprios. A falta desta adaptação ao domínio funcional e tecnológico cria lacunas importantes na avaliação inicial.

Por outro lado, a inexistência de informação operacional básica como volumetria, SLAs/SLOs, janelas de serviço, níveis de suporte (L1/L2/L3), línguas, sazonalidade ou grau de automação dificulta o dimensionamento de equipa e estimativa de esforço. Sem estes dados, a previsão de carga e o modelo comercial tornam-se incertos, aumentando a probabilidade de subdimensionamento (com impacto negativo nos SLAs) ou sobredimensionamento (com custos incompatíveis com o cliente).

O desenvolvimento deste *back-end* surge, assim, da necessidade de criar uma plataforma que apoie a CGI no onboarding estruturado de projetos AMS, reunindo informação operacional relevante, validando-a de forma consistente e proporcionando uma

2 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.

base sólida para avaliar riscos, esforço e maturidade. Ao sistematizar a recolha de dados e evidências, a plataforma reduz incertezas, antecipa dificuldades, identifica lacunas críticas e melhora significativamente a previsibilidade do processo de decisão sobre a aceitação de novos projetos. Desta forma, contribui para maior rigor, consistência e eficiência em todo o ciclo de transição e operação AMS.

1.3 Objetivos

O objetivo principal deste Trabalho Final de Curso é desenvolver o núcleo operacional do *back-end* da plataforma de Application Management Services (AMS) da CGI. Este núcleo será responsável por organizar, centralizar e disponibilizar toda a informação essencial ao funcionamento de projetos AMS, garantindo uma base estável, consistente e confiável para os restantes componentes da solução.

De forma prática, o trabalho visa criar uma API capaz de gerir entidades fundamentais como aplicações, contratos de serviço, ambientes, incidentes, problemas, mudanças, *releases*, conhecimento e métricas operacionais. O sistema deverá ainda disponibilizar indicadores importantes, como MTTA, MTTR e recorrência, permitindo uma visão clara do estado e desempenho de cada aplicação.

Além de suportar a operação diária, este back-end pretende servir como fundação para os outros grupos do projeto, fornecendo dados estruturados que alimentam tanto a interface inteligente (Grupo 2) como o motor de IA responsável por *scoring* e análise de risco (Grupo 3). Assim, o trabalho estabelece uma base sólida para evoluções futuras, promovendo maior eficiência, transparência e qualidade na gestão de projetos AMS dentro da CGI.

1.4 Estrutura do Documento

Na Secção 1, é apresentado o enquadramento geral do trabalho, incluindo o contexto em que o projeto surge, o problema identificado, bem como os objetivos principais da solução a desenvolver.

De seguida, na Secção 2, é realizada a análise da pertinência e viabilidade do projeto, abordando a necessidade da plataforma no contexto da CGI e justificando o impacto que esta terá no processo de *onboarding* de projetos AMS.

Posteriormente, na Secção 3, encontra-se a especificação e modelação do sistema, onde são descritos os requisitos funcionais e não funcionais, os modelos conceptuais e a estrutura base que sustenta o desenvolvimento do *back-end*.

Na Secção 4, é apresentada a solução proposta, detalhando a arquitetura do *back-end*, as tecnologias selecionadas e a forma como cada componente contribui para a resposta ao problema identificado.

Seguidamente, na Secção 5, são descritos os testes realizados e os procedimentos de validação, garantindo que a solução cumpre os requisitos definidos e funciona de forma estável e coerente.

Na Secção 6, é exposto o método de trabalho adotado, assim como o planeamento geral do projeto, incluindo fases, tarefas e organização temporal do desenvolvimento.

Posteriormente, na Secção 7, são apresentados os resultados obtidos, evidenciando a evolução do sistema e o grau de cumprimento dos objetivos inicialmente estabelecidos.

Por fim, na Secção 8, é apresentada a conclusão do trabalho, sintetizando os contributos essenciais do projeto e identificando possíveis direções para trabalho futuro.

4 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.

2 Pertinência e Viabilidade

Esta secção tem como objetivo avaliar a pertinência e a viabilidade da solução proposta, demonstrando de que forma o projeto contribui para resolver o problema identificado e apresenta condições técnicas e organizacionais para a sua implementação de forma sustentável.

O tema do projeto foi proposto diretamente pela CGI no âmbito das necessidades operacionais associadas aos serviços de *Application Management Services* (AMS), especialmente no contexto de transições e *onboarding* de novos projetos. Esta origem reforça a relevância prática da solução, uma vez que responde a problemas reais identificados pela própria organização, nomeadamente a falta de estruturas normalizadas de recolha de informação, a dificuldade em estimar risco e esforço e a ausência de métricas consistentes durante as fases de transição.

2.1 Pertinência

Este projeto apresenta uma elevada pertinência, uma vez que responde diretamente a uma necessidade concreta identificada pela CGI: a inexistência de uma solução centralizada, normalizada e tecnicamente estruturada para suportar o *onboarding* de novos projetos de Application Management Services (AMS). Atualmente, a informação necessária para fazer o *onboarding* de um novo projeto AMS encontra-se dispersa por múltiplas ferramentas e formatos, o que dificulta perceber os riscos, o planeamento do trabalho e a tomada de decisão sobre a aceitação de novos contratos.

A implementação da parte do *back-end* neste trabalho permitirá centralizar dados críticos, padronizar processos de recolha de informação e melhorar a rastreabilidade de todas as atividades relacionadas com o *onboarding*. Além disso, ao disponibilizar uma API REST unificada e um modelo de dados consistente, a solução contribui para a redução de incoerências, aumenta a fiabilidade das métricas operacionais e torna o processo de transição mais previsível e transparente. O facto de o desenvolvimento ser realizado internamente constitui também uma vantagem significativa: evita custos de aquisição de *software*, reduz dependências externas e permite criar uma solução totalmente alinhada com as necessidades específicas das equipas AMS da CGI.

Podemos dividir o impacto positivo do projeto em três dimensões:

- **Organizacional:** melhoria da forma como as equipas nacionais e internacionais trabalham entre si, maior transparência nas fases de transição,ou seja toda a gente consegue ver o que já foi feito, o que falta fazer e o estado do *onboarding*, e criação de um único sítio onde toda a informação correta e atualizada está guardada, evitando confusões e versões diferentes dos dados.

- **Operacional:** Reduz o trabalho manual necessário para reunir e organizar a informação, evita que existam dados repetidos em vários sítios, melhora a qualidade e a fiabilidade dos dados recolhidos e permite identificar possíveis riscos operacionais antes de o projeto entrar em produção.
- **Estratégica:** disponibilização de métricas fiáveis (como MTTR, recorrência de incidentes, volume de mudanças, estado de ambientes) que suportam decisões críticas, como aceitar, renegociar ou rejeitar novos projetos AMS, contribuindo assim para uma gestão mais informada e sustentável do portfólio da empresa.

A validação da pertinência do projeto é assegurada pela própria CGI, que identificou esta necessidade no âmbito das suas operações globais de AMS e acompanhou o desenvolvimento através de orientações técnicas e funcionais. Este envolvimento garantiu que a solução fosse concebida em conformidade com as práticas corporativas e interoperável com os sistemas internos da organização.

2.2 Viabilidade

A viabilidade deste projeto foi avaliada com base em critérios econométricos e organizacionais, garantindo que a solução proposta não só pode ser implementada com sucesso, como também possui condições para continuar a ser desenvolvida e utilizada após a conclusão do TFC, não se esgotando enquanto protótipo académico.

Este TFC encontra-se alinhado com os Objetivos de Desenvolvimento Sustentável que "definem as prioridades e aspirações globais para 2030 em áreas que afetam a qualidade de vida de todos os cidadãos do mundo e daqueles que ainda estão para vir" [4], aos quais achamos mais relevantes:

- Trabalho Digno e Crescimento Económico (ODS nº8): A plataforma contribui para melhorar a eficiência das equipas AMS, reduzindo tarefas manuais, eliminando redundâncias e facilitando processos de transição. Ao otimizar o fluxo de trabalho e melhorar a qualidade da informação, promove um ambiente de trabalho mais produtivo e sustentável a longo prazo.
- Indústria, Inovação e Infraestruturas (ODS nº9): Este projeto introduz uma infraestrutura digital robusta e inovadora para suportar o *onboarding* de novos projetos AMS. A criação de um *back-end* centralizado, com modelo de dados normalizado e API REST, melhora a modernização tecnológica interna da CGI e incentiva práticas de engenharia mais estruturadas e eficientes.
- Produção e Consumo Responsáveis (ODS nº12): A solução proposta reduz desperdício de recursos, evitando múltiplas ferramentas paralelas, documentação duplicada e fluxos de trabalho desestruturados. Garante que a informação e os sistemas são usados de forma adequada, evitando confusão e tornando os processos mais limpos e organizados.

2 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.

A solução proposta apresenta elevada viabilidade, uma vez que será realizada através da utilização de tecnologias acessíveis e confiáveis, como Java, Spring Boot e bases de dados relacionais, que já são amplamente utilizadas e suportadas dentro da CGI, garantindo facilidade de implementação, manutenção e evolução. Do ponto de vista económico, o projeto é sustentável, pois recorre exclusivamente a ferramentas *open-source* e não exige custos adicionais de licenciamento ou aquisição de *software*, ao mesmo tempo que reduz esforço manual, falhas nas transições e retrabalho, gerando benefícios operacionais significativos. Em termos sociais, a solução tem impacto direto nas equipas AMS, ao simplificar tarefas repetitivas, centralizar informação que antes estava dispersa e melhorar a colaboração entre equipas nacionais e internacionais, tornando o processo de *onboarding* mais claro, eficiente e acessível para todos os envolvidos.

2.3 Análise Comparativa com Soluções Existentes

2.3.1 Soluções existentes

Este capítulo apresenta a análise comparativa entre a solução desenvolvida neste projeto e outras alternativas atualmente disponíveis no mercado. Esta comparação é fundamental para destacar o caráter diferenciador do nosso projeto e perceber de que forma a solução proposta se diferencia face às ferramentas já existentes.

Existem várias plataformas de *IT Service Management* amplamente utilizadas, que, embora cubram muitos processos operacionais, não foram desenhadas especificamente para o *onboarding* de projetos AMS, o que justifica a importância desta análise. Para tal, foram consideradas as seguintes soluções: ServiceNow ITSM, Jira Service Management, Freshservice, ManageEngine ServiceDesk Plus, BMC Helix ITSM, Zendesk Suite e SolarWinds Service Desk.

ServiceNow IT Service Management: Plataforma líder no mercado de ITSM, que integra módulos completos para incidentes, problemas, mudanças, contratos, ativos e conhecimento. É altamente personalizável, sendo usada por grande parte das grandes empresas. No entanto, não inclui um módulo específico para onboarding AMS, obrigando a grandes personalizações quando se pretende consolidar dados de transição ou avaliar risco pré-operacional. [5]

Jira Service Management: Ferramenta orientada para equipas técnicas e de desenvolvimento, oferecendo *workflows* flexíveis, gestão de incidentes e mudanças. Apesar de ser bastante versátil, não apresenta qualquer estrutura dedicada para recolher, organizar e analisar informação necessária ao *onboarding* AMS, como ambientes, contratos de serviço ou métricas operacionais consolidadas. [6]

Freshservice: Solução ITSM moderna, com foco na automação e na simplicidade de utilização. Inclui funcionalidades essenciais para incidentes, problemas, mudanças e gestão de ativos. Contudo, apesar da sua interface intuitiva, não contempla processos específicos de transição AMS, nem mecanismos aprofundados para consolidar informação técnica e funcional dispersa. [7]

ManageEngine ServiceDesk Plus: Ferramenta ITSM sólida e bastante popular em empresas que procuram uma solução com boa relação custo-benefício. Disponibiliza gestão de incidentes, problemas, mudanças e CMDB. Ainda assim, não fornece um modelo de dados estruturado para suportar *onboarding* AMS, obrigando as equipas a recorrer a documentação manual, folhas Excel ou integrações externas para recolher informação essencial antes da entrada em suporte. [8]

BMC Helix ITSM: Plataforma robusta e orientada para empresas que exigem elevada automação e integração com soluções AIOps. Oferece *discovery* automático, gestão avançada de ativos e processos ITIL. No entanto, não integra um fluxo dedicado para preparar transições AMS, obrigando a combinar vários módulos independentes para obter uma visão mínima do estado da aplicação. [9]

Zendesk Suite: Muito utilizada para suporte ao cliente e *helpdesk* geral, oferecendo tickets, base de conhecimento e automação básica. No entanto, não é uma solução ITSM completa, e carece totalmente de funcionalidades necessárias ao *onboarding* AMS, como gestão de mudanças, ambientes técnicos, métricas operacionais ou avaliação de risco técnico. [10]

SolarWinds Service Desk: Ferramenta ITSM com módulos de incidentes, problemas, mudanças, ativos e automação. Embora ofereça cobertura para suporte diário, não possui qualquer mecanismo especializado para consolidar dados de transição, avaliar maturidade da aplicação ou preparar a fase inicial de suporte, elementos fundamentais num *onboarding* AMS. [11]

2.3.2 Análise de benchmarking

Com base na análise comparativa realizada, é possível concluir que, embora todas as plataformas analisadas disponibilizem funcionalidades importantes no contexto ITSM, nenhuma delas responde de forma completa aos requisitos específicos do *onboarding* de projetos AMS. Falham principalmente em aspetos essenciais como a normalização da informação de transição, a avaliação de risco pré-operacional, a consolidação automática de métricas AMS, e a centralização de dados provenientes de diferentes equipas e ferramentas. Estes elementos são fundamentais para garantir uma transição eficiente, estruturada e previsível, estão ausentes ou apenas parcialmente presentes nas soluções existentes. Assim, evidencia-se uma falha clara no mercado: as ferramentas analisadas não

4 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.

oferecem suporte dedicado ao processo de onboarding AMS, o que reforça a necessidade e relevância da solução proposta neste projeto.

Na tabela 1 percebemos que, embora todas as plataformas avaliadas ofereçam os módulos ITSM clássicos(**incidentes, problemas, mudanças, *requests*, automação e integrações**) nenhuma delas cobre as necessidades específicas do onboarding de projetos AMS. As ferramentas analisadas revelam um foco claro na operação diária, mas necessitam de capacidades essenciais na fase pré-operacional, como normalização de informação de transição, avaliação de risco técnico, consolidação de métricas AMS e centralização de dados provenientes de equipas distintas.

Mesmo soluções mais avançadas como ServiceNow e Jira SM oferecem componentes potentes de ITSM, mas não incluem mecanismos para organizar *discovery*, validar maturidade operacional ou apoiar decisões sobre aceitação de novos projetos. Por este motivo, as equipas AMS acabam frequentemente por recorrer a documentos manuais, folhas Excel ou fluxos paralelos para recolher a informação necessária.

Assim, a **Tabela 1** demonstra de forma clara que existe uma falha no mercado,mostrando que nenhuma das plataformas disponibiliza um módulo dedicado ao onboarding AMS. A solução proposta destaca-se precisamente por preencher essa falha, oferecendo capacidades de risco, *discovery*, centralização de dados e integração com scoring IA.

Tabela *1* - Análise benchmarking

Funcionalidade	ServiceNow	Jira SM	Freshservice	ManageEngine SDP	BMC Helix	Zendesk	SolarWinds SD	Solução AMS Proposta
Gestão de Incidentes	X	X	X	X	X	X	X	X
Gestão de Problemas	X	X	X	X	X		X	X
Gestão de Mudanças	X	X	X	X	X		X	X
Pedidos de Serviço (Requests)	X	X	X	X	X	X	X	X
CMDB / Gestão de Configurações	X			X	X			X
Gestão de Ativos	X		X	X	X		X	X
Discovery Automático					X			
Workflows de Automação	X	X	X	X	X		X	X
SLAs / SLOs configuráveis	X	X	X	X	X		X	X
Dashboards e Reporting	X	X	X	X	X	X	X	X(focado em AMS)
APIs de Integração	X	X	X	X	X	X	X	X
Autenticação e Perfis (ACL/RBAC)	X	X	X	X	X		X	X
Observabilidade (Links APM)						X		X
Leitura de Logs / Métricas externas						X		
Avaliação de risco operacional								X
Normalização de dados da transição								X
Centralização de informação dispersa								X
Onboarding AMS Dedicado								X
Arquitetura Modular	X	X			X		X	X
Audit Trail (histórico detalhado)	X	X			X			X
Integração com ferramentas externas	X	X	X	X	X	X	X	X

6 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.

2.4 Proposta de inovação e mais-valias

Apesar de existirem várias plataformas no mercado dedicadas à gestão de serviços e monitorização de aplicações, nenhuma delas foi desenvolvida especificamente para apoiar o processo de *onboarding* de projetos AMS. As soluções atuais focam-se sobretudo na operação diária (incidentes, mudanças, pedidos, monitorização), mas não contemplam um mecanismo estruturado para recolher, consolidar e avaliar a informação crítica necessária na fase de transição. Além disso, não consideram aspetos essenciais deste processo, como a normalização de dados, a avaliação de risco, a consolidação automática de métricas operacionais ou a necessidade de integrar informação dispersa entre equipas e ferramentas distintas.

Acreditamos que a solução proposta terá um impacto significativo, pois introduz uma abordagem dedicada ao *onboarding* AMS, reunindo toda a informação que, neste momento, está espalhada por várias ferramentas e equipas. Ao combinar centralização, normalização, métricas confiáveis, integração por API e arquitetura modular, a plataforma responde diretamente às necessidades reais identificadas pela CGI e oferece uma forma mais eficiente, transparente e sustentável de preparar novos projetos para começarem a ser suportados pela equipa AMS. Esta abordagem focada exclusivamente no *onboarding* AMS torna a solução diferente de tudo o que existe atualmente no mercado e aumenta significativamente as hipóteses de ser adotada com sucesso pela CGI.

2.5 Identificação de oportunidade de negócio

A nossa solução introduz uma abordagem colaborativa entre todas as equipas envolvidas nos serviços AMS, ao permitir que informação crítica seja recolhida, organizada e partilhada de forma centralizada. Esta partilha estruturada facilita o envolvimento direto das equipas *nearshore* e internacionais, algo que atualmente está ausente nas ferramentas existentes. Esta funcionalidade não só melhora o trabalho diário das equipas, como também fortalece a comunicação, reduz falhas e promove uma verdadeira cooperação operacional.

Além disso, a plataforma tem potencial para estabelecer integrações e parcerias internas com outras áreas tecnológicas da CGI. Esta capacidade de expansão cria oportunidades de crescimento e sustentabilidade, posicionando o projeto como uma referência interna na gestão eficiente e estruturada de transições AMS.

Estamos motivados a desenvolver uma solução inovadora que responda de forma prática e direta às necessidades das equipas de suporte, preenchendo uma falha evidente no mercado e oferecendo uma ferramenta com elevado valor operacional, organizacional e estratégico. Acreditamos que esta plataforma contribuirá para processos mais claros, eficientes e sustentáveis, tornando o *onboarding* de novos projetos AMS um procedimento mais simples e confiável para toda a organização.

3 Especificação e Modelação

3.1 Análise de Requisitos

3.1.1 Enumeração de Requisitos

Nesta secção, apresentamos os requisitos funcionais e não funcionais levantados durante a análise inicial do projeto.

Nas **tabelas 2 e 3**, estão sumariados todos os requisitos, bem como a sua classificação, classificação e esforço de desenvolvimento.

Tabela 2 - Matriz de requisitos funcionais

ID	Title	Journey Classification	Description	MoSCoW	Classification	Dev. Effort
1	**REQF_01 - Aplicações**	**Identify operational risk**	Criar/listar/atualizar aplicações, com criticidade e responsáveis.	**Must Have**	Bespoke development	M
2	**REQF_02 - Contratos de Serviço**	**Assess operational risk**	Manter contratos de serviço por aplicação, com SLAs/SLOs, janelas de serviço, on-call, idiomas, volumetria.	**Must Have**	Bespoke development	M
3	**REQF_03 - Ambientes**	**Identify operational risk**	Registar ambientes DEV/TEST/UAT/PROD, endpoints e notas.	**Must Have**	Configuration	S
4	**REQF_04 - Integrações e Observabilidade**	**Assess operational risk**	Guardar identificadores de integrações externas e links de observabilidade por aplicação.	**Should Have**	Configuration	M
5	**REQF_05 - Incidentes**	**Assess operational risk**	Criar/atualizar/consultar incidentes, com severidade, origem, timestamps e estado.	**Must Have**	Bespoke development	L
6	**REQF_06 - Ligações a Problemas e KB**	**Assess operational risk**	Associar incidentes a problemas e a artigos de conhecimento.	**Must Have**	Bespoke development	M
7	**REQF_07 - Webhook ITSM**	**Assess operational risk**	Ingestão automática de incidentes e requests via webhook de sistemas ITSM.	**Should Have**	Bespoke development	L
8	**REQF_08 - Mudanças**	**Implement controls**	Criar mudanças Normal/Standard/Emergency, com janela temporal, rollback plan e notas.	**Should Have**	Bespoke development	M
9	**REQF_09 - Releases**	**Implement controls**	Criar releases e associar mudanças.	**Should Have**	Configuration	S

8 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.

ID	Title	Journey Classification	Description	MoSCoW	Classification	Dev. Effort
10	**REQF_10 - Risk Score**	**Make risk decision**	Receber e expor o riskScore (0–100) atribuído pelo motor de IA.	**Must Have**	Bespoke development	M
11	**REQF_11 - Base de Conhecimento**	**Supervise**	Criar artigos de KB com título, passos, validação, evidências e tags.	**Could Have**	Configuration	S
12	**REQF_12 - Runbooks**	**Implement controls**	Criar runbooks com trigger, passos, rollback e KPIs.	**Could Have**	Configuration	M
13	**REQF_13 - Discovery Answers**	**Identify operational risk**	Guardar discovery answers por aplicação: chave, valor, origem, confiança, data.	**Must Have**	Bespoke development	M
14	**REQF_14 - Exportação de Discovery**	**Assess operational risk**	Endpoint para exportar discovery answers para o motor de scoring.	**Should Have**	Bespoke development	S
15	**REQF_15 - KPIs**	**Supervise**	Endpoint de KPIs (MTTA, MTTR, change success rate, top 10 recorrência, disponibilidade) por aplicação e intervalo.	**Should Have**	Bespoke development	M
16	**REQF_16 - Metadados de KPIs**	**Supervise**	Incluir metadados nos resultados de KPI (queryWindow, countedItems, dataSource).	**Should Have**	Bespoke development	M
17	**REQF_17 - Autenticação e RBAC**	**Implement controls**	Autenticação JWT com papéis viewer, ams.operator, ams.owner e RBAC nas rotas.	**Must Have**	Out of the box	S
18	**REQF_18 - Auditoria**	**Supervise**	Registar eventos sempre que entidades são criadas/atualizadas/removidas.	**Must Have**	Bespoke development	M

Tabela 3 - Matriz de requisitos não funcionais

ID	Area	Description
NFR1	**Performance**	p95 latency < 300 ms nos endpoints de leitura; KPIs devem responder em < 2 s para 10k incidentes.
NFR2	**Availability**	Health e readiness endpoints; serviço deve degradar-se graciosamente (modo read-only) se serviços internos falharem.
NFR3	**Security**	Validação JWT, RBAC por função, política de privilégio mínimo, secrets geridos via environment/ secret manager e evitar PII nos logs.
NFR4	**Logging**	Logging estruturado em JSON; métricas por endpoint (latência, erros, throughput).
NFR5	**Infrastructure / Purpose**	Testes unitários + request; ≥70% coverage nos serviços core; análise estática/lint em CI.
NFR6	**Compatibility**	Serviço containerizado; execução local com um único comando (make dev ou npm run dev).
NFR7	**Purpose**	Documentação atualizada no repositório (OpenAPI, README), seed scripts e changelog por release.

 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.

3.1.2 Descrição detalhada dos requisitos principais

Os requisitos abaixo descritos representam as funcionalidades de maior impacto no âmbito do núcleo operacional da plataforma AMS, sendo detalhados quanto aos seus objetivos, dependências, critérios de aceitação e enquadramento nos processos de negócio.

REQF_01 - Aplicações

Objetivo:
Permitir criar, atualizar e consultar aplicações AMS, incluindo criticidade (*Gold/Silver/Bronze*) e responsáveis associados. Garante o registo central de todas as aplicações suportadas.

Dependências:
– REQF_02 (Contratos de Serviço)
– REQF_03 (Ambientes)
– REQF_10 (*Risk Score*)

Critérios de Aceitação:
– Deve permitir criar aplicações com nome único, criticidade e owners.
– Deve validar campos obrigatórios.
– Deve listar relações com contratos, ambientes e incidentes.

Processo de Negócio:
O *Owner* AMS regista uma aplicação, iniciando o processo de *onboarding*.
Relacionado com o caso de uso “Gerir Aplicações”.

REQF_02 - Contratos de Serviço

Objetivo:
Registar SLAs, SLOs, janelas de serviço, regimes de *on-call*, idiomas e volumetria por aplicação.

Dependências:
– REQF_01 (cada contrato pertence a uma aplicação)

Critérios de Aceitação:
– Permitir configurar SLA de resposta e resolução.
– Configurar janelas de serviço e períodos de validade.
– Relacionar obrigatoriamente com *Application*.

Processo de Negócio:
Define o modelo de suporte AMS.
Ligado ao caso de uso “Configurar Contratos de Serviço”.

REQF_05 - Incidentes

Objetivo:
Criar, atualizar e consultar incidentes, com severidade, origem, *timestamps* e ligação a problemas e artigos de conhecimento.

Dependências:
– REQF_06 (Problemas e KB)
– REQF_15 (KPIs operacionais dependem dos incidentes)

Critérios de Aceitação:
– Registar *timestamps* (criação, resolução).
– Suportar severidade e origem.
– Associar incidentes a problemas e KB, quando aplicável.

Processo de Negócio:
Serve de base ao cálculo de MTTA, MTTR e recorrência.
Associado ao caso de uso "Registar Incidente".

REQF_10 - Risk Score (Motor IA)

Objetivo:
Integrar com o motor de IA para obter um score de risco (0–100) baseado nas respostas de *discovery* e na informação operacional.

Dependências:
– REQF_13 (*Discovery Answers*)
– Motor de IA externo

Critérios de Aceitação:
– Enviar ao motor IA os dados necessários para avaliação.
– Atualizar *Application.riskScore* com o resultado.
– Garantir *fallback* seguro em caso de erro externo.

Processo de Negócio:
Utilizado para decidir se uma aplicação pode entrar em regime AMS.
Relaciona-se com o caso de uso "Avaliar Risco da Aplicação".

REQF_13 - Discovery Answers

Objetivo:
Armazenar respostas estruturadas do *discovery*: chave, valor, origem, confiança e evidências.

Dependências:
– REQF_14 (Exportação para IA)
– REQF_10 (*Risk Score* depende deste requisito)

Critérios de Aceitação:
– Suportar origem da resposta (manual, evidência, API).
– Guardar evidências (links, ficheiros, IDs).
– Consolidar informação por aplicação.

Processo de Negócio:
Fornece o conhecimento estruturado para os módulos NBQ e IA.
Associado ao caso de uso "Executar *Discovery*".

REQF_18 - Auditoria

Objetivo:
Registar todas as operações de criação, edição, eliminação ou alteração de permissões no sistema.

Dependências:
– REQF_17 (Autenticação e RBAC)
– Todas as operações CRUD do *back-end*

Critérios de Aceitação:
– Armazenar *timestamp*, utilizador, ação e *payload* alterado.
– Permitir consultas filtradas por entidade, utilizador ou ação.
– Garantir integridade e imutabilidade dos registos.

Processo de Negócio:
Suporta governação, *compliance* e segurança operacional.
Relaciona-se com o caso de uso "Consultar Auditoria".

3.1.3 Casos de Uso/*User Stories*

Os *user stories* apresentados nesta secção foram elaborados com o objetivo de representar, de forma clara e orientada ao utilizador, os diferentes cenários reais de interação com o núcleo *back-end* da plataforma AMS. Isso permite compreender as necessidades e expectativas dos vários perfis envolvidos no processo de *onboarding* e operação de projetos AMS, garantindo que as funcionalidades desenvolvidas respondem de forma eficaz aos desafios operacionais identificados.

Através da sua estrutura centrada no utilizador, os *user stories* facilitam o alinhamento entre os requisitos funcionais, os processos de negócio da CGI e os fluxos técnicos necessários para assegurar uma gestão confiável de aplicações empresariais. Esta abordagem contribui para uma visão integrada da solução, promovendo uma experiência consistente entre equipas de desenvolvimento, operação e gestão de serviço.

Seguidamente, apresenta-se a enumeração dos *user stories*, acompanhados de uma breve descrição contextual que ilustra o respetivo caso de uso no âmbito da plataforma AMS:

EPIC E1- Plataforma de *Onboarding* AMS

FEATURE E1-F1 - Aplicações e Contratos de Serviço

- **E1-F1-US1.1 -** Como *Owner* de Aplicação, quero criar, listar e atualizar aplicações com criticidade e *owners* para que a equipa AMS possa gerir níveis de serviço por aplicação.
- **E1-F1-US1.2 -** Como *Owner* de Aplicação, quero definir contratos de serviço (SLAs/SLOs, janelas de serviço, on-call, idiomas, volumetrias) para que a operação seja mensurável.

FEATURE E1-F2 - Ambientes e Integrações

- **E1-F2-US2.1 -** Como *DevOps Engineer*, quero registar ambientes DEV/TEST/UAT/PROD para que links e notas fiquem disponíveis para operação.
- **E1-F2-US2.2 -** Como Operador AMS, quero guardar URLs de APM/logs por aplicação para aceder rapidamente a *dashboards* de observabilidade.
- **E1-F2-US2.3 -** Como SRE, quero que o *readiness* falhe quando a base de dados está indisponível para garantir uma orquestração segura.

FEATURE E1-F3 - Incidentes, Pedidos e Problemas

- **E1-F3-US3.1 -** Como Operador AMS, quero criar e acompanhar incidentes com severidade e *timestamps* para medir tempos de resposta e resolução.
- **E1-F3-US3.2 -** Como Operador AMS, quero ligar incidentes a problemas e artigos de KB para reduzir recorrência.
- **E1-F3-US3.3 -** Como Engenheiro de Integrações, quero ingerir *webhooks* ITSM para que eventos externos criem incidentes/pedidos de forma automática.

FEATURE E1-F4 - Mudanças e Releases

- **E1-F4-US4.1 -** Como Operador AMS, quero registar mudanças com tipo, janela e *rollback* para que releases sejam controladas.
- **E1-F4-US4.2 -** Como *Release Manager*, quero agrupar mudanças em *releases* para medir defeitos pós-release.
- **E1-F4-US4.3 -** Como Operador AMS, quero consultar o *riskScore* numa *Change* para planear mitigação.

FEATURE E1-F5 - Base de Conhecimento e Runbooks

- **E1-F5-US5.1 -** Como Operador AMS, quero criar artigos de KB com passos, validação e evidências para garantir repetibilidade.
- **E1-F5-US5.2 -** Como Operador AMS, quero criar *runbooks* com *rollback* e KPIs para garantir ações críticas seguras.

FEATURE E1-F6 - Discovery e Intake

- **E1-F6-US6.1 -** Como *Owner* de Aplicação, quero guardar perguntas/respostas normalizadas de *discovery* para que o Grupo 3 possa calcular continuidade.
- **E1-F6-US6.2 -** Como Engenheiro de IA, quero exportar informações de *discovery* de forma consistente para *scoring*.

FEATURE E1-F7 - KPIs e Reporting

- **E1-F7-US7.1 -** Como Service Manager, quero obter KPIs por intervalo temporal para avaliar desempenho.
- **E1-F7-US7.2 -** Como *Service Manager*, quero ser alertado quando o intervalo temporal fornecido é inválido.

FEATURE E1-F8 - Segurança, Admin e Auditoria

- **E1-F8-US8.1 -** Como *Security Officer*, quero RBAC para garantir mínimo privilégio no acesso às funcionalidades da plataforma.
- **E1-F8-US8.2 -** Como *Compliance Lead*, quero *logs* de auditoria para rastrear alterações e garantir conformidade.

Requisitos Não Funcionais (NFR)

E1-NFR1 - Desempenho

- **E1-NFR1-US1.1 -** Como SRE, quero APIs responsivas para que *dashboards* e operações se mantenham utilizáveis.

E1-NFR2 - Fiabilidade

- **E1-NFR2-US2.1 -** Como *On-call Engineer*, quero *health/readiness* e modos de falha seguros para que incidentes sejam mitigados.

E1-NFR3 - Segurança

- **E1-NFR3-US3.1 -** Como *Security Officer*, quero autenticação JWT, RBAC, gestão de segredos e privacidade para garantir proteção de dados.

E1-NFR4 - Observabilidade

- **E1-NFR4-US4.1 -** Como SRE, quero *logs* estruturados e métricas por rota para diagnosticar problemas.

E1-NFR5 - Qualidade

- **E1-NFR5-US5.1 -** Como *Tech Lead*, quero testes automatizados e CI gates para evitar regressões.

E1-NFR6 - Portabilidade

- **E1-NFR6-US6.1 -** Como *Developer*, quero executar o serviço com um único comando para facilitar *onboarding*.

E1-NFR7 - Documentação

- **E1-NFR7-US7.1 -** Como *New Joiner*, quero OpenAPI, README e Changelog atualizados para compreender e utilizar a API rapidamente.

Diagrama 1 - Casos de uso: Core Operacional AMS

O Diagrama de Casos de Uso relativo ao Core Operacional AMS, expresso na **Figura 1** representa uma visão unificada de todas as funcionalidades essenciais que compõem o núcleo do *back-end* da plataforma. Neste cenário, a plataforma AMS é apresentada como o sistema central que agrega, governa e expõe toda a informação operacional necessária ao ciclo de vida de um projeto AMS, estando em conformidade com os requisitos funcionais definidos (REQF_01 a REQF_18).

O modelo identifica três perfis de utilizador humano (**Viewer, Operador AMS e Owner AMS**) cada um com permissões distintas, refletindo o modelo de controlo de acessos (RBAC) do sistema. O **Viewer** interage apenas com funcionalidades de consulta, podendo visualizar aplicações, incidentes, métricas, dados de *discovery* e outros elementos informativos. O **Operador AMS**, por sua vez, desempenha um papel ativo na operação diária, possuindo capacidade para criar e atualizar entidades operacionais como incidentes, problemas, mudanças, releases, artigos de conhecimento e runbooks, além de consultar KPIs e informação consolidada de *discovery*. O ***Owner* AMS** tem o nível mais elevado de responsabilidade, abrangendo tanto as operações diárias como a configuração estrutural da plataforma, incluindo gestão de aplicações, contratos de serviço, ambientes, configurações de observabilidade e integrações técnicas.

O diagrama evidencia ainda a interação com dois sistemas externos relevantes:

1. **A plataforma ITSM**, responsável por enviar eventos operacionais através de *webhooks*, que originam incidentes ou pedidos;
2. **O motor de *scoring* baseado em IA**, que consome dados da plataforma para cálculo de risco e continuidade, devolvendo depois valores como o *riskScore* que passam a integrar a gestão das aplicações.

De forma transversal a todos os casos de uso que alteram informação, surge a funcionalidade de auditoria, que regista cada operação de criação, atualização ou remoção, garantindo conformidade com requisitos internos e com políticas de governação. Em conjunto, o diagrama de Caso de Uso demonstra como o *back-end* AMS suporta todos os processos essenciais de gestão de aplicações empresariais, mantendo clara a separação de responsabilidades entre atores, sistemas externos e mecanismos de governação.

Figura 1 - Diagrama de Caso de Uso “Core Operacional AMS”

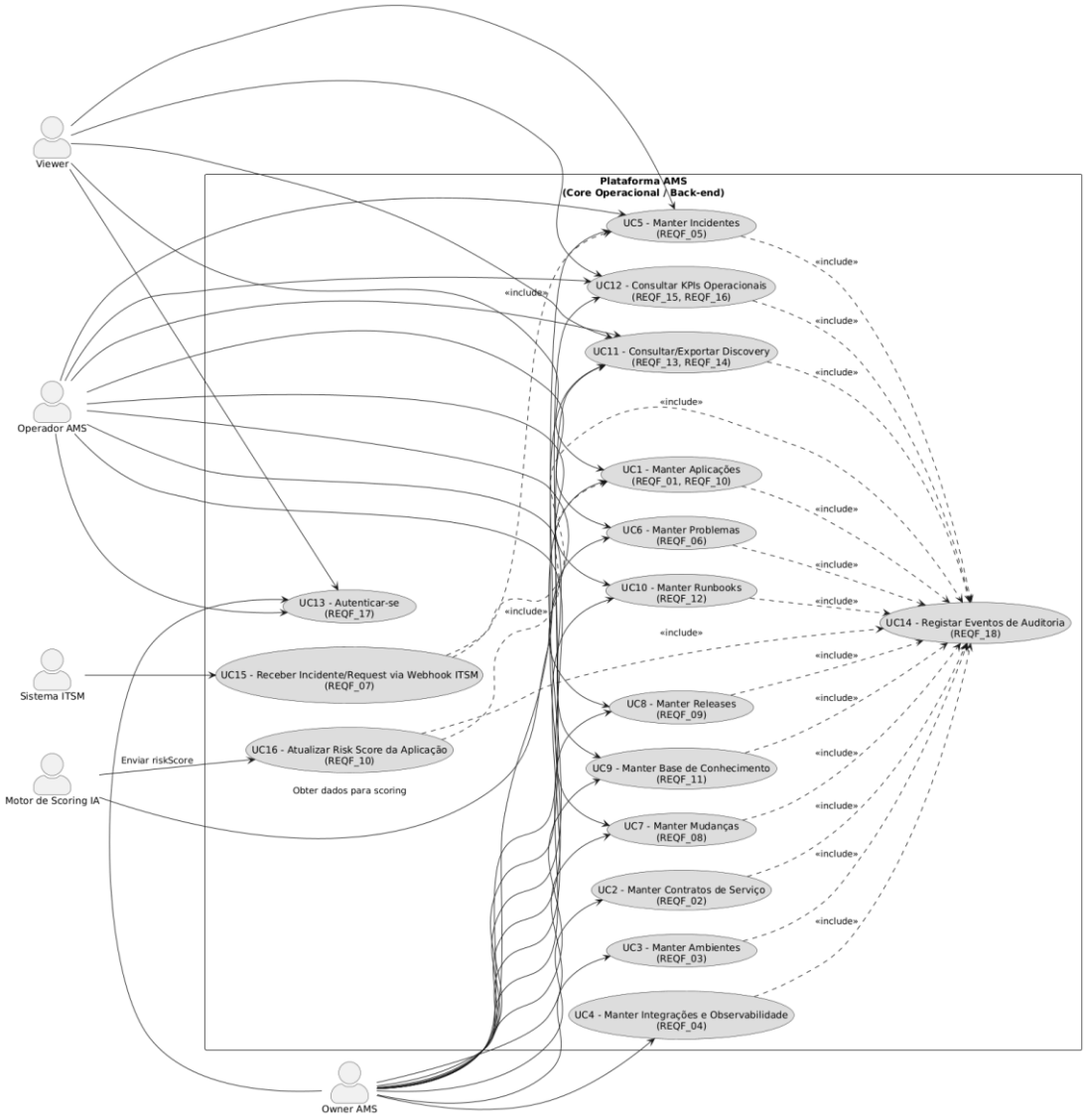


Diagrama 2 - Casos de uso: Discovery & NBQ

O Diagrama de Casos de Uso dedicado ao *Discovery* e ao mecanismo *Next-Best Question* (NBQ), expresso na **Figura 2**, ilustra o funcionamento interno da componente de recolha dinâmica de informação e da interação entre a plataforma AMS e o motor inteligente responsável por selecionar a pergunta mais relevante em cada momento. Esta visão complementa o diagrama anterior, focando-se exclusivamente nos fluxos relacionados com *intake*, questionários e consolidação de conhecimento.

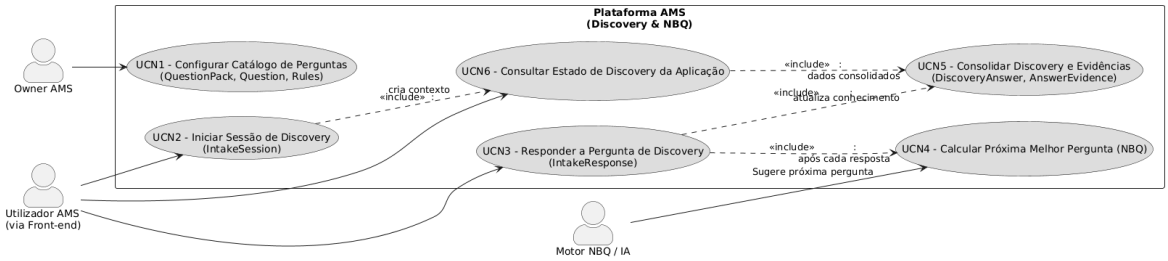
No centro do diagrama da **Figura 2** encontra-se a “Plataforma AMS (*Discovery & NBQ*)”, que gere sessões de *discovery*, respostas dos utilizadores, catálogo de perguntas e

evidências. Os principais atores humanos são o **Utilizador AMS** e o ***Owner* AMS**, que desempenham papéis distintos no processo. O **Utilizador AMS** conduz a sessão prática de *discovery*, podendo iniciar um novo processo de recolha, responder às perguntas apresentadas e consultar o estado atual das respostas conhecidas sobre uma aplicação. Cada resposta submetida é registada como *IntakeResponse* e desencadeia a atualização do conhecimento consolidado (*DiscoveryAnswer*), permitindo ao sistema manter informação coerente, estruturada e reutilizável. O **Owner AMS**, por outro lado, possui responsabilidades de configuração, sendo responsável pela gestão do catálogo de perguntas (packs, perguntas, condições e regras). Esta capacidade permite adaptar o questionário aos diferentes tipos de aplicação, setor ou contexto, sem necessidade de alterar o código da plataforma.

O terceiro ator é o **Motor NBQ/IA**, que desempenha o papel de recomendador inteligente no fluxo. Após cada resposta, o *back-end* invoca este motor para apurar a próxima questão mais relevante com base no estado atual da aplicação, nas respostas anteriores e nas regras parametrizadas no catálogo [12]. Este mecanismo garante um processo de *discovery* adaptativo, eficiente e orientado à redução de risco e incerteza.

A informação recolhida é continuamente consolidada e enriquecida com evidências, permitindo que o sistema mantenha uma representação única, limpa e normalizada do conhecimento sobre cada aplicação. O diagrama demonstra, assim, como o *back-end* AMS suporta nativamente o paradigma NBQ, integrando funcionalidades de gestão do questionário, armazenamento de respostas, consolidação de conhecimento e comunicação com o motor inteligente externo, mantendo sempre uma separação clara entre lógica operacional, regras configuráveis e recomendação automática.

Figura 2 - Diagrama de Caso de Uso "Discovery & NBQ"



3.2 Modelação

Nesta secção apresenta-se o modelo entidade-relação (ER) que suporta o núcleo operacional do *back-end* da plataforma de *onboarding* de projetos AMS. O modelo foi derivado dos requisitos funcionais e não funcionais identificados na Secção 3.1 (REQF_01-REQF_18) e segue princípios de normalização até à 3.ª forma normal, assegurando integridade, coerência e governação dos dados.

O ER inclui todas as entidades fundamentais do domínio AMS: *Application, ServiceContract, Environment, Incident, Problem, Change, Release, KBArticle, Runbook, DiscoveryAnswer*, entre outras, estabelecendo uma base relacional consistente que suporta a gestão diária de operações AMS. Além disso, integra mecanismos de auditoria (*AuditEvent*) e estruturas para integrações externas (ITSM, observabilidade), reforçando a rastreabilidade e robustez dos processos.

Embora este trabalho se centre no *back-end* (Grupo 1), o modelo foi concebido para garantir compatibilidade com a interface inteligente (Grupo 2) e com os mecanismos analíticos e de IA (Grupo 3), seguindo boas práticas de modularidade arquitetural utilizadas em sistemas empresariais.

3.2.1 Integração com o paradigma *Next Best Question* (NBQ)

Uma parte essencial da modelação é o suporte ao paradigma ***Next Best Question* (NBQ)**, responsável por determinar, dinamicamente, qual a pergunta mais informativa a colocar ao utilizador durante processos de *discovery*. Este paradigma baseia-se nos princípios da teoria do valor da informação de Howard (1966) [13], que formaliza o benefício esperado de obter informação adicional antes de tomar uma decisão.

Modelos modernos de **decisão sequencial** (como *adaptive submodularity*) demonstram que abordagens gananciosas (*greedy policies*), que selecionam sempre a próxima pergunta com maior valor esperado, são quase ótimas na prática Golovin & Krause, 2011 [14]. Estes princípios estão amplamente documentados na literatura de diagnóstico ativo e *troubleshooting*, onde perguntas ou testes são escolhidos para reduzir incerteza. [15]

Para operacionalizar o NBQ, o modelo ER inclui um submodelo composto por:

- ***QuestionPack* e *Question*:** catálogo parametrizável de perguntas;
- ***Condition* e *Rule*:** regras determinísticas que suportam lógica adaptativa;
- ***IntakeSession* e *IntakeResponse*:** execução e registo das sessões de *discovery*.

Este desenho segue princípios semelhantes aos usados em *troubleshooting* baseado em redes Bayesianas [16] em *frameworks* de *clarifying* questions na área de conversational IR [17].

O modelo inclui ainda as entidades ***DiscoveryAnswer*** e ***AnswerEvidence***, responsáveis por consolidar conhecimento sobre cada aplicação, incluindo origem, confiança e evidências. Este mecanismo está alinhado com práticas de ***Knowledge-Centered Service* (KCS)**, que defendem a captura contínua de conhecimento durante o fluxo de suporte [18].

Estas entidades não só permitem ao motor NBQ evitar perguntas redundantes, como também fornecem dados estruturados para o motor de *scoring* do Grupo 3, em conformidade com REQF_10.

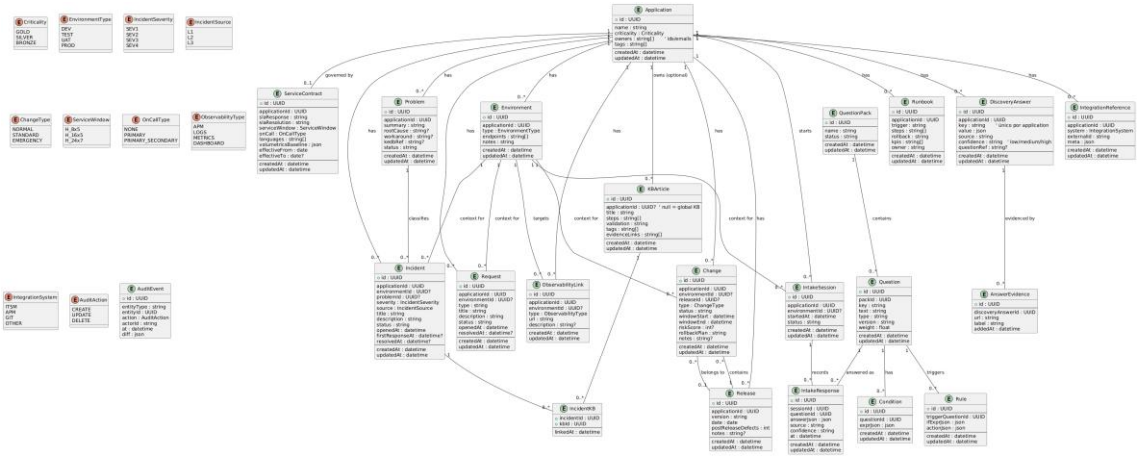
O ER suporta ainda a chamada ao motor de IA externo responsável pela avaliação de risco, cujo resultado é refletido no atributo ***riskScore*** da entidade *Application*. Este fluxo segue práticas comuns em sistemas de decisão automatizada baseados em modelos de utilidade e *scoring* iterativo [19].

Desta forma, o modelo ER, presente na **Figura 3**, cumpre o objetivo central do trabalho: construir um núcleo *back-end* que, para além de suportar a operação diária de projetos AMS, serve como **infraestrutura de dados e serviços** para soluções NBQ mais avançadas, que serão desenvolvidas em camadas superiores da arquitetura.

A Figura 3 apresenta o diagrama entidade-relação (ER) completo do núcleo operacional AMS, incluindo tanto o modelo core como as entidades adicionais responsáveis por suportar o paradigma de ***Next Best Question*** (NBQ). O modelo integra as principais entidades de negócio: *Application, ServiceContract, Environment, Incident, Problem, Change, Release, KBArticle, Runbook, DiscoveryAnswer, AnswerEvidence, ObservabilityLink, IntegrationReference e AuditEvent*, bem como os respetivos relacionamentos e tipos enumerados de suporte.

Este ER global representa, de forma normalizada, todos os requisitos funcionais REQF_01-REQF_18 definidos para o sistema, permitindo gerir aplicações e contratos, registar incidentes e mudanças, manter conhecimento operacional, recolher dados de discovery, consolidar evidências e garantir integridade, governação e auditoria. O modelo serve também de base para o funcionamento dos componentes dos Grupos 2 e 3, assegurando que o *front-end* inteligente e o motor de IA podem consumir dados estruturados e consistentes.

Figura 3 - ER core AMS + NBQ



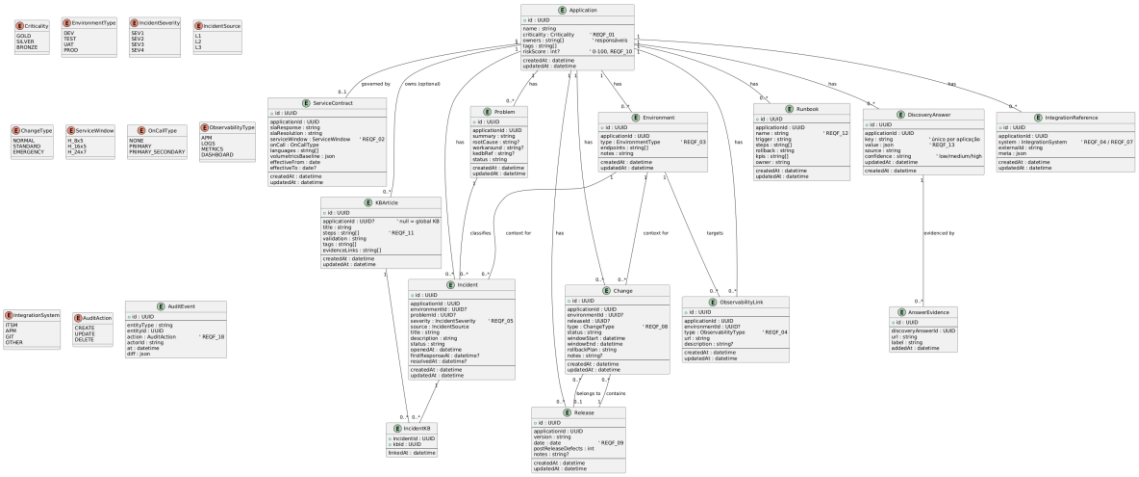
A Figura 4 apresenta o modelo entidade-relação correspondente exclusivamente ao núcleo operacional AMS, excluindo as componentes ligadas ao *discovery* adaptativo e ao NBQ. Este modelo representa as entidades centrais da operação AMS: *Application, ServiceContract, Environment, Incident, Problem, Change, Release, KBArticle, Runbook, ObservabilityLink, IntegrationReference e AuditEvent*, bem como os seus relacionamentos diretos.

O ER Core AMS cobre os requisitos funcionais fundamentais do sistema, nomeadamente:

- **REQF_01-REQF_04** - Aplicações, contratos, ambientes e integrações;
- **REQF_05-REQF_06** - Gestão de incidentes, problemas e ligação à base de conhecimento;
- **REQF_08-REQF_12** - Mudanças, *releases*, conhecimento, *runbooks*;
- **REQF_13-REQF_14** - Suporte a dados de *discovery*;
- **REQF_17-REQF_18** - Segurança, RBAC e auditoria;
- **REQF_15-REQF_16** - Cálculo de KPIs operacionais com base nos dados registados.

Este modelo constitui o esquema relacional central da plataforma, garantindo coerência interna, rastreabilidade e estruturação de todos os processos AMS.

Figura 4 - Modelo ER - Core AMS

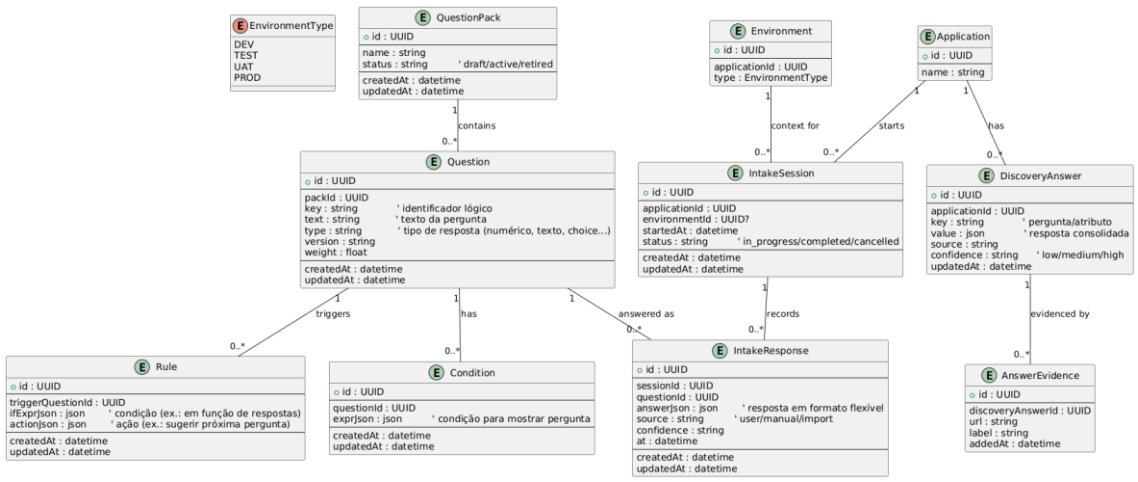


A **Figura 5** representa o submodelo entidade-relação dedicado ao suporte do paradigma ***Next Best Question*** (NBQ) e ao processo de *discovery* estruturado. Este submodelo inclui:

- ***QuestionPack*** *e* ***Question*** - definição do catálogo de perguntas;
- ***Condition*** e ***Rule*** - condições e regras determinísticas para a seleção de perguntas;
- ***IntakeSession*** e ***IntakeResponse*** - registo de sessões e respostas do utilizador;
- ***DiscoveryAnswer*** e ***AnswerEvidence*** - consolidação de conhecimento e evidências sobre cada aplicação.

Estas entidades constituem o "motor de dados" sobre o qual os algoritmos de NBQ (Grupo 3) e a interface inteligente (Grupo 2) operam, garantindo que o processo de *discovery* é adaptativo, registado e auditável. Este submodelo é totalmente compatível com o motor de IA responsável por *scoring* e risco, permitindo integrar *discovery*, recomendações e avaliação contínua da maturidade operacional de cada aplicação AMS.

Figura 5 - Modelo ER - Questionário e Next Best Question



A **Tabela 4** apresenta o mapeamento entre os requisitos funcionais definidos na Secção 3.1 e as entidades do modelo ER. Este resumo permite verificar de forma clara como cada requisito é suportado ao nível de dados, identificando as tabelas, atributos e relações responsáveis por implementar cada funcionalidade da plataforma AMS. Esta correspondência demonstra que o modelo se encontra alinhado com os objetivos operacionais do projeto e garante cobertura completa das necessidades do *back-end*.

Tabela 4 - Mapeamento dos Requisitos Funcionais para as Entidades do Modelo ER

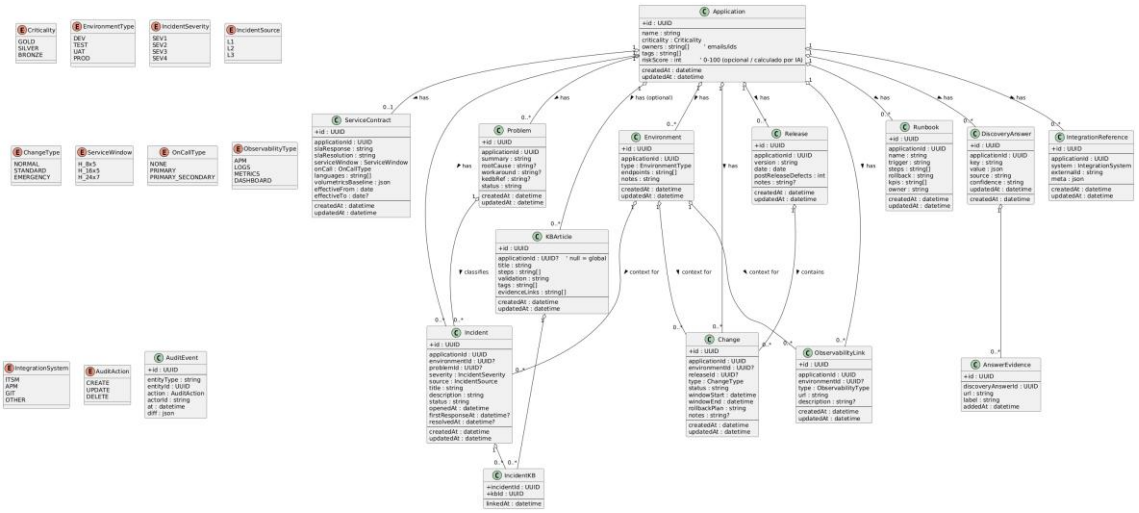
Requisito	Entidades / Atributos no ER
REQF_01 - Aplicações	**Application:** id, name, criticality (GOLD/SILVER/BRONZE), owners[], tags[], riskScore (derivado por IA, ligado a REQF_10).
REQF_02 - Contratos de Serviço	**ServiceContract:** id, applicationId, slaResponse, slaResolution, serviceWindow, onCall, languages[], volumetricsBaseline, effectiveFrom, effectiveTo; relação 1-1 com **Application**.
REQF_03 - Ambientes	**Environment:** id, applicationId, type (DEV/TEST/UAT/PROD), endpoints[], notes; relação 1-N com **Application**.
REQF_04 - Integrações e Observabilidade	**ObservabilityLink:** id, applicationId, environmentId?, type (APM/LOGS/METRICS/DASHBOARD), url, description. **IntegrationReference:** id, applicationId, system (ITSM/APM/GIT/OTHER), externalId, meta.
REQF_05 - Incidentes	**Incident:** id, applicationId, environmentId?, problemId?, severity, source, title, description, status, openedAt, firstResponseAt, resolvedAt; relações com **Application, Environment, Problem**.
REQF_06 - Ligações a Problemas e KB	Ligação **Incident - Problem** via problemId; entidade **Problem:** id, applicationId, summary, rootCause, workaround, kedbRef, status. Tabela de junção **IncidentKB** (incidentId, kbId, linkedAt) entre **Incident e KBArticle**.
REQF_07 - Webhook ITSM	**IntegrationReference** (system = ITSM, externalId, meta) para mapear origem ITSM; destino de ingestão: entidades **Incident** (e opcionalmente **Request**, se modelada) ligadas a applicationId/environmentId. (Requisito é sobretudo de API, mas é suportado por estas entidades).
REQF_08 - Mudanças	**Change:** id, applicationId, environmentId?, releaseId?, type (NORMAL/STANDARD/EMERGENCY), status, windowStart, windowEnd, rollbackPlan, notes; relação com **Application, Environment e Release**.
REQF_09 - Releases	**Release:** id, applicationId, version, date, postReleaseDefects, notes. Relação 1-N **Release - Change** (campo releaseId em Change).
REQF_10 - Risk Score	Atributo riskScore (int, 0-100) em **Application**, atualizado por serviços externos de scoring
REQF_11 - Base de Conhecimento	**KBArticle:** id, applicationId? (null = global), title, steps[], validation, tags[], evidenceLinks[], createdAt, updatedAt; relação com **Application** e **Incident** via **IncidentKB**.
REQF_12 - Runbooks	**Runbook:** id, applicationId, name, trigger, steps[], rollback, kpis[], owner, timestamps; relação N-1 com **Application**.
REQF_13 - Discovery Answers	**DiscoveryAnswer:** id, applicationId, key (único por aplicação), value (json), source, confidence, updatedAt, createdAt; relação N- com **Application**. Evidências ligadas via **AnswerEvidence** (discoveryAnswerId, url, label, addedAt).
REQF_14 - Exportação de Discovery	Não exige nova tabela: o endpoint expõe dados de **DiscoveryAnswer** (e opcionalmente **AnswerEvidence**) filtrados por applicationId. O ER garante chave/valor por aplicação e metadados necessários.
REQF_15 - KPIs	KPIs são derivados sobre entidades do core: **Incident** (openedAt, firstResponseAt, resolvedAt, severity, status) para MTTA/MTTR e recorrência; **Change + Release** para taxa de sucesso de mudanças; **Incident** (e eventualmente Environment/ServiceContract) para disponibilidade. Não é necessária tabela específica de KPIs no ER.
REQF_16 - Metadados de KPIs	Também implementado a nível de serviço/endpoint. Os metadados são calculados sobre as mesmas entidades de REQF_15 (**Incident, Change, Release, Application, Environment**); não requer entidade própria no ER.
REQF_17 - Autenticação e RBAC	Requisito de **segurança da API**, não diretamente modelado no ER (tokens e roles são geridos a nível de identidade/JWT). Opcionalmente pode mapear-se para tabelas de User/Role fora do núcleo AMS, mas não são necessárias para o modelo de dados apresentado.
REQF_18 - Auditoria	**AuditEvent:** id, entityType, entityId, action (CREATE/UPDATE/DELETE), actorId, at, diff (json). Esta tabela permite auditar qualquer entidade do modelo (Application, Incident, Change, etc.).

3.2.2 Diagrama de Classes

O **Diagrama de Classes do núcleo operacional AMS** representa a estrutura orientada a objetos utilizada pela API para implementar as entidades principais do sistema. Este modelo mostra as classes *Application, ServiceContract, Environment, Incident, Problem, Change, Release, KBArticle, Runbook e AuditEvent*, bem como os seus atributos e associações.

Este diagrama da **Figura 6** complementa o modelo ER, ilustrando como as entidades são materializadas na camada de lógica de negócio e como se relacionam através de serviços e controladores. As classes refletem as estruturas utilizadas no domínio AMS e suportam as operações CRUD expostas pela API.

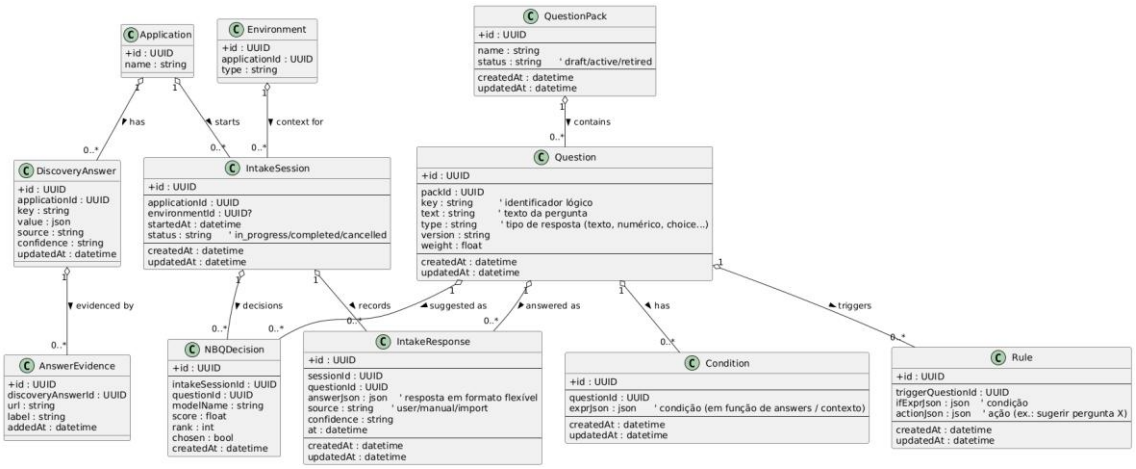
Figura 6 - Diagrama de Classes - *Core* AMS



O diagrama da **Figura 7** representa as classes que suportam o paradigma NBQ (*Next Best Question*), incluindo *QuestionPack, Question, Condition, Rule, IntakeSession e IntakeResponse*.
Demonstram-se também as relações entre estas classes e o modelo de *discovery* (*DiscoveryAnswer* e *AnswerEvidence*).

Este diagrama traduz a lógica usada durante os fluxos de *discovery* e recolha de respostas, permitindo que o motor NBQ e o motor de IA interajam de forma estruturada com a API.

Figura 7 - Diagrama de Classes - *Discovery* NBQ

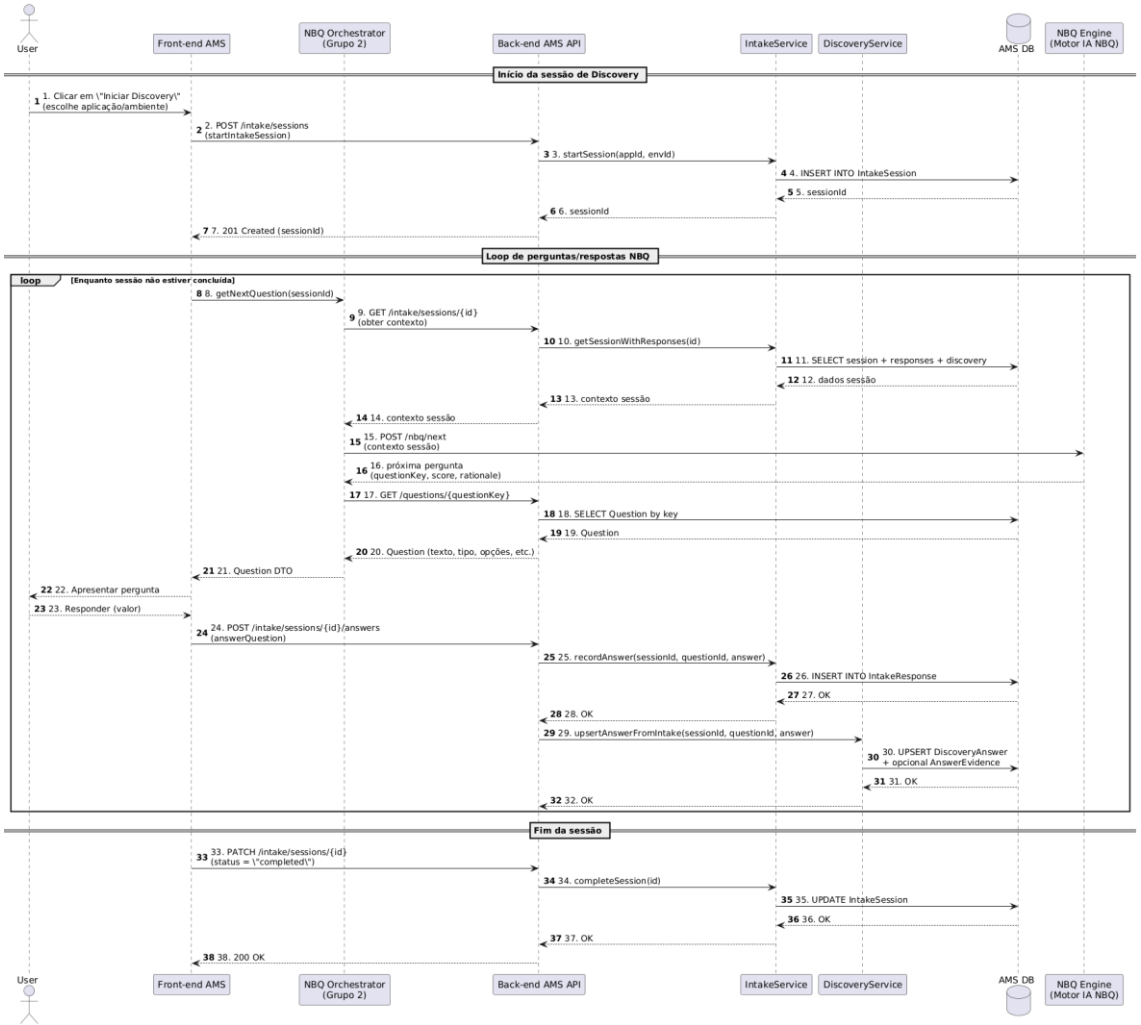


3.2.3 Diagrama de Sequência

Os diagramas de sequência descrevem a interação temporal entre o *front-end*, o *back-end* AMS, serviços externos, o motor NBQ e o motor de IA. Estes diagramas detalham os fluxos essenciais para a operação do sistema, nomeadamente a abertura e execução de sessões de *discovery*, a comunicação com o motor de IA para avaliação de risco e a ingestão automática de incidentes provenientes de sistemas ITSM.

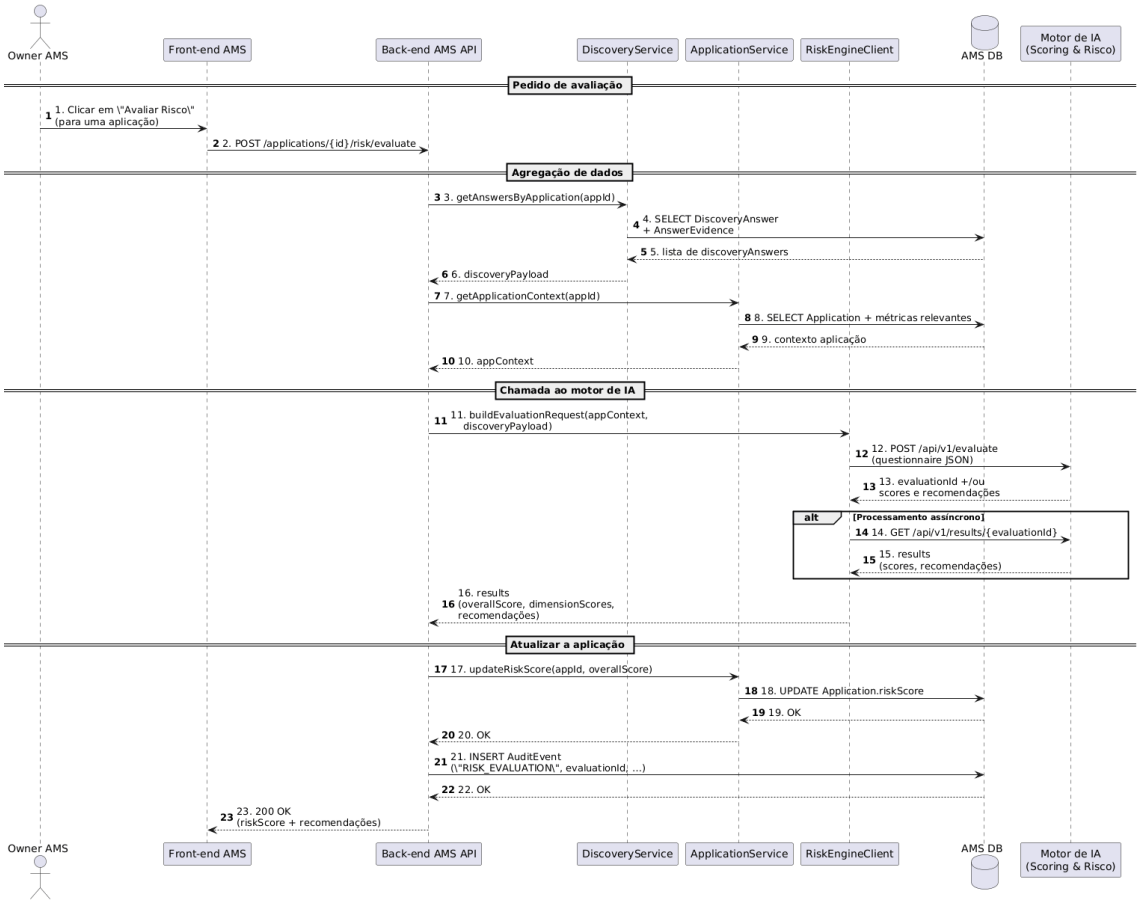
O diagrama da **Figura 8** modela o fluxo completo de uma sessão de *discovery* em que o utilizador responde a perguntas geradas dinamicamente pelo mecanismo NBQ (*Next Best Question*). O processo envolve vários componentes: o *front-end*, *o back-end* AMS, o orquestrador NBQ e o motor de IA responsável pela escolha da próxima pergunta mais relevante.

Figura 8 - Diagrama de Sequência - Sessão de *Discovery* com NBQ



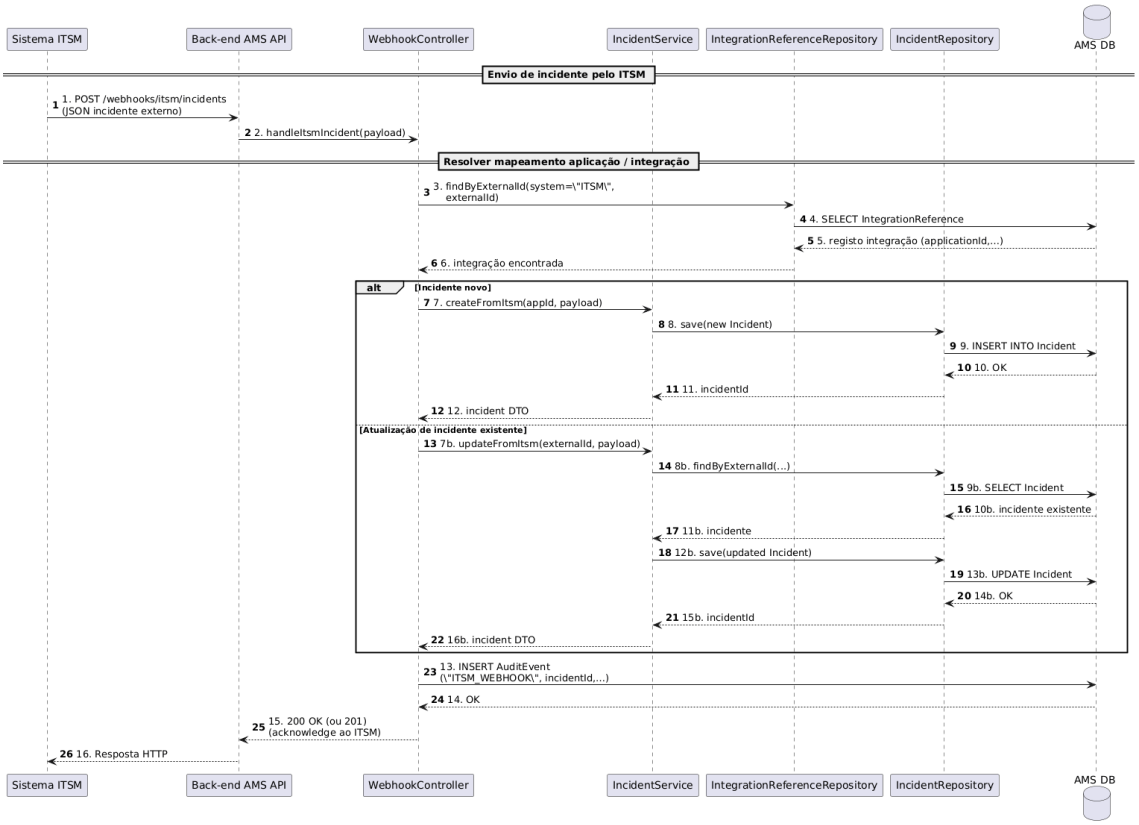
Já o diagrama da **Figura 9** representa o processo através do qual o *back-end* solicita ao motor de IA externo a avaliação do risco de uma aplicação AMS, tendo por base informação operacional e dados de *discovery* recolhidos previamente.

Figura 9 - Diagrama de Sequência - Avaliação de Risco com Motor de IA



O diagrama da **Figura 10** descreve o processo automatizado de ingestão de incidentes oriundos de sistemas ITSM externos (como ServiceNow ou Jira Service Management). Cumpre o requisito REQF_07 e permite que novos incidentes sejam registados sem intervenção manual.

Figura 10 - Diagrama de Sequência - Receção de Incidente via *Webhook* ITSM



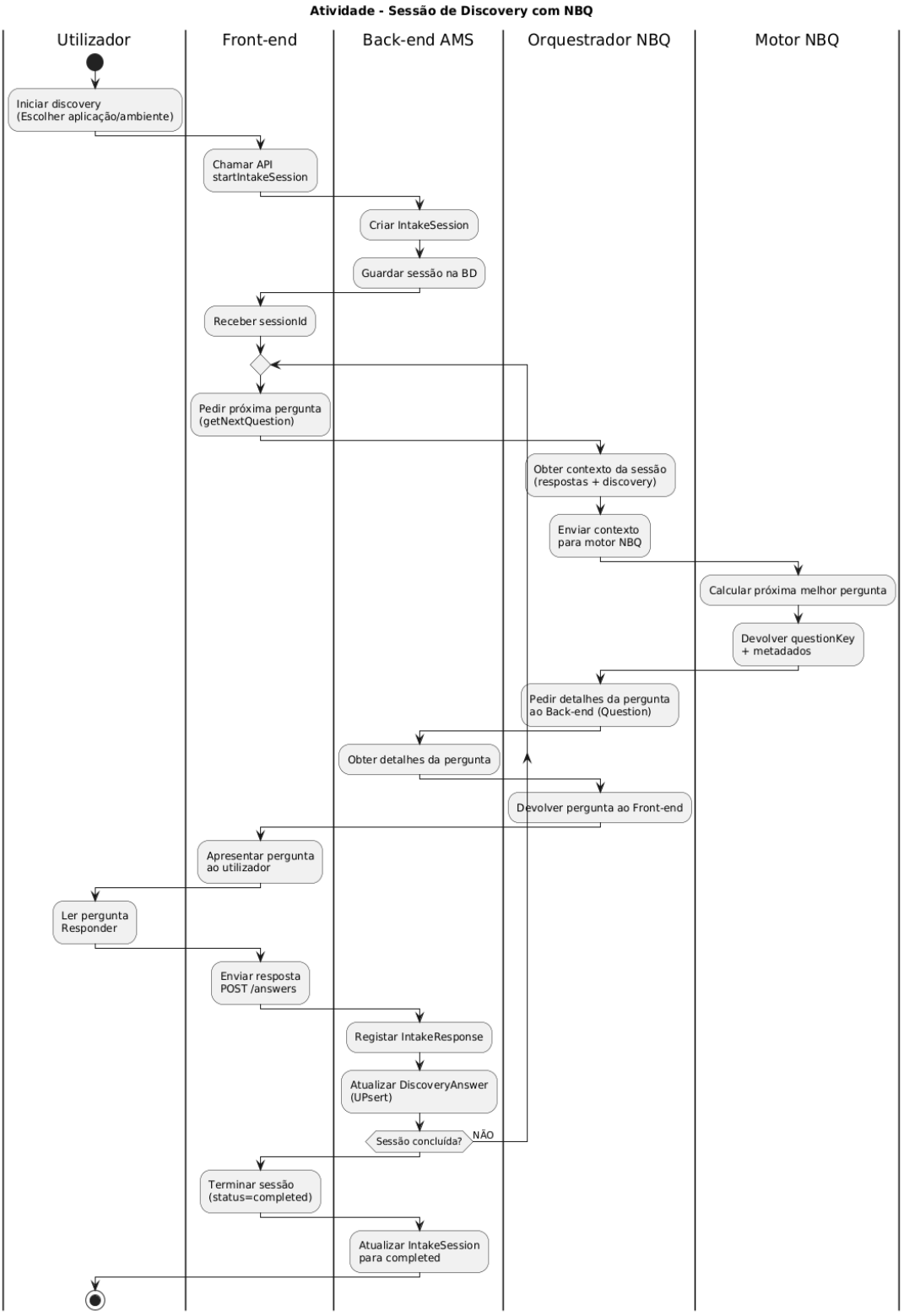
3.2.4 Diagramas de Atividade

Os diagramas de atividade representam o fluxo operacional das principais funcionalidades suportadas pelo *back-end* AMS, descrevendo de forma visual o conjunto de ações, decisões e interações necessárias para que cada processo seja executado. Diferentemente dos diagramas de sequência, que se focam na comunicação entre componentes, os diagramas de atividade modelam a lógica de processo, evidenciando etapas, regras de transição, ciclos e pontos de decisão.

No contexto da plataforma AMS, estes diagramas são particularmente relevantes para demonstrar a lógica por trás do *discovery* adaptativo baseado no paradigma *Next Best Question* (NBQ) e o processo de avaliação de risco suportado pelo motor de IA. Assim, asseguram uma compreensão clara e estruturada dos fluxos críticos que sustentam o *onboarding* e a análise operacional de aplicações.

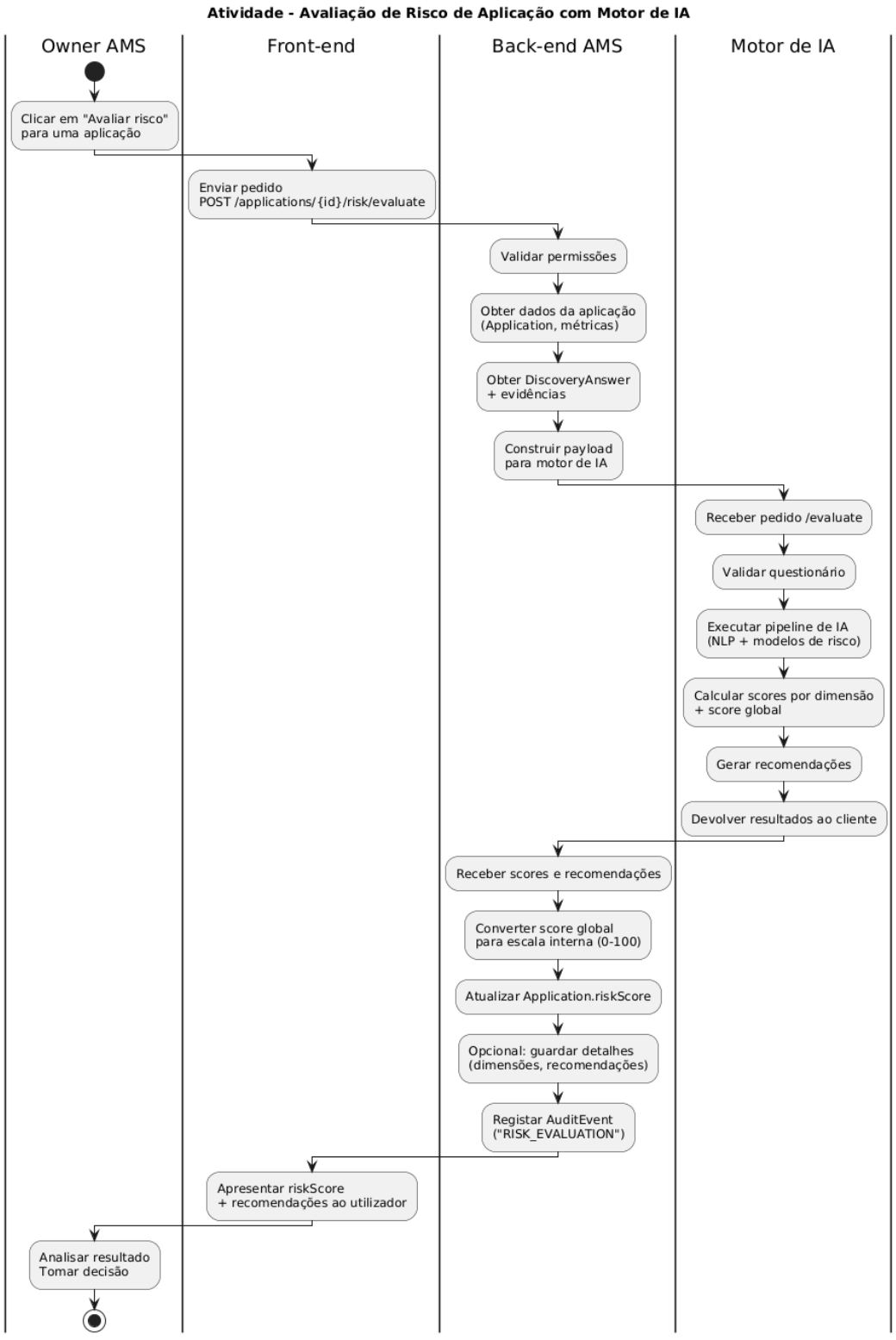
O diagrama da **Figura 11** descreve o fluxo completo associado ao processo de *discovery* adaptativo, onde o utilizador responde a um conjunto de perguntas selecionadas dinamicamente pelo motor NBQ. O processo inicia-se com a escolha da aplicação e do ambiente, seguida da criação de uma nova sessão de *discovery*.

Figura 11 - Diagrama de Atividade - *Discovery* + *Next Best Question* (NBQ)



A **Figura 12** representa o fluxo de avaliação de risco de uma aplicação AMS, processo que combina informação operacional com dados de *discovery* estruturado.

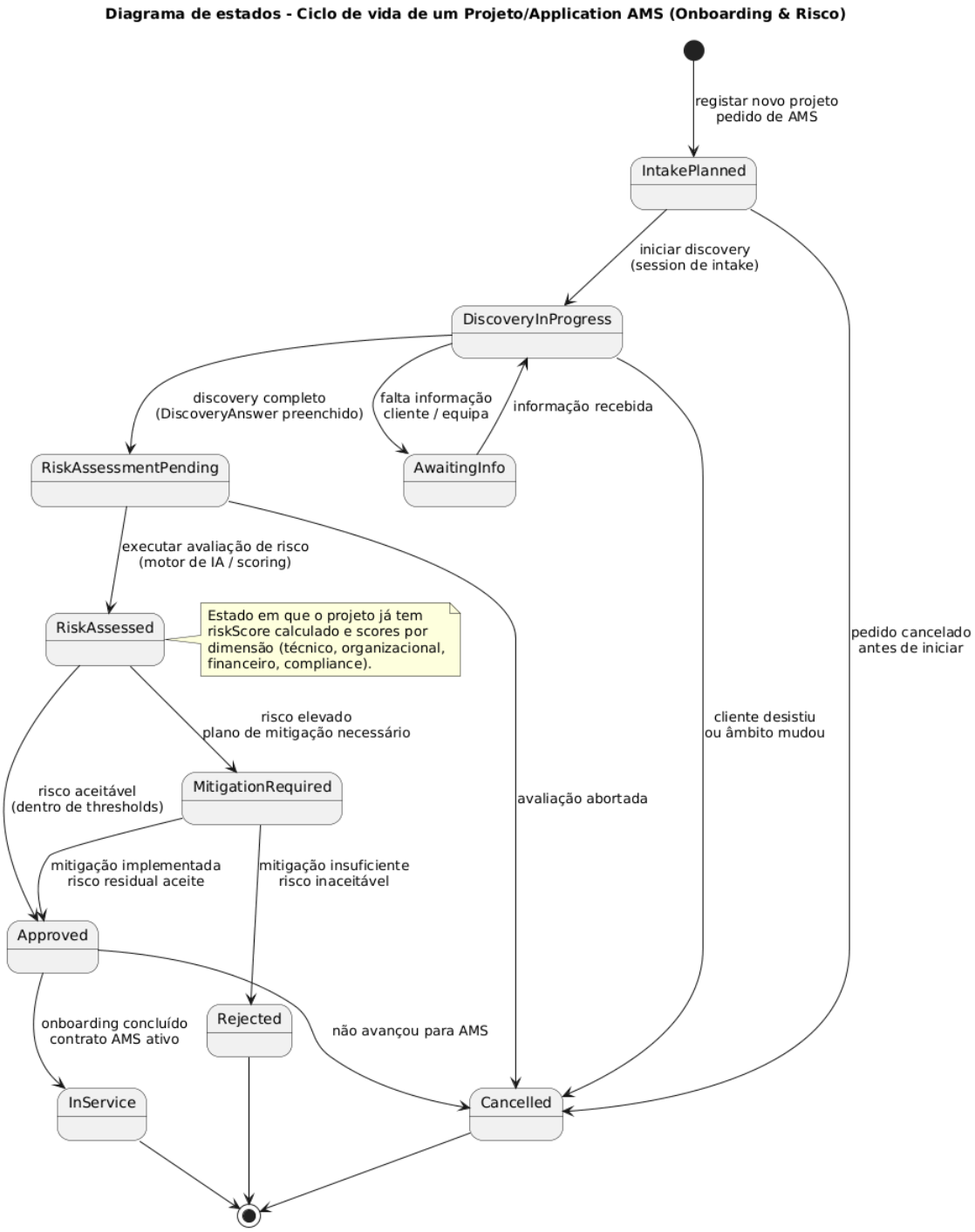
Figura 12 - Diagrama de Atividade - Avaliação de Risco com Motor de IA



3.2.5 Diagrama de Estados – Ciclo de Vida de Aplicação MAS

Este diagrama da **Figura 13** descreve os vários estados possíveis de uma aplicação AMS e as transições que ocorrem durante o processo de *onboarding*, *discovery* e avaliação de risco.

Figura 13 - Diagrama de Estados - Ciclo de Vida de Aplicação AMS



3.3 Protótipos de Interface

Uma vez que o *back-end* não dispõe de interface gráfica própria, o "mapa aplicacional" é representado através das estruturas de dados fornecidas ao *front-end*, que por sua vez constrói os ecrãs apresentados ao utilizador final. Assim, nesta secção são apresentados os **protótipos de interface lógica**, materializados através das respostas JSON devolvidas pela API, que correspondem diretamente ao conteúdo exibido nos diferentes módulos da aplicação.

Estes JSONs desempenham, portanto, o mesmo papel que *mockups* ou *wireframes* desempenhariam num projeto centrado no *front-end*, permitindo visualizar que informação é disponibilizada, como está organizada e como será utilizada para compor cada ecrã. Além disso, fornecem uma representação clara dos objetos que circulam entre a interface e o núcleo da plataforma AMS.

O mapa aplicacional apresentado nesta secção corresponde, assim, à navegação esperada pelos ecrãs do *front-end*, ilustrando os dados que cada um consome e os *endpoints* responsáveis pela sua disponibilização.

3.3.1 Lista de Aplicações (GET /applications)

Este JSON corresponde aos dados apresentados no ecrã de **Lista de Aplicações**, onde o utilizador pode consultar o catálogo de serviços sob gestão AMS. Cada entrada representa uma aplicação registada, incluindo o seu identificador, nome, criticidade e *owner*. O *front-end* utiliza esta resposta para construir tabelas, filtros e mecanismos de pesquisa.


```
[
  {
    "id": "app-001",
    "name": "Checkout Service",
    "criticality": "Gold",
    "owner": "payments-team"
  },
  {
    "id": "app-002",
    "name": "Inventory API",
    "criticality": "Silver",
    "owner": "ops-team"
  }
]
```

3.3.2 Detalhe da Aplicação (GET /applications/{id})

Este objeto representa os dados necessários ao ecrã de **Detalhe da Aplicação**. A partir dele, o *front-end* apresenta informações completas sobre a aplicação selecionada, incluindo:

- dados gerais (nome, criticidade, owners)
- contrato de serviço associado (SLA, on-call, idiomas)
- lista de ambientes existentes

Este JSON é fundamental para compor páginas de configuração, navegação entre separadores (contract, environments, discovery, KPIs) e ações administrativas.


```
{
  "id": "app-001",
  "name": "Checkout Service",
  "criticality": "Gold",
  "owners": ["payments-team"],
  "contract": {
    "sla": "99.9",
    "languages": ["PT", "EN"],
    "onCall": true
  },
  "environments": ["DEV", "TEST", "UAT", "PROD"]
}
```


3.3.3 Lista de Incidentes (GET /incidents?applicationId=app-001)

Este JSON alimenta o ecrã de **Incidentes da Aplicação**, apresentando os eventos operacionais registados para a aplicação selecionada. Permite ao *front-end* construir listas, aplicar filtros por severidade e estado, e mostrar informação essencial sobre cada ocorrência, tais como *timestamps* e níveis de severidade.


```
[
  {
    "id": "inc-120",
```



```
    "applicationId":  "app-001",
    "title":  "Timeout during checkout",
    "severity":  "SEV2",
    "status":  "Open",
    "openedAt":  "2025-11-10T09:12:00Z"
  }
]
```


3.3.4 Detalhe de Incidente (GET /incidents/{id})

Este JSON representa a estrutura de dados utilizada no ecrã de **Detalhe de Incidente**. Mostra informações completas sobre a ocorrência, incluindo:

- descrição técnica
- severidade
- timestamps chave: abertura, primeira resposta e fecho
- artigos de conhecimento associados

Esta informação permite ao operador analisar o incidente, consultar documentação relacionada e acompanhar o seu ciclo de vida.


```
{
 "id": "inc-120",
 "applicationId": "app-001",
 "title": "Timeout during checkout",
 "description": "High latency detected during payment step.",
 "severity": "SEV2",
 "openedAt": "2025-11-10T09:12:00Z",
 "firstResponseAt": "2025-11-10T09:14:32Z",
 "closedAt": null,
 "kbLinks": [
  { "kbId": "kb-305" }
 ]
}
```

3.3.5 Discovery (POST /discovery/{applicationId})

Este exemplo ilustra o *payload* usado pelo *front-end* para registar respostas no processo de *discovery*. Cada resposta representa um elemento de conhecimento recolhido sobre a aplicação, podendo incluir valores numéricos, booleanos ou estruturas compostas. O campo *questionRef* liga a resposta à pergunta NBQ apresentada ao operador.


```
{
  "key": "checkout_slos",
  "value": {
    "availability": "99.9",
    "latency_ms": 300
  },
  "questionRef": "checkout_slos@1.0.0",
  "updatedAt": "2025-11-10T10:02:00Z"
}
```


3.3.6 Exportação de Discovery (GET /export/{applicationId})

Este JSON é utilizado para compor o ecrã de **Discovery Consolidado**, onde o operador visualiza todas as respostas recolhidas para a aplicação. Inclui:

- chave (*key*)
- valor final consolidado
- referência da pergunta
- timestamp de última atualização

É também utilizado pelo motor de IA para scoring.


```
[
  {
    "key": "checkout_slos",
    "value": {
      "availability": "99.9",
      "latency_ms": 300
    },
    "questionRef": "checkout_slos@1.0.0",
    "updatedAt": "2025-11-10T10:02:00Z"
```



```
    },
    {
        "key": "monitoring_tools",
        "value": ["Grafana", "Elastic"],
        "updatedAt": "2025-11-10T09:50:00Z"
    }
]
```


3.3.7 KPIs da Aplicação (GET /kpi/{applicationId}?from&to)

Este JSON alimenta o ecrã de **Indicadores Operacionais**, apresentando métricas como MTTA, MTTR, número de incidentes e taxa de recorrência. O *front-end* utiliza-o para construir gráficos, tabelas de desempenho e *dashboards* de avaliação do serviço.


```
{
    "applicationId": "app-001",
    "mtta": 125,
    "mttr": 480,
    "recurrence": 0.10,
    "incidentCount": 34,
    "period": {
        "from": "2025-10-01",
        "to": "2025-10-31"
    }
}
```


3.3.8 Conclusão

Os exemplos apresentados permitem visualizar de forma clara o contrato entre a API do *back-end* e os ecrãs do *front-end*, demonstrando como a informação operacional é organizada, transmitida e utilizada para construir a interface da plataforma AMS. Estes protótipos de interface lógica funcionam como um mapa aplicacional orientado a dados, permitindo compreender os fluxos de navegação, a estrutura das entidades e a forma como cada módulo é abastecido pela camada de serviços. Esta representação assegura coerência entre requisitos, modelação, arquitetura e implementação, constituindo uma base sólida para o desenvolvimento das restantes camadas da solução.

 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.

4 Solução Proposta

4.1 Apresentação

A solução proposta corresponde ao núcleo *back-end* da plataforma de *onboarding* de projetos AMS, concebido para centralizar informação operacional, normalizar processos e suportar decisões durante a transição de um novo serviço para operação. Funcionalmente, a solução permite gerir aplicações, contratos de serviço, ambientes, incidentes, problemas, mudanças*, releases*, artigos de conhecimento, *runbooks*, integrações externas, e recolher dados estruturados de *discovery* através do **paradigma *Next Best Question* (NBQ)**. Para além destas funcionalidades *core*, o sistema inclui mecanismos de consolidação de conhecimento, cálculo de métricas (MTTA, MTTR, recorrência), auditoria completa das operações e integração com um motor de IA responsável pela avaliação de risco da aplicação.

Quando comparada com as soluções analisadas no benchmarking, esta solução distingue-se por integrar, desde a base, os processos específicos de ***onboarding* AMS**, os **fluxos de *discovery* orientados a NBQ** e a **avaliação automática de risco**. Enquanto as outras plataformas permitem operar serviços já estabilizados, a proposta aqui apresentada está desenhada especificamente para o contexto AMS da CGI, com foco na recolha estruturada de informação, normalização dos dados e suporte aos restantes grupos do projeto (interface inteligente e motor de IA). Em termos funcionais, a solução cobre um conjunto de capacidades equivalentes às encontradas nos produtos de mercado, mas introduz maior flexibilidade no modelo de dados, integração nativa com algoritmos de risco e um fluxo de *intake* mais orientado à eficiência.

De forma a garantir uma compreensão clara e estruturada da solução proposta, este capítulo encontra-se organizado da seguinte forma:

- **Arquitetura** - apresenta a estrutura geral da solução, os seus componentes principais e as opções que orientaram a definição do modelo arquitetural, incluindo o papel central do *back-end* AMS e a sua articulação com sistemas externos.
- **Integração com o Motor de IA** - descreve o modo como o back-end comunica com o serviço inteligente responsável pela avaliação de risco, recomendações e suporte ao paradigma *Next Best Question* (NBQ), detalhando objetivos, fluxos, endpoints e garantias de segurança.
- **Tecnologias e Ferramentas Utilizadas** - identifica as tecnologias adotadas no desenvolvimento, bem como as ferramentas de apoio utilizadas durante o projeto.
- **Ambientes de Teste e de Produção** - apresenta o ambiente previsto para execução da solução, bem como configurações relevantes para a sua operação.
- **Abrangência** - relaciona as áreas científicas e unidades curriculares envolvidas na construção da solução.

- **Componentes** - detalha os componentes principais da implementação, destacando aspetos técnicos relevantes.
- **Interfaces** - apresenta o mapa aplicacional e os ecrãs representativos da solução, acompanhados de uma breve explicação do seu funcionamento.

4.2 Arquitetura

A arquitetura do *back-end* AMS foi concebida para garantir **modularidade**, **escalabilidade**, **baixo acoplamento**, **segurança**, **observabilidade** e **integridade dos dados**, assegurando que a solução suporta tanto a operação diária AMS como os mecanismos de *scoring* e NBQ dos restantes grupos.

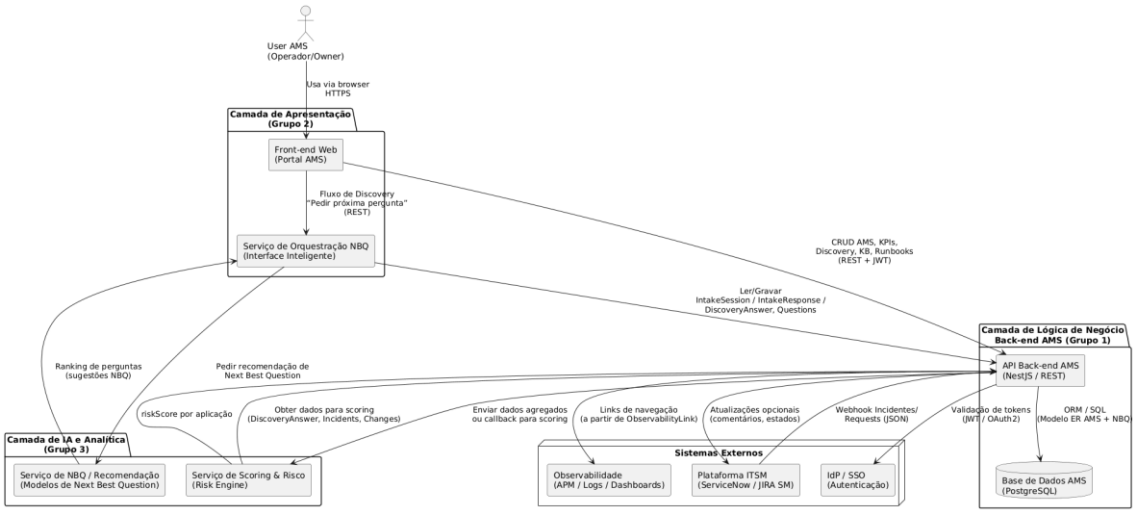
A solução segue um modelo arquitetural **em camadas (*layered architecture*)**, amplamente utilizado em aplicações empresariais, favorecendo a organização de responsabilidades e permitindo evolução futura sem comprometer o núcleo de dados. O desenho da arquitetura e dos seus componentes está representado nas figuras incluídas no relatório (arquitetura lógica, arquitetura por camadas e diagrama de pacotes).

4.2.1 Arquitetura Lógica da Solução

A arquitetura lógica da solução, expressa na **Figura 14,** foi concebida para representar, de forma clara e de alto nível, os principais blocos funcionais que compõem o ecossistema da plataforma AMS e as interações essenciais entre eles. Esta visão estrutural destaca não apenas o papel central do *back-end* AMS, mas também a forma como este se articula com os restantes elementos do sistema, assegurando um funcionamento coeso, escalável e orientado à colaboração entre módulos.

No centro da arquitetura encontra-se o *back-end* AMS, responsável pela gestão dos dados operacionais, organização de fluxos e disponibilização de funcionalidades essenciais ao *onboarding* e operação de aplicações. Sobre este núcleo assenta uma camada de apresentação composta pela interface *web* e pelo serviço de organização NBQ, que interagem com o *back-end* para recolher e apresentar informação ao utilizador. Paralelamente, o sistema comunica com serviços externos especializados, nomeadamente o motor de IA (para avaliação de risco e recomendação de perguntas) e as plataformas ITSM, que enviam incidentes e pedidos através de integrações específicas. Por fim, a base de dados relacional suporta de forma consistente e auditável toda a informação gerida pelo *back-end*. Esta organização modular garante separação clara de responsabilidades e uma arquitetura flexível, preparada para evoluções futuras.

Figura 14 - Arquitetura lógica da solução



4.2.2 Arquitetura por Camadas

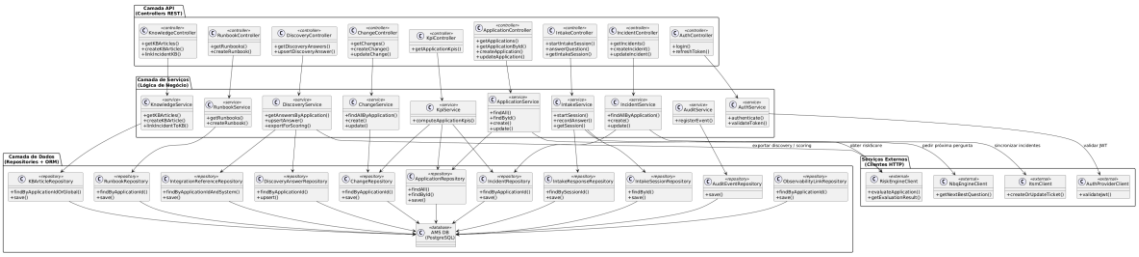
A arquitetura por camadas adotada no *back-end* AMS segue um modelo estruturado que organiza o sistema em unidades funcionais bem definidas, facilitando a manutenção, a evolução e a testabilidade da solução. Este padrão permite separar responsabilidades, reduzir acoplamento entre componentes e assegurar que cada camada exerce um papel distinto e coerente dentro do fluxo de processamento da aplicação. A **Figura 15** apresenta esta organização, evidenciando o fluxo lógico de comunicação entre as camadas internas e os serviços externos que complementam o funcionamento do sistema.

A camada superior corresponde à **API REST**, onde são disponibilizados os *endpoints* necessários ao funcionamento da plataforma. Estes controladores são responsáveis pela receção de pedidos, validação inicial dos dados, aplicação de políticas de autenticação e autorização, e encaminhamento das operações para os serviços adequados. Logo abaixo encontra-se a **camada de serviços**, que implementa a lógica de negócio associada à gestão das entidades operacionais, cálculo de métricas, execução dos fluxos de *discovery* e integração com sistemas externos, como o motor de IA para avaliação de risco ou o serviço NBQ para apoio à tomada de decisão. Esta camada funciona como núcleo funcional da aplicação, coordenando toda a lógica que suporta o domínio AMS.

A interação com a base de dados é realizada pela **camada de repositórios**, que encapsula o acesso ao modelo relacional e oferece operações especializadas para consulta, criação, atualização e remoção de dados. Esta camada assegura o isolamento entre as regras de negócio e a persistência, garantindo consistência e integridade através de mecanismos como transações e validações aplicadas no domínio dos dados. Por sua vez, a **base de dados relacional** representa o ponto final de persistência, onde se encontra o modelo normalizado que suporta todas as entidades consideradas no sistema. Para além das camadas internas, a arquitetura inclui ainda os **serviços externos**, que fornecem funcionalidades complementares como a avaliação automática de risco, a sugestão de perguntas NBQ ou a ingestão de incidentes provenientes de plataformas ITSM. A comunicação entre estes

componentes é realizada de forma desacoplada, através de clientes especializados que garantem robustez, tratamento de falhas e integração segura.

Figura 15 - Arquitetura lógica do *back-end* AMS organizada em camadas *controller/service/repository*



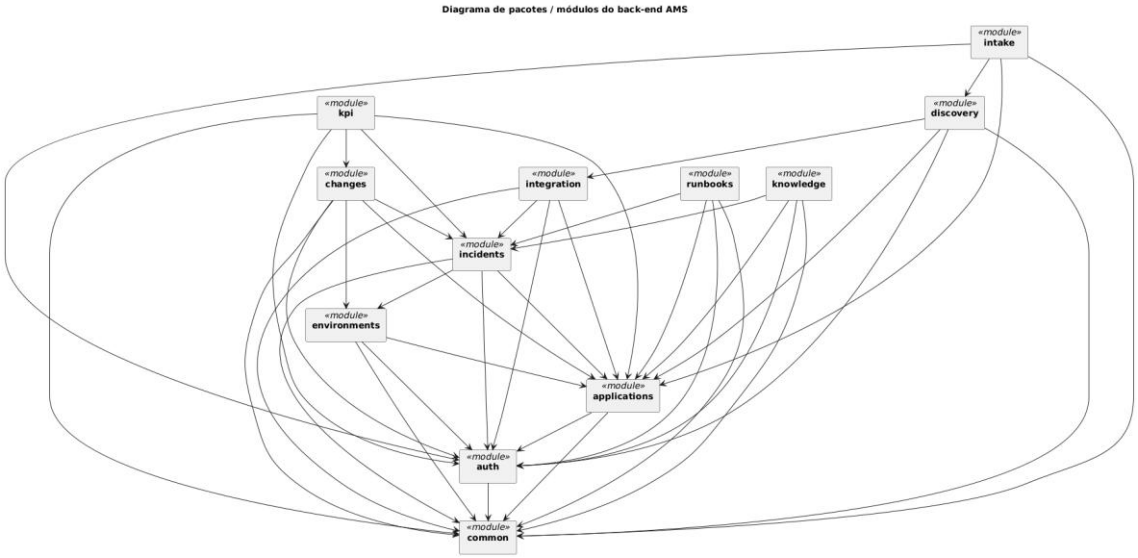
4.2.3 Diagrama de Pacotes

O diagrama de pacotes organiza a estrutura interna do código do back-end AMS em módulos lógicos, permitindo visualizar de forma clara como as diferentes áreas funcionais do sistema se encontram agrupadas e como estas se relacionam entre si. Esta representação é particularmente importante para compreender a modularidade da solução, facilitando a manutenção, o desenvolvimento incremental e a colaboração entre equipas. A **Figura 16** apresenta esta organização, destacando os principais pacotes que compõem o sistema e a forma como estes se articulam para suportar o domínio AMS.

A estrutura encontra-se dividida em pacotes que refletem diretamente as responsabilidades do sistema. Os módulos ***application***, ***contract*** e ***environment*** concentram a gestão das entidades base do modelo AMS, enquanto módulos como ***incident***, ***problem***, ***change*** e ***release*** representam os processos operacionais associados à gestão de serviços. A vertente de conhecimento é suportada pelos pacotes ***kb*** e ***runbook***, onde se encontram os componentes relacionados com artigos de conhecimento e guias operacionais. O subsistema de *discovery* e NBQ encontra-se igualmente representado através dos pacotes ***questionnaire***, ***nbq***, ***intake*** e ***discovery***, que incluem o catálogo de perguntas, a gestão de sessões de *intake* e a consolidação de respostas.

Para além dos pacotes funcionais, o diagrama inclui ainda módulos transversais como ***security***, responsável pela autenticação e controlo de acessos, ***audit***, que gere o registo de operações relevantes, e ***integration***, onde são implementadas as ligações aos serviços externos, incluindo o motor de IA, NBQ e plataformas ITSM. Esta organização modular assegura uma separação equilibrada entre componentes, permitindo que cada área funcional evolua de forma independente, sem comprometer a coerência global do sistema. A divisão em pacotes contribui igualmente para uma melhor navegabilidade do código, maior testabilidade e uma estrutura escalável adequada ao crescimento futuro da plataforma AMS.

Figura 16 - Diagrama de Pacotes/Módulos do *back-end* AMS



4.3 Integração com o motor de IA

A plataforma AMS integra-se com um motor de inteligência artificial externo responsável pela Avaliação de Continuidade e Gestão da Mudança em projetos AMS. Esta integração permite ao sistema incorporar capacidades avançadas de classificação de risco, análise contextual e geração de recomendações, enriquecendo o processo de *onboarding* e avaliação de maturidade das aplicações. O motor de IA é tratado como um serviço independente, comunicando com o *back-end* AMS através de chamadas REST e mantendo total desacoplamento entre sistemas. O *back-end* AMS assume o papel de cliente deste serviço, preparando os dados necessários, efetuando chamadas aos endpoints apropriados e armazenando os resultados de forma consistente no modelo interno.

4.3.1 Objetivos da Integração

A integração com o motor de IA serve três objetivos principais:

1. Avaliação automática de risco por aplicação AMS

O *back-end* AMS deve ser capaz de solicitar ao motor de IA uma avaliação de risco para cada aplicação, utilizando as respostas recolhidas durante o *discovery* e outra informação operacional disponível. O resultado desta avaliação alimenta o campo *riskScore* da entidade ***Application***, apoiando a identificação de fatores críticos e potenciais fragilidades.

2. Recolha de recomendações contextuais

Para além de um *score* global, o motor de IA gera recomendações organizadas por prioridade, permitindo apoiar equipas AMS em decisões relacionadas com *onboarding*, revisão de contratos ou definição de ações mitigadoras. Estas

recomendações podem ser apresentadas no *front-end*, mas são sempre consumidas a partir da API interna do *back-end* AMS.

3. Base para integração futura com *Next Best Question* (NBQ)

A arquitetura prepara ainda o suporte à funcionalidade NBQ, permitindo que o motor de IA determine, a cada iteração de *discovery*, qual a próxima pergunta mais informativa. Esta funcionalidade complementará o subsistema já existente para gestão de questionários e sessões de *intake*, reforçando o diagnóstico adaptativo.

4.3.2 Endpoints e Fluxo de Avaliação de Risco

O motor de IA disponibiliza um conjunto de endpoints REST bem definidos, utilizados pelo *back-end* AMS para submissão e consulta de avaliações:

- **POST /api/v1/evaluate** - Submissão de dados de avaliação.
- **GET /api/v1/results/{evaluation_id}** - Consulta de resultados já processados.
- **GET /api/v1/health** - Verificação de disponibilidade do motor de IA.
- **GET /api/v1/policy/version** - Consulta da versão ativa da política de *scoring*.

O processo de integração segue quatro etapas principais:

1) Preparação do *payload*

O *back-end* agrega informações provenientes do modelo interno, incluindo:

- dados da aplicação,
- respostas de *discovery* (tabela ***DiscoveryAnswer***),
- informação operacional relevante (volumetria, criticidade, histórico, etc.).

Este conjunto é transformado num JSON compatível com o formato esperado pelo *endpoint* de avaliação.

2) Submissão de avaliação

O *endpoint* **POST /api/v1/evaluate** é invocado com autenticação via API *key*. O motor pode responder:

- **200 OK** - resultados imediatos
- **202 Accepted** - processamento assíncrono, devolvendo evaluation_id

3) Consulta de resultados

Quando o processamento é assíncrono, o *back-end* consulta periodicamente **GET /api/v1/results/{evaluation_id}** até obter:

- *score* global
- *scores* por dimensão

- lista de recomendações
- metadados (*timestamp*, versão do modelo)

4) Atualização do modelo interno

Após receber os resultados, o *back-end*:

- converte o *score* global para a escala interna (0–100),
- atualiza *Application.riskScore*,
- opcionalmente persiste detalhes adicionais,
- regista um evento de auditoria (***AuditEvent***) associado à avaliação.

Toda a complexidade da comunicação com o motor de IA é abstraída da interface, garantindo que o *front-end* apenas consome a informação já processada pelo *back-end*.

4.3.3 Integração com o Submodelo NBQ

A integração com NBQ representa uma extensão evolutiva na qual o motor de IA pode selecionar a próxima pergunta mais informativa com base nas respostas já existentes. O *endpoint* previsto (**POST /nbq/next**) recebe:

- o contexto da sessão (respostas de *IntakeResponse*),
- conhecimento consolidado (*DiscoveryAnswer*),
- metadados da aplicação.

O fluxo esperado será o seguinte:

- O *back-end* continua a ser responsável pela gestão de ***QuestionPack, Question, IntakeSession e IntakeResponse.***
- Um componente de orquestração recolhe o contexto e invoca o endpoint **/nbq/next**.
- O motor devolve uma identificação lógica da pergunta recomendada e metadados.
- O *back-end* mapeia essa identificação para uma pergunta concreta do catálogo e devolve ao *front-end*.
- O ciclo repete-se a cada nova resposta do utilizador.

Este mecanismo garante separação de responsabilidades:

- o **motor de IA** decide qual pergunta é mais informativa,
- o ***back-end* AMS** decide como essa pergunta é representada e persistida,
- o ***front-end*** decide como a pergunta é apresentada.

4.3.4 Considerações de Segurança e Robustez

A integração respeita os requisitos não funcionais definidos pelo motor de IA, incluindo **tempo de resposta, concorrência, disponibilidade, documentação da API** e **autenticação por API key.**

Do lado do *back-end* AMS:

- As chamadas ao motor de IA são encapsuladas em **serviços específicos de integração**, isolando credenciais (*API keys*) e lógica de *retry* em caso de falhas temporárias.
- *Logs* de integração são registados com o mínimo de dados sensíveis, cumprindo as políticas de segurança e facilitando o diagnóstico de problemas.
- A invocação do endpoint *GET /api/v1/health* pode ser utilizada em tarefas agendadas de monitorização, permitindo ao *back-end* AMS detetar antecipadamente indisponibilidade do motor de IA e reagir de forma adequada (por exemplo, bloqueando temporariamente a avaliação automática e alertando os utilizadores).

Em síntese, a integração com o motor de IA do grupo DEISI2159 transforma o *back-end* AMS numa **plataforma habilitada para IA**, capaz de combinar dados operacionais e de *discovery* com modelos avançados de NLP e classificação de risco, mantendo ao mesmo tempo uma fronteira clara entre responsabilidades de cada grupo e uma API bem definida para evolução futura.

4.4 Tecnologias e Ferramentas Utilizadas

A solução será desenvolvida em **Java**, utilizando a framework **Spring Boot**, pela sua simplicidade na criação de APIs REST e pela boa integração com o ecossistema Java. A gestão de dependências será realizada com **Maven**, garantindo organização e consistência no projeto. Para armazenamento e gestão dos dados, será utilizada uma base de dados relacional **SQL Server**, adequada ao modelo entidade-relação definido.No desenvolvimento, recorreremos ao **Visual Studio Code** como IDE, devido à sua integração nativa com Spring e facilidade de utilização. Todo o projeto é gerido com Git, garantindo organização, controlo de alterações e colaboração eficiente.

4.5 Ambientes de Teste e de Produção

A solução foi preparada para funcionar em dois ambientes distintos - **Ambiente de Teste** e **Ambiente de Produção** - garantindo isolamento, estabilidade e segurança ao longo das fases de desenvolvimento, validação e operação da plataforma AMS.

4.5.1 Ambiente de Teste

O ambiente de teste é utilizado durante o desenvolvimento, permitindo iteração rápida, execução local e validação contínua das funcionalidades implementadas. Este ambiente privilegia flexibilidade e reprodutibilidade, sendo adequado para testes funcionais, de integração e simulação de fluxos externos.

As aplicações são executadas numa máquina local ou em *containers*, com configurações típicas de **2 vCPUs**, **4–8 GB de RAM**, cerca de **500 MB** para artefactos de build e logs, e rede local. O *back-end* corre em **Java 17+** com **Spring Boot**, podendo ser executado diretamente ou via **Podman**, enquanto a base de dados utiliza uma instância leve de **SQL Server** em ambiente local ou container. Serviços externos são simulados através de ferramentas como Postman ou endpoints mockados, permitindo testar integrações com ITSM e com o motor de IA sem dependência de infraestrutura real.

Este ambiente utiliza dados fictícios e logging detalhado, facilitando o diagnóstico e a repetição de testes, assegurando que todas as funcionalidades operam corretamente antes da passagem para produção.

4.5.2 Ambiente de Produção

O ambiente de produção destina-se à execução estável e segura da plataforma em contexto empresarial, garantindo disponibilidade e integração com sistemas externos reais. É recomendado o uso de uma máquina virtual ou cluster Kubernetes com **2–4 vCPUs**, **8 GB de RAM** e armazenamento mínimo de **10 GB** para a base de dados e **2 GB** para logs com rotação configurada.

O *back-end* é distribuído em *containers* **Docker**, com possibilidade de escalabilidade horizontal. O perfil de produção do Spring Boot, juntamente com mecanismos de observabilidade, garante monitorização eficiente. A base de dados utiliza **SQL Server** ou um motor relacional equivalente, com *backups* automáticos e configuração de retenção. A rede deve estar protegida por TLS, *firewalls* e políticas de acesso restritas a IPs internos, com largura de banda mínima de **100 Mbps** entre API e base de dados.

A integração com sistemas ITSM (como ServiceNow ou Jira SM), bem como com o motor de IA da avaliação de risco, ocorre de forma segura através de endpoints protegidos. Entre os requisitos de operação incluem-se disponibilidade superior a **99%**, tempos de resposta otimizados, *health checks*, *logging* estruturado e validação JWT, garantindo conformidade com as necessidades de um ambiente AMS real.

4.6 Abrangência

O desenvolvimento deste Trabalho Final de Curso integrou conhecimentos adquiridos ao longo da licenciatura, combinando várias áreas técnicas e metodológicas essenciais para a construção do núcleo *back-end* da plataforma de onboarding de projetos AMS. Seguem-se as principais unidades curriculares envolvidas e a forma como contribuíram para o projeto:

- **Linguagens de Programação II**

Forneceu os fundamentos essenciais da programação em Java, nomeadamente encapsulamento, classes, interfaces, polimorfismo e boas práticas de engenharia de software. Estes conceitos foram aplicados na implementação das entidades de domínio (Application, Incident, Change, Runbook, DiscoveryAnswer, etc.) e na estrutura geral dos serviços do *back-end*.

- **Algoritmia e Estruturas de Dados**

Os conhecimentos de algoritmos de pesquisa e ordenação foram cruciais para implementar lógica interna de filtragem, validação, cálculo de KPIs e manipulação de dados recebidos pelas APIs.

- **Bases de Dados**

Contribuíram diretamente para o desenho do modelo entidade-relação (ER), normalização das tabelas, definição de *keys* e integridade referencial. Estes conhecimentos foram aplicados na criação do esquema relacional que suporta todas as entidades do sistema AMS.

- **Engenharia de Software**

Abordou metodologias de desenvolvimento, processos de engenharia, métodos ágeis, análise de requisitos e qualidade de software. Estes conhecimentos foram aplicados na elaboração do *backlog*, organização do planeamento, definição dos requisitos REQF_01-REQF_18, estruturação das user stories e planeamento dos testes.

- **Computação Distribuída**

Forneceu os fundamentos necessários para a comunicação entre sistemas, APIs REST, arquiteturas distribuídas, padrões de comunicação, camadas *service/repository* e integração de componentes externos. Estes conhecimentos foram essenciais para estruturar a API do *back-end*, separar responsabilidades entre controladores, serviços e repositórios, e garantir a comunicação com o motor de IA e serviços NBQ.

4.7 Componentes

Esta secção descreve os principais componentes internos que constituem o *back-end* da plataforma AMS. A solução foi desenvolvida seguindo uma arquitetura modular organizada por camadas, garantindo baixo acoplamento, elevada coesão e facilidade de manutenção. Cada componente desempenha um papel específico no processamento das operações, desde a receção de pedidos até ao acesso aos dados persistidos, assegurando que toda a lógica de negócio é executada de forma consistente, auditável e segura.

A divisão por componentes também facilita a escalabilidade e a evolução do sistema, permitindo que funcionalidades adicionais sejam incorporadas sem comprometer a estrutura existente. Seguem-se os principais blocos lógicos do *back-end* e as suas responsabilidades.

4.7.1 API Layer (REST *Controllers*)

Este componente representa o ponto de entrada de todos os pedidos externos. Cada controlador expõe um conjunto de endpoints REST responsáveis por operações específicas do domínio AMS.

Principais controladores:

- *ApplicationsController* - /applications
- *IncidentsController* - /incidents
- *ChangesController* - /changes
- *KnowledgeController* - /kb, /runbook
- *DiscoveryController* - /discovery/{applicationId}, /export/{applicationId}
- *KpiController* - /kpi/{applicationId}

Responsabilidades técnicas:

- Receber e interpretar pedidos HTTP.
- Validar parâmetros e estruturas básicas do payload.
- Aplicar políticas de autenticação e autorização.
- Encaminhar pedidos para os serviços de domínio.
- Devolver respostas JSON com códigos HTTP adequados.

Os controladores não contêm lógica de negócio; funcionam como intermediários entre o cliente e as camadas internas.

4.7.2 Domain Services (*Core Business Logic*)

A camada de serviços implementa toda a lógica de negócio da solução, sendo responsável por aplicar regras, garantir consistência e organizar interações entre entidades e repositórios.

Principais serviços:

- *ApplicationService*
- *IncidentService*
- *ProblemService*
- *ChangeService*
- *ReleaseService*
- *KnowledgeService*
- *RunbookService*
- *DiscoveryService*
- *KpiService*

Responsabilidades técnicas:

- Aplicar validações complexas (ex.: limites temporais, estados permitidos).
- Implementar regras de negócio descritas nos requisitos funcionais.
- Coordenar múltiplos repositórios numa mesma operação.
- Gerir transações e garantir consistência no domínio.
- Executar cálculos de métricas operacionais (MTTA, MTTR, recorrência).

Esta camada é independente da tecnologia HTTP ou da base de dados, tornando o sistema flexível e testável.

4.7.3 Persistence Layer (Repositories)

A camada de persistência encapsula o acesso à base de dados relacional. Os repositórios fornecem métodos especializados para consulta, criação, atualização e remoção de entidades.

Principais repositórios:

- *ApplicationRepository*
- *IncidentRepository*
- *ChangeRepository*
- *ReleaseRepository*

- *KbRepository*
- *RunbookRepository*
- *DiscoveryRepository*
- *KpiRepository*

Responsabilidades técnicas:

- Executar queries otimizadas ao modelo relacional.
- Suportar paginação, filtros e ordenações.
- Assegurar integridade referencial.
- Encapsular toda a lógica de acesso a dados, isolando-a da lógica de negócio.

4.7.4 Security / Authentication Component

Este componente implementa os mecanismos de autenticação e controlo de acessos, garantindo que apenas utilizadores autorizados interagem com a API.

Responsabilidades técnicas:

- Validar tokens JWT antes da execução dos controladores.
- Aplicar políticas de RBAC, distinguindo perfis como *viewer*, *operator* e *owner*.
- Bloquear operações não permitidas (403 *Forbidden*).
- Garantir proteção contra pedidos não autenticados (401 *Unauthorized*).

Este módulo atua como *middleware* transversal a todas as operações.

4.7.5 Observability Component (Logs e Métricas)

A observabilidade é um componente transversal responsável por registar a atividade e desempenho do sistema.

Responsabilidades técnicas:

- Produzir *logs* estruturados com: *timestamp*, *requestId*, rota, *status*, *latencyMs*.
- Expor métricas relevantes por *endpoint*.
- Suportar a monitorização e diagnóstico de incidentes.
- Integrar com futuros sistemas de observabilidade (APM, *dashboards*).

4.7.6 Componentes Internos

A **Figura 17** apresenta o diagrama de componentes internos do *back-end*, sintetizando a organização modular da solução e o fluxo de comunicação entre as diferentes camadas que constituem o núcleo operacional da plataforma AMS. O diagrama evidencia claramente a separação de responsabilidades, o desacoplamento entre camadas e a forma como os diferentes blocos colaboram para processar cada pedido recebido pela API.

Na parte superior encontra-se a interação com os clientes externos(nomeadamente a interface da aplicação e os serviços inteligentes) que comunicam com o sistema através de pedidos HTTP autenticados via JWT. Cada pedido recebido passa, em primeiro lugar, pelo filtro de autenticação e autorização, responsável por validar o token e assegurar que o utilizador possui as permissões necessárias para executar a operação pretendida.

Após validação, o pedido é encaminhado para o componente **API Layer**, onde se encontram os diversos controladores responsáveis por expor os *endpoints* públicos: *ApplicationsController, IncidentsController, ChangesController, DiscoveryController, KnowledgeController e KpiController*. Estes controladores realizam validações iniciais, interpretam os parâmetros recebidos e direcionam o pedido para o serviço correspondente.

A camada seguinte, **Domain Services**, implementa a lógica de negócio associada ao domínio AMS. Cada serviço aplica regras específicas, garante consistência entre entidades, coordena operações entre múltiplos módulos e executa cálculos como KPIs ou verificações de estado. Nesta camada encontram-se serviços como *ApplicationService*, *IncidentService*, *ChangeService, DiscoveryService, KnowledgeService e KpiService*.

Em seguida, os serviços de domínio comunicam com a **Persistence Layer**, que agrega os repositórios responsáveis pelo acesso à base de dados. Cada repositório encapsula operações CRUD, queries específicas, filtros e paginação sobre as entidades principais do sistema, tais como aplicações, incidentes, mudanças, artigos de conhecimento, *runbooks*, *discovery* e métricas. Estes repositórios, por sua vez, interagem diretamente com a base de dados operacional, garantindo persistência e integridade dos dados.

Ao longo de todas as camadas atua o componente transversal de ***Logging* e Métricas**, responsável por registar informação estruturada sobre cada pedido (*timestamp*, *requestId,* rota, latência, estado) e expor métricas relevantes ao funcionamento e monitorização do sistema.

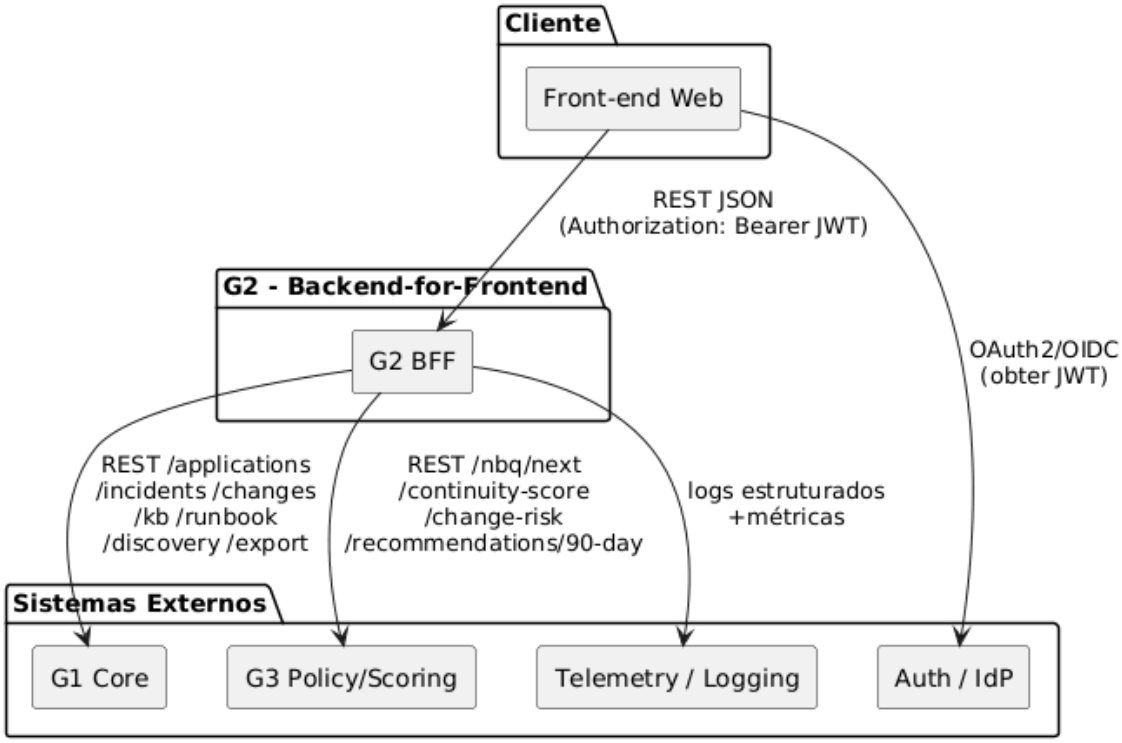
De forma geral, o diagrama ilustra o fluxo típico de uma operação:

1. O cliente envia um pedido HTTP com token JWT.
2. O filtro de autenticação valida o acesso.
3. O pedido é encaminhado para o controlador responsável.
4. O controlador invoca o serviço correspondente.

5. O serviço interage com os repositórios.
6. Os repositórios acedem à base de dados operacional.
7. *Logs* e métricas são registados em todas as etapas.

Este modelo modular garante organização clara, elevada testabilidade, facilidade de manutenção e uma arquitetura escalável preparada para a evolução da plataforma.

Figura 17 - Diagrama de Componentes Internos do *Back-end*



4.8 Interfaces

A camada de interfaces do *back-end* corresponde à API pública responsável por expor toda a informação operacional da plataforma AMS. Esta API constitui o ponto central de comunicação com a interface desenvolvida no âmbito do projeto, bem como com serviços inteligentes externos, como o motor de IA e o mecanismo NBQ. Todas as interações com o sistema são efetuadas através de pedidos HTTP/REST, seguindo contratos estáveis, formatos normalizados e mecanismos rigorosos de segurança, garantindo a integridade e consistência dos dados.

A presente secção descreve os principais *endpoints* disponibilizados, os consumidores previstos, as estruturas de mensagens e os componentes transversais de autenticação, erros e paginação, oferecendo uma visão completa da forma como o *back-end* é utilizado pelos restantes módulos da solução.

4.8.1 Enquadramento

O *back-end* implementa o núcleo operacional da plataforma, armazenando e disponibilizando informação autoritativa sobre aplicações, contratos, ambientes, incidentes, problemas, mudanças, *releases*, artigos de conhecimento, *runbooks, discovery* e métricas operacionais. Esta informação é exposta através de uma API REST consumida pela interface de utilizador e por sistemas externos.

Os principais consumidores das interfaces são:

- ***Interface Web*:** responsável por apresentar ao utilizador final a informação do sistema.
- **Motor de IA:** utiliza dados fornecidos pelo *back-end* para calcular *riskScore*, recomendações e elementos analíticos.
- **Serviço NBQ:** apoia o fluxo de *discovery*, solicitando e enviando informação relevante.

Todos os *endpoints* foram concebidos para serem claros, consistentes e facilmente integráveis, de modo a assegurar um funcionamento fluido entre componentes

4.8.2 Interfaces expostas (API pública)

As interfaces seguintes representam o conjunto de *endpoints* disponibilizados pelo *back-end*, organizados por domínio funcional. Todos seguem o padrão REST com *payloads* JSON e métodos semânticos (GET, POST, PUT). Os *endpoints* encontram-se registados no prefixo /api/v1, garantindo estabilidade contratual da API.

4.8.2.1 /api/v1/applications

Objetivo: gerir o catálogo de aplicações AMS.

Consumidores:

- **G2:** para listar e detalhar aplicações na interface de utilizador
- **G3:** para cálculo de scoring, risco e métricas por aplicação

Operações principais:

- **GET /api/v1/applications -** lista aplicações com suporte a paginação e filtro opcional por *criticality*.
- **GET /api/v1/applications/{id} -** consulta os detalhes de uma aplicação específica.
- **POST /api/v1/applications -** cria uma nova aplicação AMS.
- **PUT /api/v1/applications/{id}** - atualiza integralmente os metadados da aplicação (nome, criticality, industry, cliente, etc.).

- **PATCH /api/v1/applications/{id} -** atualiza parcialmente os campos fornecidos no pedido.
- **DELETE /api/v1/applications/{id} -** remove logicamente ou definitivamente uma aplicação do catálogo (consoante a política de negócio).

4.8.2.2 /api/v1/incidents

Objetivo: manter o registo de incidentes associados a cada aplicação, permitindo triagem, análise operacional e histórico de falhas.

Consumidores:

- **G2**: Triage, Assistente com citações, UI de detalhe da aplicação
- **G3**: modelos de risco, recorrência, padrões de falha

Operações principais:

- **GET /api/v1/incidents/applications/{applicationId}** - lista todos os incidentes associados a uma aplicação.
- **GET /api/v1/incidents/{id}** - devolve o detalhe de um incidente específico.
- **POST /api/v1/incidents/applications/{applicationId}** - cria um novo incidente para a aplicação indicada (inclui integração com o Triage de G2).
- **PUT /api/v1/incidents/{id}** - atualiza os dados de um incidente (ex.: título, descrição, severidade, estado).
- **POST /api/v1/incidents/{id}/links** - associa um artigo de conhecimento (KB) ao incidente, permitindo ao assistente referenciar soluções oficiais.

4.8.2.3 /api/v1/changes

Objetivo: registar e acompanhar todas as mudanças (*changes*) aplicadas às aplicações AMS, incluindo janelas de execução, rollback e risco.

Consumidores:

- **G2**: Assistente, UI de detalhe da aplicação
- **G3**: scoring, risco, análise de impacto

Operações principais:

- **GET /api/v1/changes/applications/{applicationId}** - lista todas as changes associadas a uma aplicação.
- **GET /api/v1/changes/{id}** - devolve o detalhe de uma change específica.
- **POST /api/v1/changes/applications/{applicationId}** - cria uma nova change para a aplicação indicada.

- **PUT /api/v1/changes/{id}** - atualiza os dados de uma change (tipo, descrição, janela, rollback, estado, etc.).
- **DELETE /api/v1/changes/{id}** - remove uma change do sistema.

4.8.2.4 /api/v1/kb

Objetivo: armazenar e disponibilizar artigos de conhecimento (KB) oficiais associados a cada aplicação AMS. Estes artigos servem como fonte de verdade para troubleshooting, triagem e apoio ao assistente inteligente.

Consumidores:

- **G2**: KB Editor, Assistente com citações, UI de detalhe da aplicação
- **G3**: modelos de scoring e recomendações contextuais

Operações principais:

- **GET /api/v1/kb/applications/{applicationId}** - lista todos os artigos de conhecimento associados a uma aplicação.
- **GET /api/v1/kb/{id}** - devolve o detalhe de um artigo KB específico.
- **POST /api/v1/kb/applications/{applicationId}** - cria um novo artigo KB oficial para a aplicação indicada.
- **PUT /api/v1/kb/{id}** - atualiza o conteúdo de um artigo KB existente.
- **DELETE /api/v1/kb/{id}** - remove um artigo KB do sistema.

4.8.2.5 /api/v1/runbooks

Objetivo: gerir runbooks oficiais associados às aplicações AMS. Os runbooks descrevem procedimentos operacionais, passos de resolução e instruções formais para equipas de suporte.

Consumidores:

- **G2**: Runbook Editor, Assistente com citações
- **G3**: análise de maturidade operacional e risco

Operações principais:

- **GET /api/v1/runbooks/applications/{applicationId}** - lista todos os runbooks associados a uma aplicação.
- **GET /api/v1/runbooks/{id}** - devolve o detalhe de um runbook específico.
- **POST /api/v1/runbooks/applications/{applicationId}** - cria um novo runbook oficial para a aplicação.

- **PUT /api/v1/runbooks/{id}** - atualiza o conteúdo de um runbook existente.
- **DELETE /api/v1/runbooks/{id}** - remove um runbook do sistema.

4.8.2.6 /api/v1/ applications/{id}/kpis

Objetivo: expor indicadores (KPIs) agregados por aplicação, utilizados em dashboards, relatórios e pelo assistente inteligente.

Consumidores:

- **G2:** dashboards operacionais, ecrãs de detalhe de aplicação
- **G3:** modelos de risco, scoring e priorização

Operações principais:

- **GET /api/v1/applications/{id}/**kpis - devolve um conjunto de métricas agregadas para a aplicação, como por exemplo:
  - disponibilidade
  - MTTR
  - número de incidentes nos últimos N dias
  - volume de changes
  - taxa de recorrência

4.8.2.7 /api/v1/ observability

Objetivo: gerir ligações de observabilidade (dashboards, logs, métricas) associadas às aplicações.

Consumidores:

- **G2**: UI de observabilidade
- **G3**: contexto técnico

Operações principais:

- **POST /api/v1/observability/applications/{applicationId}** - cria um link de observabilidade para a aplicação.
- **GET /api/v1/observability/applications/{applicationId}** - lista links de observabilidade de uma aplicação.
- **DELETE /api/v1/observability/{id}** - remove um link de observabilidade.

4.8.2.8 /api/v1/problems

Objetivo: registar problemas associados às aplicações, suportando análise de causa raiz e recorrência.

Consumidores:

- **G2**: UI de problemas, Assistente
- **G3**: análise de estabilidade

Operações principais:

- **POST /api/v1/problems/applications/{applicationId}** - cria um problema para a aplicação.
- **GET /api/v1/problems/applications/{applicationId}** - lista problemas de uma aplicação.
- **GET /api/v1/problems/{id}** - devolve o detalhe de um problema.
- **PUT /api/v1/problems/{id}** - atualiza um problema.
- **DELETE /api/v1/problems/{id}** - remove um problema.

4.8.2.9 /api/v1/releases

Objetivo: gerir releases associadas às aplicações, permitindo relacionar mudanças, incidentes e janelas de implementação.

Consumidores:

- **G2**: UI de releases
- **G3**: análise de impacto e risco

Operações principais:

- **POST /api/v1/releases/applications/{applicationId}** - cria uma release para a aplicação.
- **GET /api/v1/releases/applications/{applicationId}** - lista releases de uma aplicação.
- **GET /api/v1/releases/{id}** - devolve o detalhe de uma release.
- **PUT /api/v1/releases/{id}** - atualiza uma release.
- **DELETE /api/v1/releases/{id}** - remove uma release.

4.8.2.10 /api/v1/contract/applications/{applicationId}

Objetivo: gerir o contrato de serviço associado a cada aplicação AMS.

Consumidores:

- **G2**: UI de contrato
- **G3**: contexto de SLAs/SLOs

Operações principais:

- **PUT /api/v1/contract/applications/{applicationId}** - cria ou atualiza (upsert) o contrato de serviço da aplicação.
- **GET /api/v1/contract/applications/{applicationId}** - devolve o contrato de serviço da aplicação.
- **DELETE /api/v1/contract/applications/{applicationId}** - remove o contrato de serviço.

4.8.2.11 /api/v1/clients

Objetivo: gerir clientes associados às aplicações AMS.

Consumidores:

G2: UI de clientes, associação de aplicações a clientes

G3: segmentação, análise por cliente

Operações principais:

- **POST /api/v1/clients** - cria um novo cliente.
- **GET /api/v1/clients** - lista todos os clientes.

4.8.2.12 /auth

Objetivo: fornecer mecanismos de autenticação e gestão de tokens para acesso seguro à API.

Consumidores: Front-end (**G2**), ferramentas internas, integrações autenticadas.

Operações principais:

- **POST /auth/register** - regista um novo utilizador.
- **POST /auth/login** - autentica um utilizador e devolve token.
- **POST /auth/refresh** - renova o token de acesso.

4.8.2.13 /health e /ready

Objetivo: expor o estado de saúde da aplicação e a sua prontidão para servir tráfego.

Consumidores: Plataformas de monitorização, orquestradores, pipelines de deployment.

Operações principais:

- **GET /health** - indica se a aplicação está “up”.
- **GET /ready** - verifica se a aplicação está pronta e se a base de dados está acessível.

4.8.2.14 /discovery/{applicationId}

Objetivo: guardar as respostas normalizadas do *Intake* (perguntas NBQ).

Consumidores: G2 (*Intake Wizard, Coverage*), G3 (para *scoring* e recomendações).

Operações principais:

- **POST /discovery/{applicationId}** - grava ou atualiza uma resposta para uma “key” específica.

Payload inclui:

  - **key** - identificador lógico da informação (ex.: checkout_slos)
  - **value** - valor normalizado (string, número, boolean ou objeto)
  - **questionRef** - referência à pergunta NBQ (ex.: checkout_slos@1.0.0)
  - **unknown / reason** - se o operador marcou como “Unknown”.

4.8.2.15 /export/{applicationId}

Objetivo: fornecer uma visão consolidada das respostas de *discovery* para auditoria/*reporting*.
Consumidores: G2 (*export* UI) ou ferramentas externas.

Operações principais:

- **GET /export/{applicationId}** - devolve todas as chaves com value, questionRef, updatedAt, etc.

4.8.3 Interfaces consumidas pelo back-end

Para além das interfaces expostas ao exterior, o *back-end* recorre também a um conjunto reduzido de interfaces internas que suportam o funcionamento do núcleo operacional da plataforma. Estas interfaces são responsáveis pelo acesso aos dados persistidos e garantem a estrutura necessária para suportar as operações executadas pelos serviços de domínio.

O *back-end* comunica principalmente com:

- **Base de dados relacional (ex.: PostgreSQL)**

- Acesso realizado através de um **ORM/repositórios**, garantindo isolamento entre lógica de negócio e persistência.
- Entidades armazenadas incluem:
  - *Application*
  - *Incident*
  - *Problem*
  - *Change*
  - *Release*
  - *KB*
  - *Runbook*
  - *DiscoveryAnswer*
  - *KPI*
- Consistência assegurada através de transações, validação e integridade relacional.

- **Storage de ficheiros (se aplicável)**
  - Apenas necessário se existirem anexos provenientes do *front-end*.
  - Não incluído no escopo da versão atual, sendo a gestão de anexos da responsabilidade da interface (caso exista).

Internamente, o *back-end* comunica apenas com a base de dados relacional através de repositórios (ex.: *ApplicationRepository, IncidentRepository*). Não existem integrações diretas com plataformas APM ou ingestão de *logs/metrics* nesta versão, estando essa funcionalidade fora do âmbito do MVP.

4.8.4 Contratos transversais da API

Todas as interfaces expostas seguem um conjunto uniforme de contratos, garantindo previsibilidade e consistência no consumo da API:

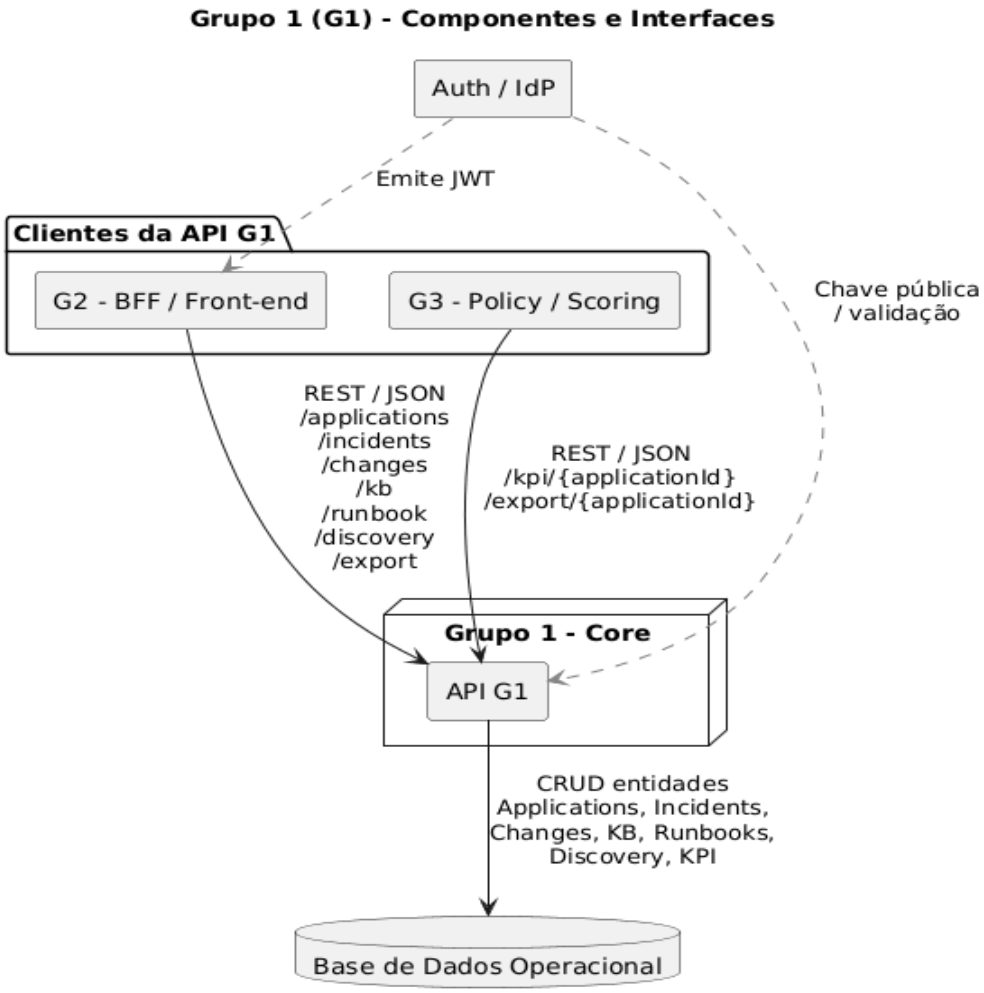
- **Formato das mensagens:**
  - JSON codificado em UTF-8
  - Estrutura consistente e validada por esquema
  - Suporte para objetos compostos e arrays
- **Autenticação e autorização:**
  - Baseada em **JWT**, enviado no header: Authorization: Bearer <token>
  - Validação obrigatória antes da execução de qualquer operação
  - Regras de acesso aplicadas com base nos papéis do utilizador:
    - *viewer*

        - *operator*
        - *owner*

- **Modelo de erros:**
    - Em caso de falha, a API devolve objetos estruturados:
        - {
        - "code": "VALIDATION_ERROR",
        - "message": "Criticality value not allowed",
        - "details": { "field": "criticality" }
        - }
- **Códigos HTTP seguem o padrão:**
    - **2xx** – sucesso
    - **4xx** – erro do cliente
    - **5xx** – erro interno
- **Paginação e filtros:**
    - Suporte para:
        - *page*
        - *pageSize*
        - *sort*
    - Filtros específicos por domínio (ex.: *applicationId, severity, type*)

A **Figura 18** ilustra um exemplo real de interação com a API do *back-end,* evidenciando o formato de resposta exposto aos consumidores da plataforma. Esta representação permite visualizar de forma concreta como os dados são devolvidos aos clientes (interface web, motor de IA ou ferramentas externas), demonstrando a estrutura JSON, os campos relevantes e o comportamento esperado dos *endpoints* descritos nesta secção.

Figura 18 - Exemplo de interface exposta pelo *back-end*



5 Testes e Validação

5.1 Abordagem de Testes

A validação da solução desenvolvida foi realizada através de uma abordagem estruturada que combinou **testes unitários, testes de integração e análise de cobertura de código**, com o objetivo de garantir a qualidade, robustez e funcionalidade do back-end AMS em contexto real.

Mais do que demonstrar o funcionamento técnico da aplicação, os testes tiveram como principal finalidade validar que a solução cumpre os objetivos definidos, nomeadamente a gestão eficiente de operações AMS, incluindo incidentes, mudanças, conhecimento (KB) e métricas (KPIs), conforme especificado nos requisitos do sistema.

A estratégia de testes adotada seguiu uma abordagem **bottom-up**, incidindo sobre os principais componentes da aplicação:

- **Controladores (REST Controllers)** - validação de endpoints, códigos HTTP e payloads
- **Serviços (Business Logic)** - validação das regras de negócio e fluxos operacionais
- **Mappers (DTO - Entity)** - consistência na transformação de dados
- **Modelos e Enums** - validação de restrições e integridade dos dados
- **Gestão de exceções** - tratamento correto de erros e respostas da API

Foram utilizadas as seguintes tecnologias:

- **JUnit 5** - framework principal de testes
- **Mockito** - simulação de dependências
- **Spring MockMvc** - testes de endpoints REST
- **AssertJ** - validação expressiva de resultados
- **JaCoCo** - análise de cobertura de código

5.2 Execução dos Testes

A execução dos testes foi realizada através da ferramenta Maven Wrapper, utilizando o comando **(.\mvnw clean test)**. Este comando permite limpar o projeto, compilar o código e executar automaticamente todos os testes definidos na aplicação.

Como se pode observar na **Figura 19**, foram executados 359 testes, não tendo sido registadas falhas, erros ou testes ignorados. O resultado final indica **"BUILD SUCCESS"**, confirmando que todos os testes foram concluídos com sucesso.

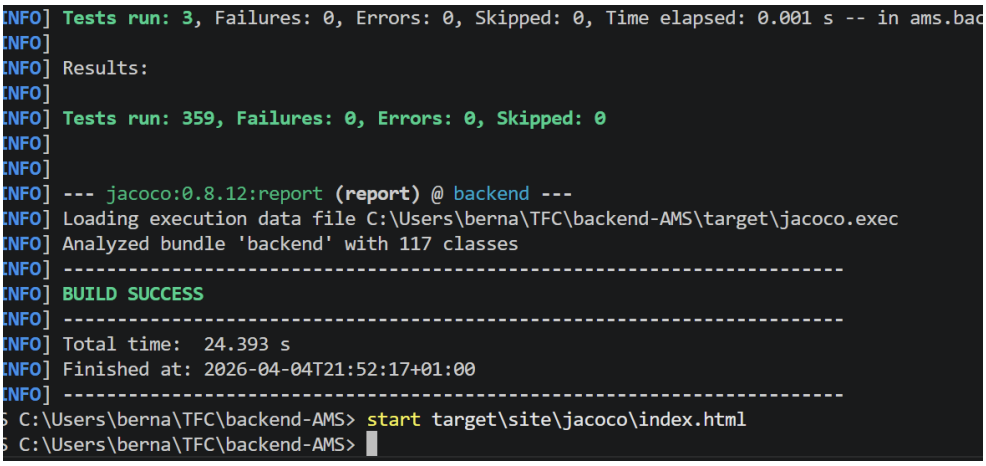
Adicionalmente, é possível verificar que o processo inclui a geração automática do relatório de cobertura através do JaCoCo, evidenciado pela execução do módulo **jacoco:report**, bem como o tempo total de execução do processo.

Este resultado demonstra:

- Elevado nível de estabilidade da aplicação
- Correta implementação das funcionalidades testadas
- Ausência de erros críticos no sistema

Assim, a execução bem-sucedida da totalidade dos testes constitui uma evidência objetiva da qualidade e fiabilidade do back-end desenvolvido.

Figura 19 - Resultado da execução dos testes unitários com Maven



5.3 Análise de Cobertura de Código

Após a execução dos testes, foi gerado automaticamente um relatório de cobertura de código com recurso à ferramenta JaCoCo, permitindo avaliar o grau de validação das diferentes componentes da aplicação.O relatório pode ser consultado através do comando **(start target\site\jacoco\index.html)**.

A **Figura 20** apresenta uma visão global da cobertura de código do projeto.De acordo com o relatório apresentado, o projeto alcança:

- **87% de cobertura de instruções**
- **71% de cobertura de ramos (branches)**

A análise detalhada da cobertura permite retirar as seguintes conclusões:

- As camadas de **serviços (business logic)** apresentam níveis de cobertura elevados, frequentemente superiores a 95%, garantindo a validação das principais regras de negócio.

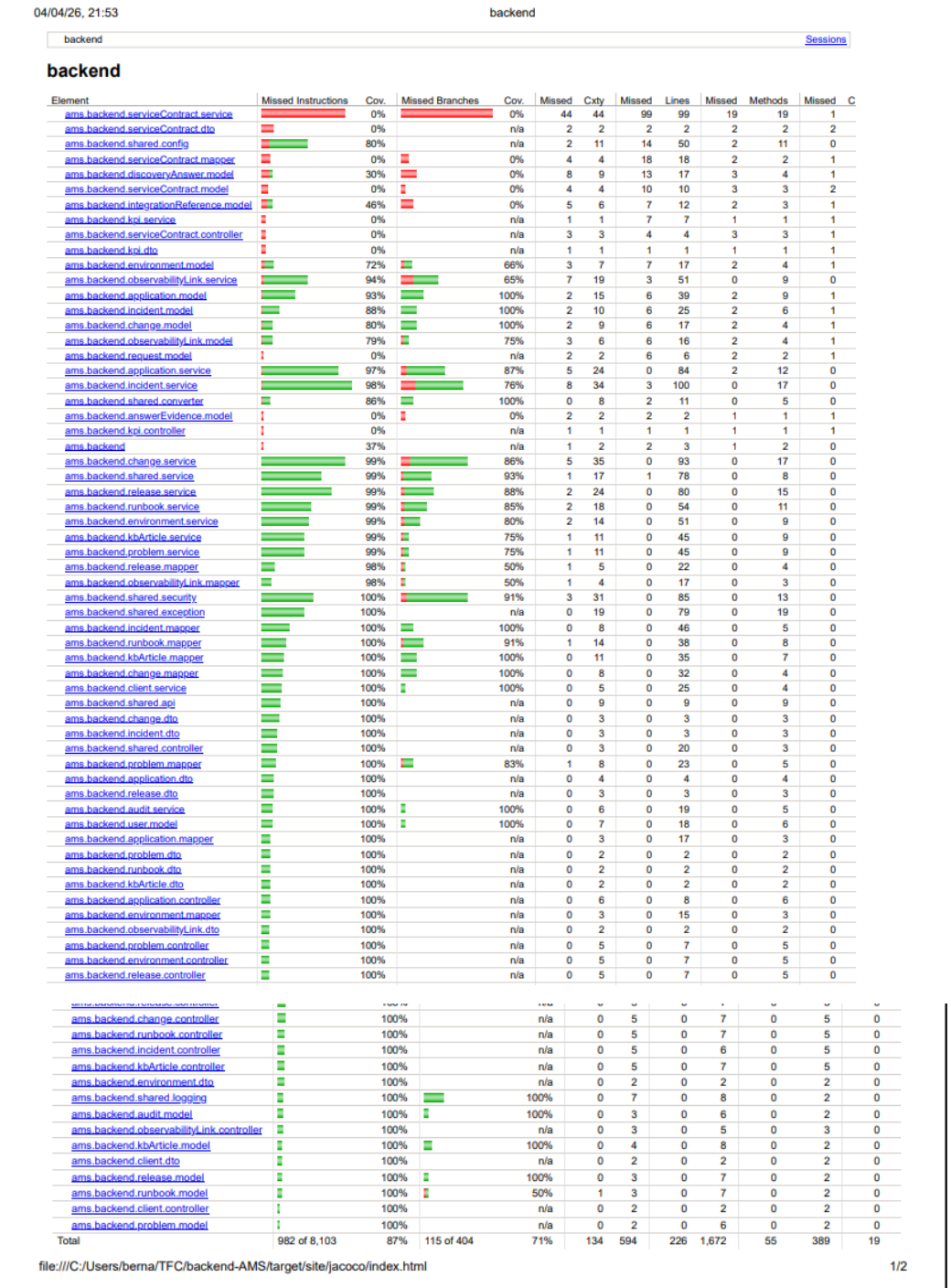
- Os **controladores REST** e **mappers** encontram-se praticamente totalmente cobertos , assegurando a consistência da API.
- As **entidades e modelos** apresentam igualmente uma cobertura significativa, validando restrições e regras de integridade

Por outro lado, algumas áreas apresentam menor cobertura, nomeadamente:

- Componentes relacionados com **ServiceContract e KPI**, ainda em evolução funcional
- Classes auxiliares ou menos críticas para o fluxo principal da aplicação

Apesar disso, a cobertura global obtida é considerada elevada e alinhada com boas práticas de desenvolvimento, garantindo um nível adequado de confiança na estabilidade do sistema. A utilização do JaCoCo permitiu ainda identificar zonas com menor cobertura, possibilitando a melhoria contínua da qualidade do código através da adição de novos testes.

Figura 20 - Relatório de cobertura de código



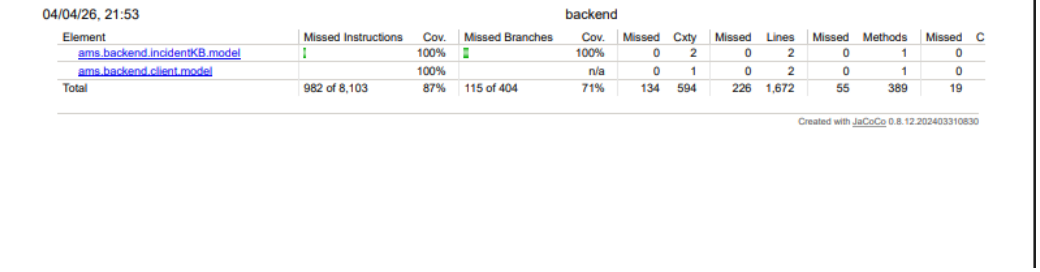
04/04/26, 21:53 backend

backend Sessions

backend

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	C
ams.backend.serviceContract.service		0%		0%	44	44	99	99	19	19	1	
ams.backend.serviceContract.dto		0%		n/a	2	2	2	2	2	2	2	
ams.backend.shared.config		80%		n/a	2	11	14	50	2	11	0	
ams.backend.serviceContract.mapper		0%		0%	4	4	18	18	2	2	1	
ams.backend.discoveryAnswer.model		30%		0%	8	9	13	17	3	4	1	
ams.backend.serviceContract.model		0%		0%	4	4	10	10	3	3	2	
ams.backend.integrationReference.model		46%		0%	5	6	7	12	2	3	1	
ams.backend.kpi.service		0%		n/a	1	1	7	7	1	1	1	
ams.backend.serviceContract.controller		0%		n/a	3	3	4	4	3	3	1	
ams.backend.kpi.dto		0%		n/a	1	1	1	1	1	1	1	
ams.backend.environment.model		72%		66%	3	7	7	17	2	4	1	
ams.backend.observabilityLink.service		94%		65%	7	19	3	51	0	9	0	
ams.backend.application.model		93%		100%	2	15	6	39	2	9	1	
ams.backend.incident.model		88%		100%	2	10	6	25	2	6	1	
ams.backend.change.model		80%		100%	2	9	6	17	2	4	1	
ams.backend.observabilityLink.model		79%		75%	3	6	6	16	2	4	1	
ams.backend.request.model		0%		n/a	2	2	6	6	2	2	1	
ams.backend.application.service		97%		87%	5	24	0	84	2	12	0	
ams.backend.incident.service		98%		76%	8	34	3	100	0	17	0	
ams.backend.shared.converter		86%		100%	0	8	2	11	0	5	0	
ams.backend.answerEvidence.model		0%		0%	2	2	2	2	1	1	1	
ams.backend.kpi.controller		0%		n/a	1	1	1	1	1	1	1	
ams.backend		37%		n/a	1	2	2	3	1	2	0	
ams.backend.change.service		99%		86%	5	35	0	93	0	17	0	
ams.backend.shared.service		99%		93%	1	17	1	78	0	8	0	
ams.backend.release.service		99%		88%	2	24	0	80	0	15	0	
ams.backend.runbook.service		99%		85%	2	18	0	54	0	11	0	
ams.backend.environment.service		99%		80%	2	14	0	51	0	9	0	
ams.backend.kbArticle.service		99%		75%	1	11	0	45	0	9	0	
ams.backend.problem.service		99%		75%	1	11	0	45	0	9	0	
ams.backend.release.mapper		98%		50%	1	5	0	22	0	4	0	
ams.backend.observabilityLink.mapper		98%		50%	1	4	0	17	0	3	0	
ams.backend.shared.security		100%		91%	3	31	0	85	0	13	0	
ams.backend.shared.exception		100%		n/a	0	19	0	79	0	19	0	
ams.backend.incident.mapper		100%		100%	0	8	0	46	0	5	0	
ams.backend.runbook.mapper		100%		91%	1	14	0	38	0	8	0	
ams.backend.kbArticle.mapper		100%		100%	0	11	0	35	0	7	0	
ams.backend.change.mapper		100%		100%	0	8	0	32	0	4	0	
ams.backend.client.service		100%		100%	0	5	0	25	0	4	0	
ams.backend.shared.api		100%		n/a	0	9	0	9	0	9	0	
ams.backend.change.dto		100%		n/a	0	3	0	3	0	3	0	
ams.backend.incident.dto		100%		n/a	0	3	0	3	0	3	0	
ams.backend.shared.controller		100%		n/a	0	3	0	20	0	3	0	
ams.backend.problem.mapper		100%		83%	1	8	0	23	0	5	0	
ams.backend.application.dto		100%		n/a	0	4	0	4	0	4	0	
ams.backend.release.dto		100%		n/a	0	3	0	3	0	3	0	
ams.backend.audit.service		100%		100%	0	6	0	19	0	5	0	
ams.backend.user.model		100%		100%	0	7	0	18	0	6	0	
ams.backend.application.mapper		100%		n/a	0	3	0	17	0	3	0	
ams.backend.problem.dto		100%		n/a	0	2	0	2	0	2	0	
ams.backend.runbook.dto		100%		n/a	0	2	0	2	0	2	0	
ams.backend.kbArticle.dto		100%		n/a	0	2	0	2	0	2	0	
ams.backend.application.controller		100%		n/a	0	6	0	8	0	6	0	
ams.backend.environment.mapper		100%		n/a	0	3	0	15	0	3	0	
ams.backend.observabilityLink.dto		100%		n/a	0	2	0	2	0	2	0	
ams.backend.problem.controller		100%		n/a	0	5	0	7	0	5	0	
ams.backend.environment.controller		100%		n/a	0	5	0	7	0	5	0	
ams.backend.release.controller		100%		n/a	0	5	0	7	0	5	0	
ams.backend.change.controller		100%		n/a	0	5	0	7	0	5	0	
ams.backend.runbook.controller		100%		n/a	0	5	0	7	0	5	0	
ams.backend.incident.controller		100%		n/a	0	5	0	6	0	5	0	
ams.backend.kbArticle.controller		100%		n/a	0	5	0	7	0	5	0	
ams.backend.environment.dto		100%		n/a	0	2	0	2	0	2	0	
ams.backend.shared.logging		100%		100%	0	7	0	8	0	2	0	
ams.backend.audit.model		100%		100%	0	3	0	6	0	2	0	
ams.backend.observabilityLink.controller		100%		n/a	0	3	0	5	0	3	0	
ams.backend.kbArticle.model		100%		100%	0	4	0	8	0	2	0	
ams.backend.client.dto		100%		n/a	0	2	0	2	0	2	0	
ams.backend.release.model		100%		100%	0	3	0	7	0	2	0	
ams.backend.runbook.model		100%		50%	1	3	0	7	0	2	0	
ams.backend.client.controller		100%		n/a	0	2	0	2	0	2	0	
ams.backend.problem.model		100%		n/a	0	2	0	6	0	2	0	
Total	982 of 8,103	87%	115 of 404	71%	134	594	226	1,672	55	389	19	

file:///C:/Users/berna/TFC/backend-AMS/target/site/jacoco/index.html 1/2



04/04/26, 21:53 backend

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	C
ams.backend.incidentKB.model		100%		100%	0	2	0	2	0	1	0	
ams.backend.client.model		100%		n/a	0	1	0	2	0	1	0	
Total	982 of 8,103	87%	115 of 404	71%	134	594	226	1,672	55	389	19	

Created with JaCoCo 0.8.12.202403310830

 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.

5.4 Estrutura dos Testes

A organização dos testes segue a mesma estrutura modular do código fonte, permitindo uma separação clara entre as diferentes camadas da aplicação e garantindo maior facilidade de manutenção e escalabilidade.

A **Figura 21** apresenta a estrutura dos testes implementados no projeto. Como se pode observar, os testes encontram-se organizados por módulos funcionais (por exemplo, *application*, *incident*, *change*, *client*), refletindo a arquitetura do sistema. Dentro de cada módulo, os testes estão divididos de acordo com as respetivas camadas:

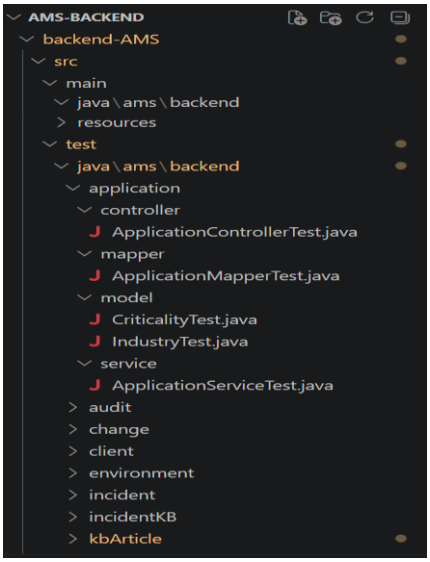
- **controller** - testes aos endpoints REST (ex: ApplicationControllerTest)
- **service** - testes à lógica de negócio (ex: ApplicationServiceTest)
- **mapper** - testes de conversão entre DTOs e entidades
- **model** - validação de enums e regras de domínio (ex: CriticalityTest, IndustryTest)

Esta abordagem apresenta várias vantagens:

- **Isolamento dos testes** - cada camada é testada de forma independente
- **Maior clareza estrutural** - alinhamento direto com a arquitetura do projeto
- **Facilidade de manutenção** - novos testes podem ser adicionados de forma consistente
- **Escalabilidade** - suporte para crescimento do sistema sem perda de organização

Além disso, esta organização permite garantir que todas as componentes críticas do sistema(desde a camada de apresentação até à lógica de negócio) são devidamente validadas, contribuindo para a robustez global da aplicação.

Figura 21 - Estrutura dos testes no projeto backend AMS



5.5 Avaliação Global e Validação da Solução

A abordagem de testes adotada, combinando testes unitários, validação de endpoints e análise de cobertura de código, permitiu garantir um elevado nível de qualidade e fiabilidade da solução desenvolvida.

Os resultados obtidos evidenciam a robustez do sistema:

- Execução de **359 testes com sucesso**, sem falhas ou erros
- **Cobertura de código de 87% (instruções)** e **71% (branches)**
- Validação consistente das principais camadas da aplicação (controller, service, mapper e model)

A análise conjunta das **Figuras 19, 20 e 21** demonstra que a aplicação foi testada de forma abrangente, desde a execução dos testes até à verificação da cobertura e organização estrutural dos mesmos.

Do ponto de vista funcional, os testes asseguram que o sistema cumpre os requisitos definidos, nomeadamente:

- Gestão de aplicações e ambientes
- Registo e acompanhamento de incidentes
- Associação de artigos de conhecimento (KB)
- Suporte a operações de mudança (change)
- Garantia de integridade e validação de dados

Adicionalmente, a utilização de ferramentas como MockMvc permitiu simular cenários próximos de um ambiente real, validando o comportamento da API em condições semelhantes às de utilização prática. A identificação de áreas com menor cobertura, através do JaCoCo, possibilita ainda uma evolução contínua do sistema, permitindo reforçar a qualidade do código em futuras iterações.

Assim, pode concluir-se que a solução desenvolvida cumpre os objetivos propostos, demonstrando aplicabilidade no contexto de Application Management Services (AMS), e apresentando um nível de qualidade adequado para utilização em ambiente real.

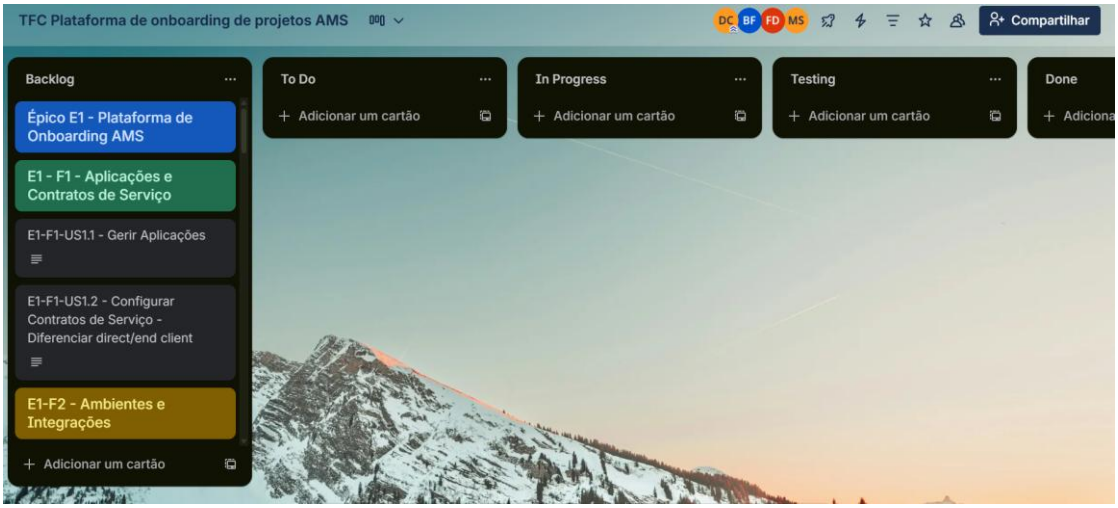
6 Método e Planeamento

6.1 Planeamento inicial

Este capítulo descreve o método de trabalho seguido na execução do projeto e o planeamento das atividades realizadas e previstas. O objetivo foi garantir uma gestão organizada, permitindo acompanhar a evolução do trabalho e assegurar que todas as tarefas necessárias à implementação do TFC fossem concluídas dentro dos prazos estabelecidos. O projeto foi organizado em etapas sucessivas, cada uma contendo tarefas próprias e prazos de entrega bem definidos.

Como suporte ao planeamento e acompanhamento do trabalho, foi utilizado o Trello para organizar as tarefas segundo princípios de Scrum. A ferramenta permitiu estruturar o backlog, distribuir trabalho por sprints, acompanhar o progresso das atividades e garantir transparência na evolução do projeto. Através das diferentes colunas (To Do, In Progress, Done), tornou-se possível visualizar rapidamente o estado de cada tarefa, gerir prioridades e assegurar que as várias fases, desde o levantamento de requisitos até à implementação e documentação, avançavam de forma consistente com o planeamento definido.

Figura 22 - Quadro de tarefas no Trello segundo metodologia Scrum



6.1.1 Introdução e Planeamento do Projeto

A primeira fase de entrega teve como principal objetivo estabelecer a base organizacional do trabalho. Onde esta esta dividida desta forma:

- **Introdução ao TFC (07/10/25 – 20/10/25):** Definição do enquadramento, objetivos e metodologia de trabalho.
- **Levantamento de requisitos (20/10/25 – 25/10/25):** Identificação e análise dos requisitos funcionais, não funcionais e técnicos do projeto.

- **User Stories & Backlog (22/10/25 – 27/10/25):** Criação e validação das user stories relacionadas com aplicações, contratos, ambientes, incidentes, problemas, mudanças, conhecimento, discovery, KPIs e segurança.
- **Modelação e Arquitetura (27/10/25 – 07/11/25):** Produção de casos de uso, diagramas de arquitetura, diagrama de pacotes e modelação conceptual do sistema.
- **Relatório da 1.ª entrega (10/10/25 – 29/11/25):** Redação das secções de enquadramento, pertinência, requisitos, modelação e arquitetura técnica.

Todas as tarefas desta fase foram concluídas com sucesso, garantindo uma base sólida para o desenvolvimento nas fases seguintes.

6.1.2 Desenvolvimento e Implementação

Esta etapa marca o início da construção técnica do sistema, com a implementação dos vários componentes, prevista para ocorrer ao longo dos primeiros meses de 2026. Onde esta esta dividida desta forma:

- **Configuração do ambiente (06/01/26 – 08/01/26):** Instalação das dependências, preparação da base de dados, estrutura inicial do projeto e configuração do ambiente de desenvolvimento.
- **Desenvolvimento dos módulos:**
    - **Core Operacional (08/01/26 – 12/01/26):** Estrutura base da API, controlo de acesso (JWT), logging, auditoria e modelos principais.
    - **Aplicações & Contratos (12/01/26 – 24/01/26):** CRUD de aplicações, contratos de serviço e auditoria das operações.
    - **Ambientes & Integrações (24/01/26 – 02/02/26):** Registo de ambientes, links de observabilidade e readiness/health-checks.
    - **Incidentes, Problemas e Pedidos (02/02/26 – 19/02/26):** Registo de incidentes, timestamps, validações e ligações a problemas/KB.
    - **Mudanças & Releases (19/02/26 – 28/02/26):** Gestão de changes, releases e cálculo de riskScore.
    - **Conhecimento & Runbooks (28/02/26 – 07/03/26):** Gestão de artigos de KB e runbooks.
    - **Discovery & NBQ (07/03/26 – 14/03/26):** Registo de respostas normalizadas e exportação para o motor de IA.
- **Integração com sistemas externos (07/02/26 – 10/03/26):** Webhook ITSM (ingestão de incidentes/pedidos) + ligação ao motor de scoring.
- **Automação e otimização (06/01/26 – 14/03/26):** Implementação de logs estruturados, métricas, testes automáticos e CI/CD básico.

- **Relatório da 2.ª entrega (14/03/26 – 12/04/26):** Documentação do progresso, funcionalidades implementadas e revisão das secções técnicas.

6.1.3 Validação e Entrega Final

Esta fase corresponde à última etapa do desenvolvimento e prepara a disponibilização da solução.

- **Testes e validação (01/04/26 – 10/05/26):** Testes funcionais, testes de carga, validação de KPIs, auditoria e readiness.
- **Análise crítica ao planeamento (10/05/26 – 18/05/26):** Avaliação do cumprimento do cronograma, identificação de desvios e impacto.
- **Relatório Final (10/05/26 – 23/06/26):** Consolidação da documentação, reflexão sobre o projeto, resultados e conclusões.

6.2 Cronograma

O cronograma detalha a calendarização global das atividades previstas para o desenvolvimento do projeto, permitindo acompanhar a evolução das tarefas e garantir que todas as fases são concluídas dentro dos prazos definidos. A estrutura temporal apresentada reflete um planeamento progressivo, orientado tanto pela análise e levantamento de requisitos, como pela construção técnica do back-end e posterior validação da solução. O cronograma serve igualmente como instrumento de gestão, facilitando a identificação de dependências, períodos críticos e distribuição do esforço ao longo do projeto.

A **Tabela 5** apresenta o cronograma completo, organizado por fases e tarefas, indicando o progresso atual, as datas de início e fim previstas para cada atividade.

Tabela 5-Cronograma

TASK NAME	START DATE	DAY OF MONTH*	END DATE	TEAM MEMBER
Introdução e Planeamento do Projeto				
Introdução ao TFC	10/7	7	10/20	Duarte/Bernardo
Levantamento de Requisitos	10/20	20	10/25	Duarte/Bernardo
User Stories & Backlog	10/22	22	10/27	Duarte/Bernardo
Modelação e Arquitetura	10/27	27	11/7	Duarte/Bernardo
Relatório da 1.ª Entrega	10/10	10	11/29	Duarte/Bernardo
Desenvolvimento e Implementação				
Configuração do Ambiente	1/6	6	1/8	Duarte/Bernardo

Core Operacional	1/8	8	1/12	Duarte
Aplicações & Contratos	1/12	12	1/24	Bernardo
Ambientes & Integrações	1/24	24	2/2	Bernardo
Incidentes, Problemas e Pedidos	2/2	2	2/19	Duarte
Mudanças & Releases	2/19	19	2/28	Bernardo
Conhecimento & Runbooks	2/28	28	3/7	Duarte
Discovery & NBQ	3/7	7	3/14	Duarte/Bernardo
Integração com Sistemas Externos	2/7	7	3/10	Duarte/Bernardo
Automação & Otimização	1/6	6	3/14	Duarte/Bernardo
Relatório da 2.ª Entrega	3/14	14	4/12	Duarte/Bernardo
Validação e Entrega Final				
Testes e Validação	4/1	1	5/10	Duarte/Bernardo
Análise Crítica ao Planeamento	5/10	10	5/18	Duarte/Bernardo
Relatório Final	5/10	10	6/23	Duarte/Bernardo

6.3 Ferramentas de Planeamento e Acompanhamento

Durante o desenvolvimento do projeto, a CGI disponibilizou uma ferramenta de gestão de trabalho, o **Notion**, que permitiu organizar, acompanhar e validar todas as tarefas relacionadas com o desenvolvimento do back-end da plataforma AMS.

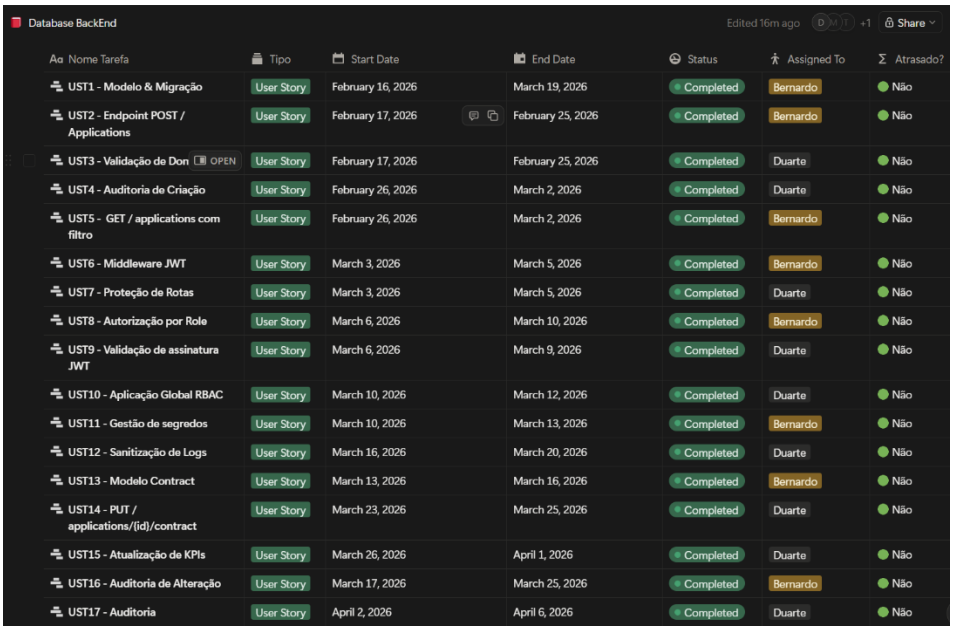
Esta ferramenta funcionou como um quadro centralizado de planeamento, onde eram registadas todas as **User Stories (UST)**, incluindo a sua descrição, datas previstas, responsáveis, estado de execução e eventuais atrasos. A sua utilização foi essencial para garantir uma visão global e atualizada do progresso do projeto.

O **Notion** incluí as seguintes funcionalidades:

- Lista completa de User Stories atribuídas ao grupo
- Datas de início e fim planeadas para cada tarefa
- Estado em tempo real (Completed, Open, In Progress)
- Identificação do responsável por cada tarefa
- Indicação automática de atraso (“Atrasado? Sim/Não”)
- Classificação por tipo (User Story, Task, Goal, etc.)

A **Figura 23** apresenta o quadro utilizado no acompanhamento do desenvolvimento do back-end, onde é possível observar as diferentes User Stories, os respetivos estados e a distribuição de responsabilidades pelos elementos da equipa.

Figura 23 - Quadro de planeamento das User Stories no Notion



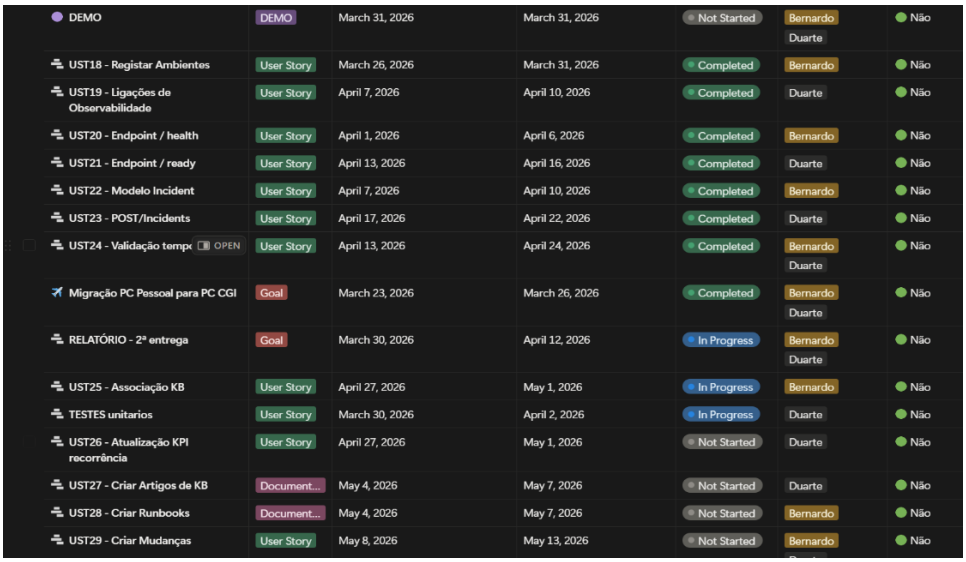
Database BackEnd — Edited 16m ago — Share

Nome Tarefa	Tipo	Start Date	End Date	Status	Assigned To	Atrasado?
UST1 - Modelo & Migração	User Story	February 16, 2026	March 19, 2026	Completed	Bernardo	Não
UST2 - Endpoint POST / Applications	User Story	February 17, 2026	February 25, 2026	Completed	Bernardo	Não
UST3 - Validação de Don OPEN	User Story	February 17, 2026	February 25, 2026	Completed	Duarte	Não
UST4 - Auditoria de Criação	User Story	February 26, 2026	March 2, 2026	Completed	Duarte	Não
UST5 - GET / applications com filtro	User Story	February 26, 2026	March 2, 2026	Completed	Bernardo	Não
UST6 - Middleware JWT	User Story	March 3, 2026	March 5, 2026	Completed	Bernardo	Não
UST7 - Proteção de Rotas	User Story	March 3, 2026	March 5, 2026	Completed	Duarte	Não
UST8 - Autorização por Role	User Story	March 6, 2026	March 10, 2026	Completed	Bernardo	Não
UST9 - Validação de assinatura JWT	User Story	March 6, 2026	March 9, 2026	Completed	Duarte	Não
UST10 - Aplicação Global RBAC	User Story	March 10, 2026	March 12, 2026	Completed	Duarte	Não
UST11 - Gestão de segredos	User Story	March 10, 2026	March 13, 2026	Completed	Bernardo	Não
UST12 - Sanitização de Logs	User Story	March 16, 2026	March 20, 2026	Completed	Duarte	Não
UST13 - Modelo Contract	User Story	March 13, 2026	March 16, 2026	Completed	Bernardo	Não
UST14 - PUT / applications/{id}/contract	User Story	March 23, 2026	March 25, 2026	Completed	Duarte	Não
UST15 - Atualização de KPIs	User Story	March 26, 2026	April 1, 2026	Completed	Duarte	Não
UST16 - Auditoria de Alteração	User Story	March 17, 2026	March 25, 2026	Completed	Bernardo	Não
UST17 - Auditoria	User Story	April 2, 2026	April 6, 2026	Completed	Duarte	Não

Como se pode verificar, tarefas relacionadas com funcionalidades críticas(**endpoints REST, modelos de dados, segurança (JWT e RBAC), auditoria, KPIs e contratos de serviço**) foram organizadas e acompanhadas de forma estruturada, garantindo o cumprimento dos prazos definidos.

Adicionalmente, a **Figura 24** evidencia a fase mais avançada do projeto, incluindo tarefas de integração, testes e documentação, onde algumas atividades se encontravam ainda em progresso ou por iniciar.

Figura 24 - Estado de execução das tarefas em fase avançada



DEMO	DEMO	March 31, 2026	March 31, 2026	Not Started	Bernardo Duarte	Não
UST18 - Registar Ambientes	User Story	March 26, 2026	March 31, 2026	Completed	Bernardo	Não
UST19 - Ligações de Observabilidade	User Story	April 7, 2026	April 10, 2026	Completed	Duarte	Não
UST20 - Endpoint / health	User Story	April 1, 2026	April 6, 2026	Completed	Bernardo	Não
UST21 - Endpoint / ready	User Story	April 13, 2026	April 16, 2026	Completed	Duarte	Não
UST22 - Modelo Incident	User Story	April 7, 2026	April 10, 2026	Completed	Bernardo	Não
UST23 - POST/Incidents	User Story	April 17, 2026	April 22, 2026	Completed	Duarte	Não
UST24 - Validação tempo OPEN	User Story	April 13, 2026	April 24, 2026	Completed	Bernardo Duarte	Não
Migração PC Pessoal para PC CGI	Goal	March 23, 2026	March 26, 2026	Completed	Bernardo Duarte	Não
RELATÓRIO - 2ª entrega	Goal	March 30, 2026	April 12, 2026	In Progress	Bernardo Duarte	Não
UST25 - Associação KB	User Story	April 27, 2026	May 1, 2026	In Progress	Bernardo	Não
TESTES unitarios	User Story	March 30, 2026	April 2, 2026	In Progress	Duarte	Não
UST26 - Atualização KPI recorrência	User Story	April 27, 2026	May 1, 2026	Not Started	Duarte	Não
UST27 - Criar Artigos de KB	Document...	May 4, 2026	May 7, 2026	Not Started	Duarte	Não
UST28 - Criar Runbooks	Document...	May 4, 2026	May 7, 2026	Not Started	Bernardo	Não
UST29 - Criar Mudanças	User Story	May 8, 2026	May 13, 2026	Not Started	Bernardo	Não

A utilização desta ferramenta revelou-se fundamental para:

- Aplicar metodologias ágeis (nomeadamente Scrum) de forma prática
- Documentar o progresso do projeto de forma objetiva e contínua
- Garantir transparência perante a entidade empresarial (CGI) e os orientadores académicos
- Apoiar a organização e coordenação interna da equipa

Assim, o uso do Notion contribuiu significativamente para uma gestão eficiente do projeto, assegurando que o desenvolvimento do back-end decorresse de forma controlada, previsível e alinhada com os objetivos definidos.

6.4 Análise Crítica ao Planeamento

A análise crítica tem como principal objetivo avaliar a adequação do cronograma face às atividades previstas, identificando riscos, dependências e possíveis ajustes necessários para garantir a entrega a tempo da solução.

Riscos identificados:

- Complexidade técnica do domínio AMS
- Dependências externas (ITSM, motores de IA)
- Possíveis conflitos na integração entre módulos
- Esforço adicional na definição de métricas e KPIs

Dependências principais:

- Conclusão do Core antes dos restantes módulos

- Base de dados e schema definidos antes da implementação
- Necessidade de validação constante com user stories e requisitos

Medidas de mitigação:

- Planeamento iterativo organizado por sprints curtos
- Revisão contínua do backlog e requisitos
- Ambiente de testes isolado para validação
- Documentação contínua para prevenir falhas e retrabalho

7 Resultados

7.1 Resultados dos Testes

7.2 Cumprimento de requisitos

 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.

8 Conclusão

8.1 Conclusão

8.2 Trabalhos Futuros

Bibliografia

[1] A. Limited, “ITIL ® Foundation ITIL 4 Edition 2,” 2019, Accessed: Nov. 25, 2025. [Online]. Available: https://www.axelos.com

[2] L. : Tso, “The Official Introduction to the ITIL Service Lifecycle,” 2007, Accessed: Nov. 25, 2025. [Online]. Available: www.tsoshop.co.uk

[3] “Site Reliability Engineering: How Google Runs Production Systems.” Accessed: Nov. 15, 2025. [Online]. Available: https://research.google/pubs/site-reliability-engineering-how-google-runs-production-systems/

[4] “17 Objetivos • ODS - BCSD Portugal.” Accessed: Nov. 25, 2025. [Online]. Available: https://ods.pt/ods/

[5] “What is ServiceNow? - ServiceNow.” Accessed: Nov. 25, 2025. [Online]. Available: https://www.servicenow.com/what-is-servicenow.html#what-we-do

[6] “Jira | Issue & Project Tracking Software | Atlassian.” Accessed: Nov. 25, 2025. [Online]. Available: https://www.atlassian.com/software/jira

[7] “Freshservice ITSM System | ITIL-aligned service desk software.” Accessed: Nov. 25, 2025. [Online]. Available: https://www.freshworks.com/freshservice/lp/home/

[8] “Zoho ManageEngine ServiceDesk Plus: AI-driven ITIL service desk.” Accessed: Nov. 25, 2025. [Online]. Available: https://www.manageengine.com/products/service-desk/

[9] “BMC Helix ITSM | BMC Helix.” Accessed: Nov. 25, 2025. [Online]. Available: https://www.helixops.ai/products/bmc-helix-itsm.html

[10] “Software de gerenciamento de serviços de TI para locais de trabalho modernos.” Accessed: Nov. 25, 2025. [Online]. Available: https://www.zendesk.com.br/employee-service/itsm/

[11] “IT Service Desk Software | SolarWinds”.

[12] “The future of AI and machine learning | Web Summit.” Accessed: Nov. 25, 2025. [Online]. Available: https://websummit.com/summaries/lis25/the-future-of-ai-and-machine-learning/

[13] R. A. Howard, “Information Value Theory,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 2, no. 1, pp. 22–26, 1966, doi: 10.1109/tssc.1966.300074.

[14] D. Golovin and A. Krause, “Adaptive Submodularity: Theory and Applications in Active Learning and Stochastic Optimization,” *Journal of Artificial Intelligence Research*, vol. 42, pp. 427–486, Nov. 2011, doi: 10.1613/JAIR.3278.

[15] H. Esfandiari, A. Karbasi, V. Mirrokni, M. Belkin, and S. Kpotufe, “Adaptivity in Adaptive Submodularity,” Jul. 21, 2021, *PMLR*. Accessed: Nov. 25, 2025. [Online]. Available: https://proceedings.mlr.press/v134/esfandiari21a.html

[16] C. Skaanning Hewlett -Packard, “A Knowledge Acquisition Tool for Bayesian-Network Troubleshooters,” *UNCERTAINTY IN ARTIFICIAL INTELLIGENCE PROCEEDINGS*, 2000.

[17] M. Aliannejadi, H. Zamani, F. Crestani, and W. B. Croft, "Asking Clarifying Questions in Open-Domain Information-Seeking Conversations," *SIGIR 2019 - Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 475–484, Jul. 2019, doi: 10.1145/3331184.3331265.

[18] "KCS v6 Practices Guide - Consortium for Service Innovation." Accessed: Nov. 25, 2025. [Online]. Available: https://library.serviceinnovation.org/KCS/KCS_v6/KCS_v6_Practices_Guide?

[19] "Partially observable Markov decision process - Wikipedia." Accessed: Nov. 25, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Partially_observable_Markov_decision_process?

Anexo 1 – Formulário de declaração de uso de ferramentas de inteligência Artificial

1. Utilização de IA

[ ] Não foram utilizadas ferramentas de IA na realização deste trabalho.
[X] Foram utilizadas ferramentas de IA na realização deste trabalho.

---

2. Ferramentas utilizadas

Assinalar todas as que se aplicam.

Assistência geral à escrita, análise ou ideação

[X] ChatGPT
[X] Microsoft Copilot
[ ] Gemini
[ ] Claude
[ ] Perplexity
[ ] Outras. Quais? _____

Assistência à programação / desenvolvimento

[X] GitHub Copilot

[ ] Claude

[X] OpenAI Codex
[ ] Cursor
[ ] Tabnine
[ ] Amazon CodeWhisperer / Amazon Q
[ ] Outras. Quais? _____

Geração de imagem / design / multimédia

[ ] DALL·E
[ ] Midjourney
[ ] Stable Diffusion
[ ] Canva AI / Magic Design
[ ] Outras. Quais? _____

Outros usos

[ ] Contexto: Ferramentas? _____

3. Fases do trabalho em que foi utilizada IA

[X] Planeamento do trabalho
[X] Pesquisa exploratória / levantamento inicial de informação
[ ] Documentação técnica
[X] Redação do relatório
[X] Desenho / modelação / arquitetura
[ ] Design / prototipagem / interface
[X] Geração de código
[X] Revisão / refatoração / debugging de código
[X] Criação de testes / casos de teste
[ ] Análise de resultados
[ ] Preparação de apresentação ou materiais auxiliares
[ ] Outros. Quais? _____

4. Tipo de utilização

Descrever sucintamente como a IA foi utilizada.

Exemplos: brainstorming, estruturação de secções, revisão linguística, sugestão de arquitetura, geração de exemplos, explicação de conceitos, geração parcial de código, correção de erros, criação de casos de teste, apoio ao design.

No desenvolvimento deste projeto, a IA foi utilizada como uma ferramenta de apoio e não como substituto do trabalho realizado. O seu uso foi mais relevante na sugestão de arquitetura e organização do back-end, tendo em conta a complexidade do sistema e a necessidade de estruturar corretamente os diferentes módulos (incidentes, mudanças, KB, KPIs,…).Foi também utilizada na geração parcial de código, sobretudo em funcionalidades ou conceitos não abordados diretamente em contexto académico, funcionando como suporte à aprendizagem. Adicionalmente, contribuiu para a identificação e correção de erros, debugging, criação de testes e validação de algumas decisões técnicas. Importa referir que, por se tratar de um projeto desenvolvido em contexto empresarial, existiram restrições ao uso de software que requer licenciamento, o que limitou algumas opções tecnológicas. Por exemplo, não foi possível utilizar Docker, tendo sido necessário recorrer ao Podman, o que implicou um esforço adicional de aprendizagem autónoma e nos obrigou a aprender a dar set up ao ambiente de execução sozinhos.

Por fim, a IA foi utilizada como apoio à redação e estruturação do relatório, permitindo melhorar a clareza e organização do documento, especialmente tendo em conta a

alteração do nosso tema de TFC, faltando cerca de uma semana da entrega do primeiro relatório, o que implicou uma adaptação rápida ao novo tema.

---

5. Partes do trabalho afetadas

Indicar as secções, componentes, módulos, ficheiros, entregáveis ou atividades que foram influenciados pelo uso de IA.

A IA foi utilizada como ferramenta de apoio e otimização de tempo, tendo impacto em várias fases do projeto. As áreas mais diretamente influenciadas incluem a definição e estruturação da arquitetura do sistema, os módulos do back-end (nomeadamente entidades, serviços e controladores), bem como a modelação da base de dados. Adicionalmente, teve impacto nas componentes relacionadas com debugging e correção de erros, na implementação e validação de lógica de negócio, e de forma significativa na criação de testes. Foi também utilizada no apoio à redação e organização do relatório, contribuindo para a sua clareza e estrutura.

---

6. Exemplos de *prompt*

Inserir exemplos de *prompt*, diferenciando por âmbito (enquadrado na questão 2) e fase (enquadrado na questão 4)

Prompt 1 (Análise Estrutural / Início do desenvolvimento): "Estou a desenvolver um back-end para gestão de Application Management Services (AMS) com entidades como Application, Incident, Change, KBArticle, etc. Qual a melhor forma de estruturar a arquitetura em Spring Boot (controllers, services, repositories) para garantir separação de responsabilidades e escalabilidade?"

Prompt 2 (Desenvolvimento / Lógica de Negócio): "Como posso calcular métricas como MTTR e taxa de recorrência de incidentes com base nos timestamps (openedAt, resolvedAt) e dados do ServiceContract armazenados na base de dados? Qual a melhor abordagem para implementar isto no service layer do KPI?"

Prompt 3 (Debugging / Correção de Erros): "Estou a receber erros de validação ao atualizar um incidente devido à ordem dos timestamps (firstResponseAt e resolvedAt). Como posso garantir que a lógica de validação respeita a ordem correta dos tempos?"

7. Validação, revisão e intervenção dos autores

Descrever que verificação, revisão, correção, adaptação ou reescrita foi realizada pelos autores.

Nota: se a IA tiver sido usada em código, testes, scripts, modelos, consultas, configurações ou outros artefactos técnicos, deve ser indicado de que forma os autores validaram o funcionamento e confirmaram a sua compreensão.

Durante o desenvolvimento, sobretudo na implementação de novos módulos do back-end e na integração entre componentes (entidades, serviços e controladores), surgiram alguns erros e inconsistências. A IA foi utilizada nestas situações como apoio à correção e adaptação do código, sempre com base na solução previamente implementada por nós. Na criação de testes, e tal como referido anteriormente, contribuiu também para otimizar o tempo disponível face aos prazos definidos. Ainda assim, todas as soluções sugeridas foram analisadas criticamente, testadas e validadas por nós e pelo orientador, através da execução dos endpoints, testes unitários e revisão da lógica implementada.

Em suma, com acompanhamento do professor orientador e das pessoas da empresa(CGI) que nos estão a ajudar a desenvolver o projeto, foram realizadas várias validações para garantir que as soluções adotadas estavam corretas, evitando a aceitação direta das respostas da IA. Sempre que existiam dúvidas ou inconsistências nas sugestões apresentadas, estas eram descartadas, recorrendo-se a pesquisa adicional sem recurso a IA.

8. Grau de utilização

[ ] Residual
[X] Moderado
[ ] Extensivo

[ ] Utilização homogénea
[X] Grau de uso diferenciado por fase ou componente de trabalho

Descrever sucintamente os diferentes usos.

Para definição da arquitetura do sistema, modelação de dados, desenvolvimento do back-end (APIs e lógica de negócio), aprendizagem de conceitos não abordados previamente, debugging e correção de erros, geração parcial de código para funcionalidades específicas, criação de testes e apoio na documentação (relatório).

9. Trabalhos em parceria

Protecção de dados confidenciais e recursos proprietários de parceiros

[X] O trabalho foi realizado em parceria com entidade externa ao DEISI

No caso da resposta anterior ser verdadeira, responder às seguintes questões:

[X] O parceiro tem regras para restringir submissão de dados

[X] As submissões validam aplicação de regras de tratamento de dados

[X] Foram implementados mecanismos para restringir a partilha de recursos proprietários

10. Declaração de responsabilidade

Ao assinarem a presente declaração, os autores declaram que:

- a informação acima é verdadeira e reflete o uso efetivo de ferramentas de IA na realização do trabalho;
- compreendem que a IA não substitui autoria nem responsabilidade académica;
- verificaram a validaram e veracidade das referências bibliográficas incluídas no relatório
- assumem integralmente a responsabilidade técnica, científica, ética e académica por todo o conteúdo submetido, incluindo texto, código, modelos, testes, imagens, diagramas e restantes artefactos entregues.

11. Identificação dos autores

Nome(s): Bernardo Ferreira | Duarte Cachata

Número(s): a22307944 | a22302828

Data: 09/04/2026

Assinatura(s): Bernardo Ferreira | Duarte Cachata

Glossário

LEI	Licenciatura em Engenharia Informática
LIG	Licenciatura em Informática de Gestão
TFC	Trabalho Final de Curso

 O presente trabalho está sujeito a um acordo de confidencialidade (NDA), impossibilitando a divulgação de dados sensíveis, código-fonte e informação interna da organização.