Politécnico
Castelo Branco
Escola Superior
de Tecnologia

Plataforma de Gestão de Aulas de Compensação

Dércio Simione Bernardo Domingos
Nº 20220103
Joseanne Nadine Maria do Amaral Gourgel
Nº 20230253

Orientadores
Mónica Isabel Teixeira da Costa
Bruno Miguel Gonçalves Matias

Trabalho de Projeto Final apresentado à Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco para cumprimento dos requisitos necessários à obtenção do grau de Licenciado em Engenharia Informática e Licenciado em Informática e Multimédia, realizado sob a orientação científica da professora Doutora Mónica Costa e coorientação do professor Bruno Miguel Gonçalves Matias, do Instituto Politécnico de Castelo Branco.

Junho, 2026

Composição do júri

Presidente do júri
Especialista, Luís Miguel Santos Silva de Ascensão Barata
Professor Adjunto Convidado da Escola Superior de Tecnologia de Castelo Branco

Vogais
Doutor, Eduardo Sabina dos Santos Valente
Professor Adjunto da Escola Superior de Tecnologia de Castelo Branco

Doutor, Mónica Isabel Teixeira da Costa
Professor Adjunto da Escola Superior de Tecnologia de Castelo Branco

Dedicatória

Este trabalho dedico primeiramente a Deus por me ajudar e dar forças em todos os momentos da minha vida, a minha querida avó e fonte de inspiração Margarida de Sá, que sempre acreditou em mim, serei eternamente grata, este trabalho é a prova que o seu investimento valeu a pena. Aos meus pais Ernesto e Civoduya Audácio, as minhas tias Horácia Gaspar e Cândida Rufino por me motivarem sempre, a Eliane, a minha avó Sónia, aos meus irmãos, professores, ao mentor e amigo Dércio Simione Domingos (Derone), amigos e demais familiares que sempre me apoiaram, incentivaram e contribuíram para chegar até aqui.

Joseanne Gourgel

Dedico este trabalho a Deus, pelo amparo e pela luz constante, e aos meus pais, cujo incentivo incansável fez deles a minha maior fonte de motivação. Dedico-o também a mim mesmo, pela resiliência e determinação demonstradas em cada etapa deste desafio.

Dércio Simione Domingos

Agradecimentos

Agradecemos, em primeiro lugar, aos nossos orientadores, pela disponibilidade, rigor científico e acompanhamento atento em todas as fases deste trabalho. Estendemos o agradecimento aos professores do curso, pelo conhecimento partilhado, pelas sugestões construtivas e pelo incentivo constante que elevaram a qualidade do projeto.

Às nossas famílias, pelo amor, paciência e apoio incondicional, e aos amigos e colegas, pela motivação diária, pela entreajuda e pelas discussões que enriqueceram a investigação e a modelação da solução, deixamos o nosso sincero obrigado.

Resumo

O presente projeto tem como finalidade o desenvolvimento da plataforma digital **Compensa+**, destinada à gestão de aulas de compensação no ensino superior. Este trabalho surge como resposta às limitações dos sistemas académicos tradicionais, onde este processo é frequentemente realizado de forma manual e descentralizada, recorrendo a e-mails e outros meios informais, o que conduz a dificuldades de organização, conflitos de horários e falta de transparência no acompanhamento das atividades.

Na fase inicial do projeto (Projeto I), foi realizada a análise e modelação do sistema com base na metodologia ICONIX, permitindo a definição estruturada dos requisitos funcionais e não funcionais, bem como a construção de modelos UML e de um protótipo não funcional que validou os fluxos de interação entre os diferentes perfis de utilizadores.

O Projeto II dá continuidade a esse trabalho, focando-se na implementação da solução proposta, transformando o modelo conceptual numa aplicação funcional. Esta fase inclui o desenvolvimento do backend, responsável pela gestão dos pedidos de compensação, verificação automática de disponibilidade e controlo de acessos, bem como a implementação de um frontend responsivo, alinhado com o protótipo previamente definido.

A plataforma suporta diferentes perfis de utilizadores, nomeadamente docentes, coordenadores de curso e administradores, permitindo a submissão, validação e acompanhamento de pedidos de compensação de forma centralizada. Integra ainda um calendário académico dinâmico, mecanismos de notificação automática e um sistema de registo auditável, promovendo maior eficiência, organização e transparência no processo.

Adicionalmente, o projeto contempla a validação da solução através de testes funcionais e de integração, garantindo o correto funcionamento das funcionalidades implementadas e a sua conformidade com os requisitos definidos.
Espera-se que a implementação da plataforma Compensa+ contribua para a redução do esforço administrativo, diminuição de conflitos de horários e melhoria da comunicação entre os intervenientes, proporcionando uma gestão mais eficiente e estruturada das aulas de compensação no contexto académico.

Palavras-chave: gestão académica; compensação de aulas; ICONIX; UML; calendário académico.

Abstract

This project aims to develop a digital platform called **Compensa+**, designed to manage make-up classes in higher education. The motivation for this work arises from the limitations of existing academic systems, where this process is often handled manually through emails and other informal methods, leading to organizational difficulties, scheduling conflicts, and a lack of transparency.

In the first phase of the project (Project I), the system was analyzed and modeled using the ICONIX methodology. This stage enabled the identification of functional and non-functional requirements, as well as the creation of UML models and a non-functional prototype to validate user interaction flows.

The second phase (Project II) focuses on the implementation of the proposed solution, transforming the conceptual model into a functional application. This includes the development of the backend, responsible for managing make-up class requests, validating availability, and handling access control, as well as the implementation of a responsive frontend aligned with the previously defined prototype.

The platform supports different user roles, namely lecturers, course coordinators, and administrators, allowing the submission, validation, and tracking of requests in a centralized environment. It also integrates a dynamic academic calendar, automatic notification mechanisms, and an auditable record of all actions, promoting greater efficiency, organization, and transparency.

Additionally, this phase includes system validation through functional and integration testing to ensure that the implemented features meet the defined requirements and perform reliably.

The expected outcome is a reduction in administrative workload, fewer scheduling conflicts, and improved communication among stakeholders, contributing to a more efficient and structured management of make-up classes in the academic context.

Keywords: academic management; make-up classes; ICONIX; UML; academic calendar.

Índice Geral

1. Introdução *1*

1.1. Objetivos **1**

1.2. Justificação **2**

1.3. Planeamento do projeto **3**

1.4. Organização do Relatório **4**

2. Arquitetura e Tecnologias *7*

2.1. Visão Geral da Arquitetura **7**

2.2. API Gateway **9**

2.3. Microserviços da Plataforma **10**

2.3.1. Microserviço *Identity* 10

2.3.2. Microserviço *Core* 11

2.3.3. Microserviço Notifications 12

2.3.4. Microserviço Compensa AI 13

2.3.5. Integração com modelos de linguagem natural 14

2.3.6. Mecanismos de fallback 15

2.3.7. Prompt Enginering 17

2.3.8. Papel e âmbito 18

2.3.9. Diretrizes conversacionais 18

2.3.10. Resolução conversacional de identificadores 18

2.3.11. Segurança e privacidade 19

2.3.12. Idioma e tom 19

2.3.13. Ferramentas e Function Calling 19

2.3.14. Cache de respostas com Redis 20

2.3.15. Integração com os microserviços internos 21

2.3.16. Métricas de desempenho, limites e custos do Compensa AI 22

2.4. Frontend Web **24**

2.5. Gestão de Dados e Comunicação **25**

2.5.1. PostgreSQL 26

2.5.2. *Redis* 26

2.5.3. RabbitMQ 27

2.6. Infraestrutura, Containerização e *Deploy* **28**

2.6.1. Ambiente de Desenvolvimento com *Docker Compose* 29

2.6.2. Orquestração em Produção com *Docker Swarm* 30

2.6.3. Definição de Réplicas e Escalabilidade 31

2.6.4. Balanceamento de Carga Interno 32

2.6.5. Atualização Contínua sem *Downtime* 32

2.6.6. Entrega do *Frontend* com *Nginx* 32

2.6.7. Síntese da Estratégia de Infraestrutura 33

2.7. Monitorização e Observabilidade **34**

2.7.1. OpenTelemetry 34

2.7.2. Prometheus 35

2.7.3. *Grafana* 35

2.7.4. *Loki* e Tempo 38

2.7.5. .NET Aspire Dashboard 38

2.7.6. *Portainer* 39

2.7.7. Estratégia de *Deploy* 41

2.7.8. Provisionamento com *Terraform* 41

2.7.9. Configuração com *Ansible* 42

2.7.10. Topologia de Máquinas Virtuais ... 42
2.7.11. Cluster Docker Swarm na Azure ... 43
2.7.12. Estratégia para o *Frontend* com *Nginx* ... 43
2.7.13. Pipeline de Deploy ... 44
2.8. Tecnologias Utilizadas ... 45
2.9. Considerações Finais ... 46
3. Implementação do Sistema ... 49
3.1. Organização do Desenvolvimento com *Kanban* ... 49
3.2. Organização Geral do Projeto ... 50
3.3. Relação entre o Projeto I e a Implementação ... 52
3.4. Implementação do *Frontend Web* ... 53
3.5. Implementação do Backend ... 69
3.6. Comunicação Assíncrona, Cache e Observabilidade na Implementação ... 88
3.7. Considerações Finais ... 89
4. Testes e Validação ... 91
4.1. Estratégia de Validação ... 91
4.2. Ambiente de Testes ... 92
4.3. Validação da Autenticação e Controlo de Acessos ... 93
4.4. Testes do Microserviço Core API ... 94
4.5. Validação da CompensaAI API ... 97
***4.6.* Testes *Unitários* e *Testes de Integração* ... 98**
4.7. Testes *End-to-End* com *Playwright* ... 105
4.8. Validação da Infraestrutura *Cloud* e *Deploy* ... 106
4.9. Resultados da Validação ... 110
4.10. Considerações Finais ... 112
5. Conclusão e Trabalho Futuro ... 114
5.1. Trabalho Futuro ... 116
Referências Bibliográficas ... 119

Índice de figuras

Figura 1 - Arquitetura geral da plataforma Compensa+. Fonte: elaboração própria. 8
Figura 2 - API Gateway e documentação centralizada dos serviços. Fonte: elaboração própria. 10
Figura 3 - Endpoints de autenticação e gestão de utilizadores. Fonte: elaboração própria. 11
Figura 4 - Endpoints principais da Core API. Fonte: elaboração própria. 12
Figura 5 - Processamento assíncrono de notificações. Fonte: elaboração própria. 13
Figura 6 - Classe AI_Coordinator responsável por gerir os modelos de linguagem natural. Fonte: elaboração própria. 14
Figura 7 - Configuração de leitura de variavel de ambiente do Microserviço Compensa AI. Fonte: elaboração própria. 15
Figura 8 - Fluxo de seleção de modelo e mecanismo de fallback do Microserviço Compensa AI. Fonte: elaboração própria. 16
Figura 9 - Monitorização do fallback do Compensa AI no Grafana, com redirecionamento de Gemini para OpenAI após erro de resposta. Fonte: elaboração própria. 17
Figura 10 - Instrução para os modelos de linguagem natural. Fonte: elaboração própria. 18
Figura 11 - Implementação da classe CacheService. Fonte: elaboração própria. 21
Figura 12 - Implementação da classe CoreApi. Fonte: elaboração própria. 22
Figura 13 - Assistente inteligente da plataforma Compensa+. Fonte: elaboração própria. 24
Figura 14 - Interface principal da plataforma Compensa+. Fonte: elaboração própria. 25
Figura 15 - Gestor de banco de dados Datagrip. Fonte: elaboração própria. 26
Figura 16 - Processamento de eventos de notificação. Fonte: elaboração própria. 27
Figura 17 - Monitorização da taxa de eventos RabbitMQ no Grafana. Fonte: elaboração própria. 28
Figura 18 - Serviços da plataforma executados em containers Docker. Fonte: elaboração própria. 29
Figura 19 - Ficheiro de configuração do Docker compose. Fonte: elaboração própria. 30
Figura 20 - Configuração do serviço Core API no ficheiro docker-stack.yml, com definição de 3 réplicas e política de atualização contínua sem interrupção de serviço (start-first). Fonte: elaboração própria. 31
Figura 21 - Comparação entre o ambiente de desenvolvimento (Docker Compose) e o ambiente de produção (Docker Swarm) da plataforma Compensa+. Fonte: elaboração própria. 33
Figura 22 - Dashboard de monitorização de métricas no Grafana. Fonte: elaboração própria. 36
Figura 23 - Visualização geral das métricas da aplicação no Grafana através do Prometheus. Fonte: elaboração própria. 37
Figura 24 - Dashboard de monitorização traces no Grafana. Fonte: elaboração própria. 37
Figura 25 - Email de alerta enviado pelo grafana em caso de problemas na plataforma. Fonte: elaboração própria. 38

Figura 26 - Monitorização dos serviços através do .NET Aspire Dashboard. Fonte: elaboração própria. 39
Figura 27 - Monitorização dos serviços através do .NET Aspire Dashboard. Fonte: elaboração própria. 39
Figura 28 - Painel de monitorização de containers via Portainer. Fonte: elaboração própria. 40
Figura 29 - Topologia de infraestrutura de produção da plataforma Compensa+ na Microsoft Azure. Fonte: elaboração própria. 44
Figura 30 - Stack tecnológica utilizada no desenvolvimento da plataforma Compensa+. Fonte: elaboração própria. 46
Figura 31 - Quadro Kanban utilizado para organização das tarefas de implementação no Jira. Fonte: elaboração própria. 50
Figura 32 - Estrutura geral do repositório da plataforma Compensa+. Fonte: elaboração própria. 51
Figura 33 - Estrutura do projeto frontend da plataforma Compensa+. Fonte: elaboração própria. 54
Figura 34 - Página de autenticação da plataforma Compensa+. Fonte: elaboração própria. 55
Figura 35 - Criação e gestão de utilizadores na plataforma Compensa+. Fonte: elaboração própria. 56
Figura 36 - Painel principal da plataforma Compensa+. Fonte: elaboração própria. 57
Figura 37 - Listagem de pedidos de compensação em grelha. Fonte: elaboração própria. 58
Figura 38 - Listagem de pedidos de compensação em tabela. Fonte: elaboração própria. 59
Figura 39 - Formulário de criação de pedido de compensação. Fonte: elaboração própria. 60
Figura 40 - Detalhes do pedido e ações de validação pelo coordenador. Fonte: elaboração própria. 61
Figura 41 - Calendário académico da plataforma Compensa+. Fonte: elaboração própria. 62
Figura 42 - Gestão de cursos, unidades curriculares e turmas. Fonte: elaboração própria. 63
Figura 43 - Gestão de salas de aula. Fonte: elaboração própria. 64
Figura 44 - Página de importação de turmas e horários. Fonte: elaboração própria. 65
Figura 45 - Pré-visualização da importação de turmas e horários. Fonte: elaboração própria. 66
Figura 46 - Confirmação e mapeamento dos dados importados. Fonte: elaboração própria. 67
Figura 47 - Página de notificações da plataforma Compensa+. Fonte: elaboração própria. 68
Figura 48 - Assistente inteligente da plataforma Compensa+. Fonte: elaboração própria. 69
Figura 49 - Estrutura em camadas dos microserviços backend. Fonte: elaboração própria. 70
Figura 50 - Configuração de rotas no API Gateway. Fonte: elaboração própria. 72
Figura 51 - Fluxo de autenticação por link mágico. Fonte: elaboração própria. 73
Figura 52 - Estrutura interna da Core API. Fonte: elaboração própria. 75

Figura 53 - Entidades de domínio implementadas na Core API. Fonte: elaboração própria. 76
Figura 54 - Organização dos serviços aplicacionais da Core API. Fonte: elaboração própria. 77
Figura 55 - Organização dos repositórios na Core API. Fonte: elaboração própria. 78
Figura 56 - Validação de conflitos de horário na Core API. Fonte: elaboração própria. 79
Figura 57 - Processamento de ficheiros HTML no serviço de importação de turmas e horários. Fonte: elaboração própria. 81
Figura 58 - Endpoints principais da Notifications API. Fonte: elaboração própria. 85
Figura 59 - Estrutura interna da CompensaAI API. Fonte: elaboração própria. 86
Figura 60 - Fluxo de funcionamento do assistente inteligente. Fonte: elaboração própria. 87
Figura 61 - Publicação e consumo de eventos entre microserviços. Fonte: elaboração própria. 88
Figura 62 - Serviços da plataforma em execução durante a validação. Fonte: elaboração própria. 93
Figura 63 - Validação do fluxo de autenticação da plataforma. Fonte: elaboração própria. 94
Figura 64 - Validação dos endpoints principais da Core API. Fonte: elaboração própria. 95
Figura 65 - Validação da criação de pedido de compensação. Fonte: elaboração própria. 96
Figura 66 - Validação das regras de conflito de horário. Fonte: elaboração própria. 97
Figura 67 - Validação do assistente inteligente da plataforma Compensa+. Fonte: elaboração própria. 98
Figura 68 - Execução com sucesso dos testes automatizados do microserviço Core API. Fonte: elaboração própria 100
Figura 69 - Exemplo de código de teste unitário em .NET com xUnit. Fonte: elaboração própria. 101
Figura 70 - Exemplo de código de teste de integração em .NET com xUnit e Moq. Fonte: elaboração própria. 102
Figura 71 - Execução dos testes automatizados e validação da cobertura de código do microserviço Core API. Fonte: elaboração própria. 103
Figura 72 - Resultado do teste de carga da Core API com k6, utilizando 5 utilizadores virtuais durante 60 segundos. Fonte: elaboração própria. 105
Figura 73 - Execução de testes end-to-end com Playwright. Fonte: elaboração própria. 106
Figura 74 - Topologia da infraestrutura de produção da plataforma Compensa+ na Microsoft Azure. Fonte: elaboração própria. 108
Figura 75 - Validação do provisionamento da infraestrutura com Terraform. Fonte: elaboração própria. 109
Figura 76 - Configuração automatizada das máquinas virtuais com Ansible. Fonte: elaboração própria. 110

Lista de tabelas

Tabela 1 - Cronograma de desenvolvimento do Projeto II. Fonte: elaboração própria. 4
Tabela 2 - Métricas de desempenho, limites e custos do Compensa AI. Fonte: elaboração própria. 23
Tabela 3 - Tabela de configuração de replicas de plataforma compensa+. Fonte: elaboração própria. 31
Tabela 4 - Topologia de Máquinas Virtuais. Fonte: elaboração própria. 42
Tabela 5 - Tecnologias utilizadas na plataforma Compensa+. Fonte: elaboração própria. 45
Tabela 6 - Relação entre casos de uso do Projeto I e funcionalidades implementadas no Projeto II. Fonte: elaboração própria. 52
Tabela 7 - Organização em camadas dos microserviços backend. Fonte: elaboração própria. 70
Tabela 8 - Resumo dos cenários de validação. Fonte: elaboração própria. 110

Lista de abreviaturas, siglas e acrónimos

AI - Artificial Intelligence
ANF - Anfiteatro
API - Application Programming Interface
ASP.NET - Active Server Pages .NET
AWS - Amazon Web Services
BOM - Byte Order Mark
CI/CD - Continuous Integration / Continuous Delivery
CORS - Cross-Origin Resource Sharing
CPU - Central Processing Unit
CRUD - Create, Read, Update and Delete
CSS - Cascading Style Sheets
DDD - Domain-Driven Design
DOM - Document Object Model
DTO - Data Transfer Object
E2E - End-to-End
GUID - Globally Unique Identifier
GPT - Generative Pre-trained Transformer
HMR - Hot Module Replacement
HTML - HyperText Markup Language
HTTP - HyperText Transfer Protocol
IA - Inteligência Artificial
IaC - Infrastructure as Code
ICONIX - Processo de modelação orientado a casos de uso
I/O - Input/Output
IP - Internet Protocol
IPCB - Instituto Politécnico de Castelo Branco
ISO - International Organization for Standardization
JSON - JavaScript Object Notation
JWT - JSON Web Token
LAB - Laboratório
LMS - Learning Management System
NSG - Network Security Group
ORM - Object-Relational Mapping
RBAC - Role-Based Access Control
REST - Representational State Transfer
SGBD - Sistema de Gestão de Base de Dados
SHA-256 - Secure Hash Algorithm 256-bit
SPA - Single Page Application
SQL - Structured Query Language
T - Teórica
TLS - Transport Layer Security
TP - Teórico-Prática

TTFT - Time to First Token
UI - User Interface
UI/UX - User Interface / User Experience
UML - Unified Modeling Language
UNESCO - United Nations Educational, Scientific and Cultural Organization
URL - Uniform Resource Locator
UTF - Unicode Transformation Format
UX - User Experience
VM - Virtual Machine
VNet - Virtual Network
YARP - Yet Another Reverse Proxy

1.Introdução

A transformação digital tem assumido um papel cada vez mais relevante no ensino superior, contribuindo para a modernização dos processos académicos e administrativos (UNESCO, 2023). Neste contexto, os sistemas de informação permitem automatizar tarefas, melhorar a comunicação entre intervenientes e aumentar a rastreabilidade das atividades realizadas (Laudon & Laudon, 2022).

Apesar desta evolução, a gestão de aulas de compensação continua, em muitos casos, a ser realizada através de correio eletrónico, folhas de cálculo ou outros mecanismos informais. Esta forma de gestão pode dificultar a coordenação entre docentes, coordenadores e serviços administrativos, aumentando a possibilidade de conflitos de horários, duplicação de informação e erros operacionais (Selwyn, 2016).

No âmbito do Projeto I realizou-se a análise e modelação da plataforma Compensa+, recorrendo à metodologia ICONIX. Esta fase permitiu identificar os requisitos funcionais e não funcionais, desenvolver modelos UML e construir um protótipo não funcional para validar os principais fluxos de interação do sistema (Rosenberg & Stephens, 2007).

Dando continuidade a esse trabalho, o Projeto II centra-se na implementação da plataforma, transformando o modelo conceptual numa aplicação funcional. A solução desenvolvida pretende centralizar e automatizar a gestão de aulas de compensação, permitindo melhorar a organização das atividades letivas, reduzir erros administrativos e aumentar a transparência no acompanhamento dos pedidos.

1.1. Objetivos

Objetivo Geral

Desenvolver e implementar a plataforma Compensa+, uma solução web destinada à gestão centralizada, automatizada e rastreável das aulas de compensação no ensino superior, permitindo a submissão, validação, acompanhamento e organização dos pedidos de forma eficiente, segura e integrada com os horários, salas, turmas e perfis de utilizador.

Objetivos Específicos

- Implementar as funcionalidades definidas na fase de análise e modelação realizada no Projeto I;

- Desenvolver uma interface web responsiva e intuitiva, alinhada com os perfis de utilizador da plataforma, nomeadamente Administrador, Coordenador e Docente;
- Implementar os microserviços responsáveis pela autenticação, gestão de pedidos de compensação, gestão académica, notificações e apoio inteligente ao utilizador;
- Garantir mecanismos de autenticação, autorização e controlo de permissões, assegurando que cada utilizador acede apenas às funcionalidades adequadas ao seu perfil;
- Implementar a submissão, consulta, aprovação, rejeição e acompanhamento dos pedidos de compensação de aulas;
- Integrar regras de validação de disponibilidade e deteção de conflitos, considerando horários, salas, turmas, docentes e calendário académico;
- Desenvolver o módulo de importação automática de turmas e horários a partir de ficheiros HTML, permitindo a criação ou atualização de dados académicos na plataforma;
- Implementar mecanismos de notificação para informar os utilizadores sobre ações relevantes, como criação, aprovação ou rejeição de pedidos;
- Integrar um assistente inteligente, através da CompensaAI API, para apoiar os utilizadores na consulta de informação e no esclarecimento de dúvidas relacionadas com a plataforma;
- Preparar a infraestrutura de execução e deploy da solução, recorrendo a Docker, Docker Swarm, Microsoft Azure, Terraform e Ansible;
- Validar a plataforma através de testes unitários, testes de integração, testes funcionais, testes de API, testes end-to-end e validação da infraestrutura;

1.2. Justificação

A gestão eficiente dos recursos académicos constitui um desafio relevante nas instituições de ensino superior, sobretudo em processos que envolvem docentes, salas, turmas, horários e validações por parte da coordenação. Os sistemas de informação permitem apoiar estes processos, reduzindo a dispersão de dados, melhorando a comunicação entre intervenientes e aumentando a rastreabilidade das atividades realizadas (Laudon & Laudon, 2022). No caso das aulas de compensação, este processo pode tornar-se complexo quando é realizado através de emails, folhas de cálculo ou outros mecanismos informais, uma vez que aumenta a possibilidade de conflitos de horário, duplicação de informação e falta de controlo sobre o estado dos pedidos.

Embora existam sistemas académicos e plataformas de apoio ao ensino, como o Fénix/FenixEdu ou o Moodle, estas soluções não estão especificamente orientadas para a gestão completa de aulas de compensação. O FenixEdu apresenta-se como uma plataforma modular para gestão académica e administrativa no ensino

superior, disponibilizando funcionalidades ligadas ao contexto institucional e académico (FenixEdu, 2026). O Moodle, por sua vez, é uma plataforma LMS orientada para a criação e gestão de ambientes de aprendizagem, cursos, conteúdos e atividades pedagógicas (Moodle, 2025). Apesar da sua utilidade, estas plataformas não respondem, de forma específica, ao fluxo completo de submissão, validação, aprovação, rejeição, acompanhamento e rastreabilidade dos pedidos de compensação de aulas.

Neste contexto, a plataforma Compensa+ surge como uma solução direcionada para este processo específico. A plataforma permite centralizar os pedidos de compensação, associá-los a cursos, turmas, unidades curriculares, salas e calendário académico, bem como controlar permissões por perfil de utilizador. Além disso, disponibiliza notificações, importação de horários e registo das ações realizadas, contribuindo para maior organização e transparência.

A implementação da Compensa+ justifica-se, assim, pela necessidade de reduzir o esforço administrativo, minimizar conflitos de horários, melhorar a comunicação entre docentes e coordenadores e garantir maior controlo sobre o estado dos pedidos. Do ponto de vista académico, o projeto permitiu ainda aplicar conhecimentos de engenharia de software, desenvolvimento web, bases de dados, microserviços, testes, containerização e automação de infraestrutura (Sommerville, 2016; Pressman & Maxim, 2020).

1.3. Planeamento do projeto

O desenvolvimento do Projeto II teve como base o trabalho realizado no Projeto I, no qual foram definidos os requisitos, a modelação do sistema e o protótipo não funcional da plataforma Compensa+. Nesta segunda fase, o foco centra-se na implementação técnica da solução proposta, bem como na sua validação e avaliação.

O planeamento do projeto foi estruturado de forma a garantir uma evolução progressiva e organizada das atividades, permitindo transformar o modelo

conceptual numa aplicação funcional. Para tal, o trabalho foi dividido em várias etapas principais:

- Implementação do Backend: desenvolvimento da lógica de negócio da aplicação, incluindo a gestão de pedidos de compensação, validação de disponibilidade, controlo de acessos e notificações;
- Implementação do Frontend: construção da interface do utilizador com base no protótipo definido, garantindo usabilidade e responsividade;
- Integração dos Componentes: ligação entre frontend e backend, assegurando o correto funcionamento do sistema como um todo;
- Testes e Validação: realização de testes funcionais e de integração para garantir a conformidade com os requisitos;
- Documentação: elaboração do relatório final e registo das decisões técnicas adotadas.

De forma a organizar temporalmente estas atividades, foi definido o seguinte cronograma de desenvolvimento:

Tabela 1 - Cronograma de desenvolvimento do Projeto II. Fonte: elaboração própria.

Tarefas	Março	Abril	Maio	Junho
Planeamento e preparação do ambiente	✓			
Implementação do Backend	✓	✓	✓	✓
Implementação do Frontend		✓	✓	✓
Integração do sistema		✓	✓	✓
Testes e validação			✓	✓
Ajustes e melhorias			✓	✓
Redação do relatório	✓	✓	✓	✓

Este cronograma permitiu uma distribuição equilibrada das tarefas ao longo do tempo, garantindo que cada fase do projeto fosse desenvolvida de forma estruturada e alinhada com os objetivos definidos. A sobreposição de algumas atividades, como a redação do relatório e os testes, permitiu acompanhar continuamente o progresso do sistema e ajustar eventuais inconsistências durante o desenvolvimento.

1.4. Organização do Relatório

O presente relatório encontra-se organizado em vários capítulos, de forma a apresentar de maneira clara e estruturada todas as fases do Projeto II, desde a introdução até às conclusões finais.

- **Capítulo 1 - Introdução:** apresenta o enquadramento do projeto, os seus objetivos, a justificação da sua realização, bem como o planeamento e a organização do relatório;
- **Capítulo 2 - Arquitetura e Tecnologias:** descreve a arquitetura do sistema implementado, incluindo a divisão entre frontend e backend, bem como as tecnologias e ferramentas utilizadas no desenvolvimento da plataforma;
- **Capítulo 3 - Implementação do Sistema:** apresenta o desenvolvimento das principais funcionalidades da aplicação, incluindo a gestão de pedidos de compensação, validação de horários, notificações e controlo de acessos;
- **Capítulo 4 - Testes e Validação:** descreve os testes realizados para verificar o correto funcionamento do sistema, bem como a avaliação dos resultados obtidos;
- **Capítulo 5 - Conclusão e Trabalho Futuro:** sintetiza os principais resultados do projeto, as dificuldades encontradas e possíveis melhorias ou evoluções futuras da plataforma.

Esta organização permite uma leitura progressiva e coerente do trabalho desenvolvido, facilitando a compreensão das diferentes etapas do projeto e das soluções implementadas.

2. Arquitetura e Tecnologias

A arquitetura de software constitui um dos elementos fundamentais no desenvolvimento de sistemas modernos, uma vez que influencia diretamente aspetos como desempenho, escalabilidade, segurança, manutenibilidade e capacidade de evolução da aplicação. Uma arquitetura bem definida permite organizar os componentes do sistema de forma estruturada, reduzindo a complexidade e promovendo uma separação clara de responsabilidades (Sommerville, 2016).

A plataforma Compensa+ foi concebida com base nos requisitos identificados durante a fase de análise e modelação realizada no Projeto I, na qual foi utilizada a metodologia ICONIX para definição dos casos de uso, modelo de domínio, diagramas UML e fluxos de interação entre os diferentes utilizadores (Rosenberg & Stephens, 2007). Esta base conceptual permitiu orientar a fase de implementação, garantindo que a solução técnica estivesse alinhada com os objetivos funcionais previamente definidos.

De forma a garantir modularidade, escalabilidade e facilidade de manutenção, optou-se por uma arquitetura baseada em microserviços. Segundo Newman (Newman, 2021), este tipo de arquitetura permite decompor uma aplicação em serviços independentes, cada um responsável por uma área específica do domínio de negócio, facilitando o desenvolvimento, a manutenção e a evolução do sistema.

No caso da Compensa+, a aplicação foi concebida como um ecossistema distribuído, composto por diferentes serviços especializados, nomeadamente API Gateway, Identity API, Core API, Notifications API e Compensa AI API. Esta separação de responsabilidades contribui para reduzir o acoplamento entre componentes e facilitar a evolução independente dos vários módulos do sistema (Fowler, 2002).

A arquitetura adotada combina comunicação síncrona, baseada em APIs REST, e comunicação assíncrona, baseada em eventos. A comunicação REST é adequada para operações em que o cliente necessita de uma resposta imediata, enquanto a comunicação orientada a eventos permite processar tarefas de segundo plano sem bloquear o fluxo principal da aplicação (Fowler, 2002).

2.1. Visão Geral da Arquitetura

A plataforma Compensa+ segue uma arquitetura baseada em microserviços descentralizados, estruturada para permitir que cada componente do sistema tenha responsabilidades bem definidas e possa evoluir de forma independente. Esta

abordagem permite maior flexibilidade na implementação, facilidade de manutenção e possibilidade de escalar componentes específicos conforme as necessidades do sistema (Newman, 2021).

A arquitetura da plataforma é composta pelos seguintes componentes principais:

- *Frontend Web* em *ReactJS*;
- *API Gateway;*
- *Identity API;*
- *Core API;*
- *Notifications API;*
- *Compensa AI API;*
- *PostgreSQL;*
- *Redis;*
- *RabbitMQ;*
- Plataforma de Observabilidade;
- *Docker e Docker Compose*.

Cada componente desempenha um papel específico no funcionamento da aplicação. O *frontend* é responsável pela interação com o utilizador; o *API Gateway* atua como ponto de entrada único; os microserviços tratam da autenticação, lógica de negócio, notificações e inteligência artificial; e os serviços de infraestrutura suportam a persistência, cache, mensageira e monitorização do sistema.

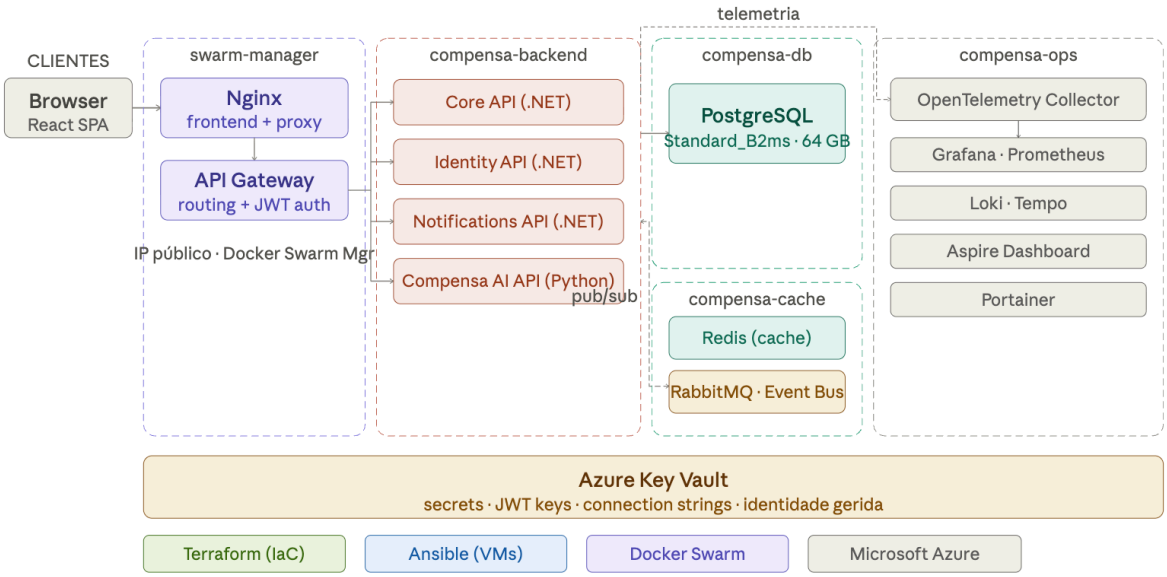


Figura 1 - Arquitetura geral da plataforma Compensa+. Fonte: elaboração própria.

Como ilustrado na Figura 1, o utilizador interage com a plataforma através da interface *web*. As requisições são encaminhadas para o *API Gateway*, que direciona cada pedido para o Microserviço correspondente. A *Core API* concentra a lógica principal do negócio, enquanto a *Identity API* gere autenticação e

permissões. A *Notifications API* reage a eventos produzidos pelo sistema, e a *CompensaAI* API fornece funcionalidades de apoio inteligente ao utilizador. Esta organização permite construir uma aplicação modular, escalável e preparada para futuras integrações.

2.2. API Gateway

O API Gateway constitui o ponto único de entrada da plataforma Compensa+. Em arquiteturas baseadas em microserviços, este padrão permite simplificar a comunicação entre o cliente e os serviços internos, centralizando responsabilidades como roteamento, autenticação, agregação de documentação e controlo de tráfego (Richardson, 2018).

Na Compensa+, o *API Gateway* foi implementado com recurso ao YARP (*Yet Another Reverse Proxy*), uma tecnologia da *Microsoft* destinada à criação de reverse *proxies* flexíveis e configuráveis. Este componente é responsável por receber as requisições provenientes do *frontend* e encaminhá-las para o microserviço adequado, de acordo com o prefixo da rota.

Entre as principais responsabilidades do *API Gateway* destacam-se:

- Encaminhamento de pedidos para os microserviços internos;
- Centralização da documentação *OpenAPI/Swagger*;
- Propagação de *tokens* de autenticação;
- Transformação e encaminhamento de *headers*;
- Organização do acesso externo ao ecossistema de serviços.

Desta forma, o API Gateway contribui para uma maior organização da comunicação entre o cliente e o *backend*, evitando que o *frontend* tenha de conhecer diretamente a localização de cada microserviço.

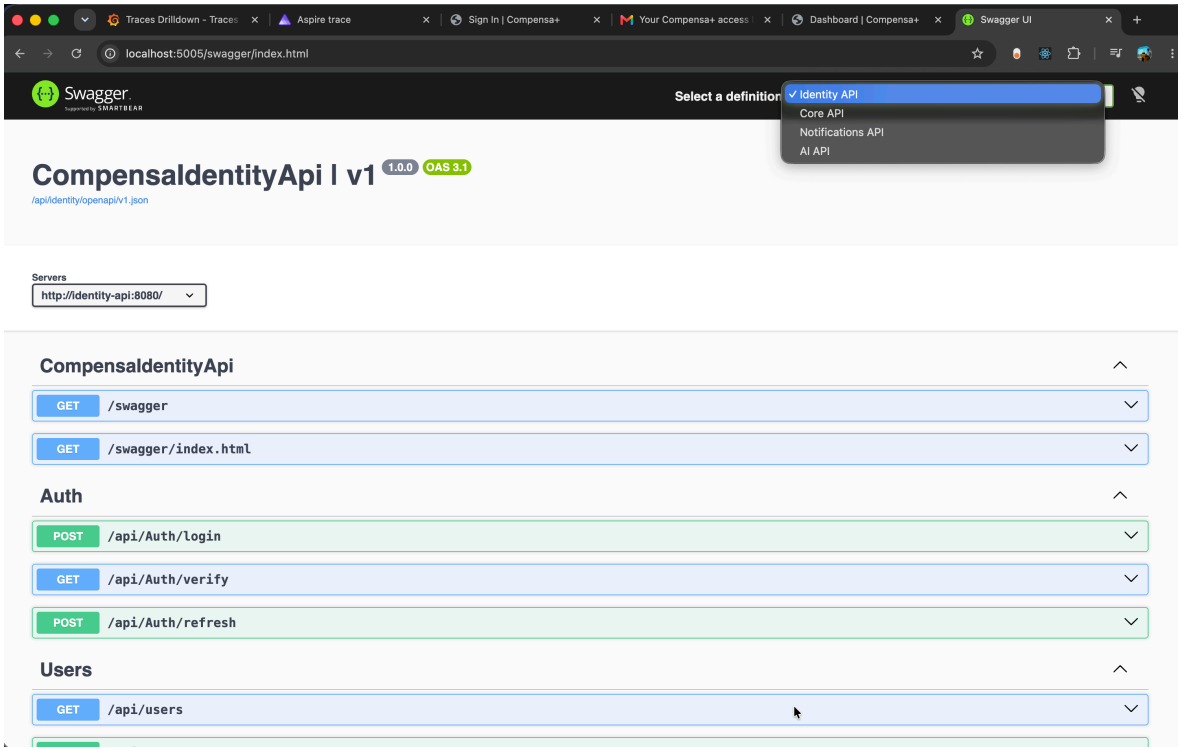


Figura 2 - API Gateway e documentação centralizada dos serviços. Fonte: elaboração própria.

2.3. Microserviços da Plataforma

A plataforma Compensa+ está organizada em microserviços, cada um responsável por um conjunto específico de funcionalidades. Esta separação permite reduzir a dependência entre componentes, facilitar a manutenção e tornar o sistema mais escalável (Newman, 2021). A seguir são apresentados os principais microserviços que compõem o sistema.

2.3.1. Microserviço *Identity*

O microserviço *Identity* é o microserviço responsável pela autenticação, autorização e gestão dos utilizadores da plataforma. A segurança é um requisito fundamental em sistemas distribuídos, sendo essencial garantir que apenas utilizadores autenticados e autorizados consigam aceder aos recursos protegidos (Microsoft, 2025).

Este serviço utiliza *ASP.NET Core Identity*, JWT (*JSON Web Tokens*) e *Entity Framework Core*. A autenticação baseada em *tokens* permite que o utilizador aceda aos diferentes serviços da plataforma de forma segura, sem necessidade de manter uma sessão tradicional no servidor.

As principais responsabilidades da *Identity API* são:

- Registo e gestão de utilizadores;

- Autenticação de utilizadores;
- Emissão e renovação de tokens JWT;
- Gestão de roles e permissões;
- Suporte a *Magic Links*;
- Gestão dos perfis *Admin, Coordinator e Teacher*.

A utilização de RBAC *(Role-Based Access Control)* permite controlar as ações disponíveis para cada perfil de utilizador. Assim, por exemplo, um docente pode submeter pedidos de compensação, enquanto um coordenador pode aprovar ou rejeitar esses pedidos.

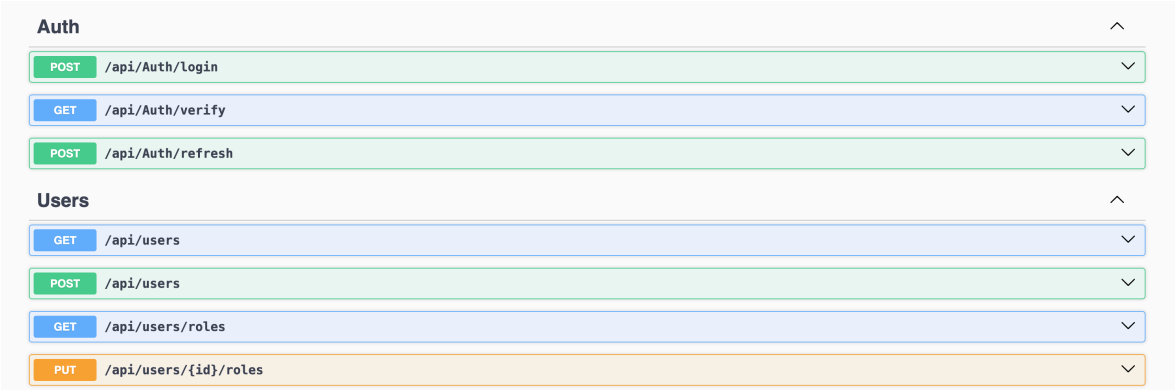


Figura 3 - Endpoints de autenticação e gestão de utilizadores. Fonte: elaboração própria.

2.3.2. Microserviço *Core*

O microserviço *Core* representa o núcleo funcional da plataforma Compensa+. Este microserviço concentra as principais regras de negócio relacionadas com a gestão académica e administrativa, sendo responsável pelo tratamento dos pedidos de compensação e pela organização dos dados centrais do sistema.

Segundo Evans (Evans, 2003), a correta modelação do domínio é essencial em sistemas empresariais, uma vez que permite representar com clareza os conceitos centrais da aplicação e as respetivas regras de negócio. Na Compensa+, *a Core API* foi estruturada tendo em conta entidades fundamentais do domínio académico.

Entre as principais entidades geridas por este serviço destacam-se:

- *Course*: representa os cursos existentes na instituição;
- *AcademicYear*: define o ano letivo ativo;
- *UserUnitAssignment*: associa utilizadores às respetivas unidades orgânicas;
- *CompensationRequest*: representa o pedido de compensação de aula.

As principais responsabilidades da *Core API* incluem:

- Gestão de cursos;
- Gestão de anos letivos;

- Submissão de pedidos de compensação;
- Aprovação e rejeição de pedidos;
- Consulta e filtragem de pedidos;
- Geração de métricas para o *dashboard*;
- Integração com Redis para cache de dados frequentemente consultados.

A *Core API* é, portanto, o componente responsável por materializar as regras principais do sistema, assegurando que os pedidos de compensação são registados, validados e acompanhados corretamente.

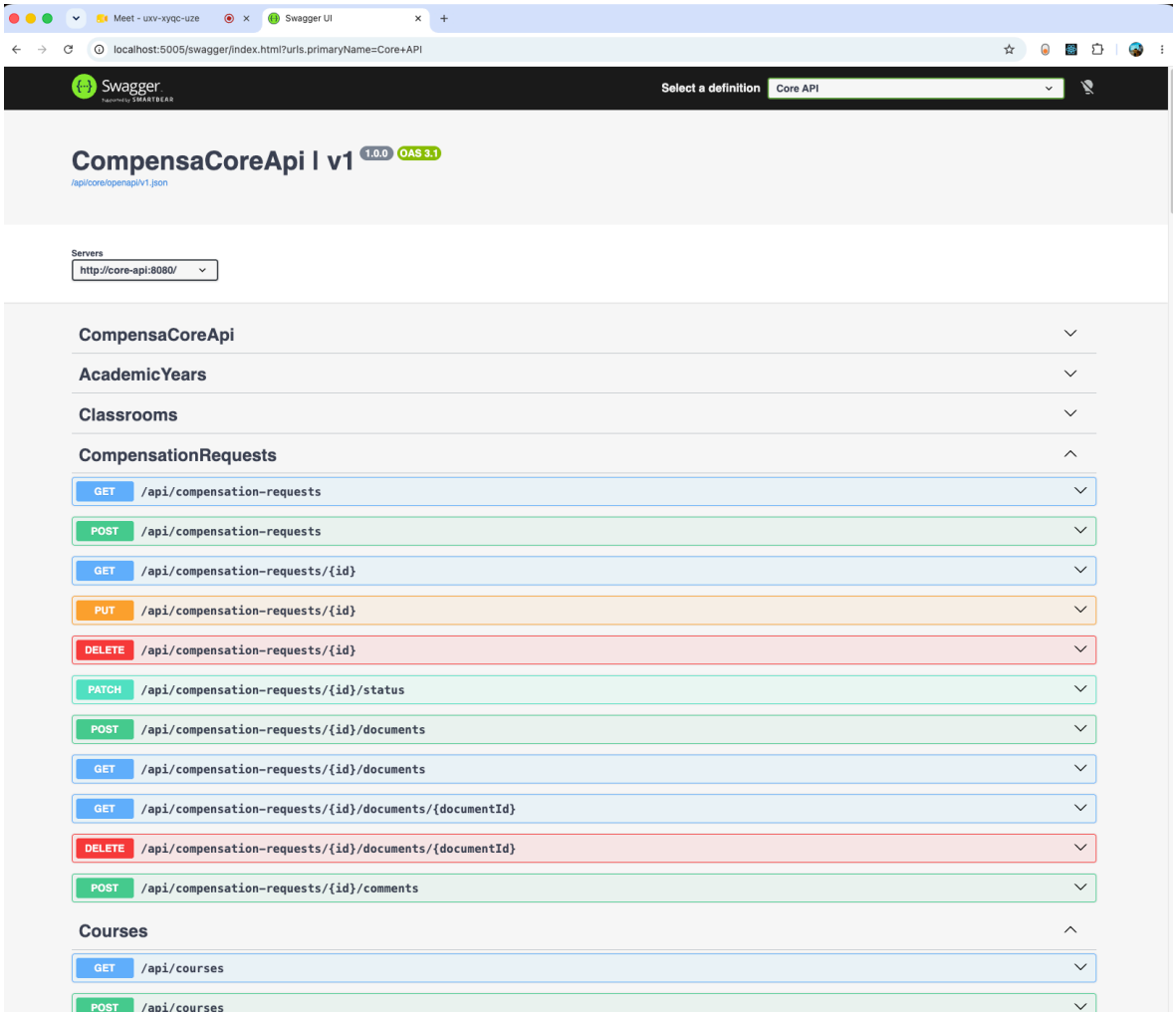


Figura 4 - Endpoints principais da Core API. Fonte: elaboração própria.

2.3.3. Microserviço Notifications

O Microserviço *Notifications é* responsável pelo processamento de notificações da plataforma. Este serviço foi concebido de forma independente para garantir que o envio de notificações não interfere diretamente com o funcionamento principal da *Core API*.

A comunicação entre a *Core API* e a *Notifications API* é realizada através de eventos publicados no RabbitMQ. Este modelo segue o *paradigma Event-Driven Architecture,* no qual os serviços comunicam através de eventos assíncronos. Segundo Hohpe e Woolf (Hohpe, 2003), este tipo de integração permite reduzir o acoplamento entre componentes e aumentar a tolerância a falhas em sistemas distribuídos.

Entre os eventos tratados pela *Notifications API* destacam-se:

- *RequestCreatedEvent;*
- *RequestStatusUpdatedEvent.*

As principais funcionalidades deste Microserviço incluem:

- Envio de notificações por email;
- Criação de notificações internas na aplicação;
- Registo do histórico de notificações;
- Marcação de notificações como lidas;
- Processamento assíncrono de eventos.

Esta abordagem permite que, mesmo que o serviço de notificações esteja temporariamente indisponível, as mensagens permaneçam na fila e sejam processadas assim que o serviço volte a estar operacional.

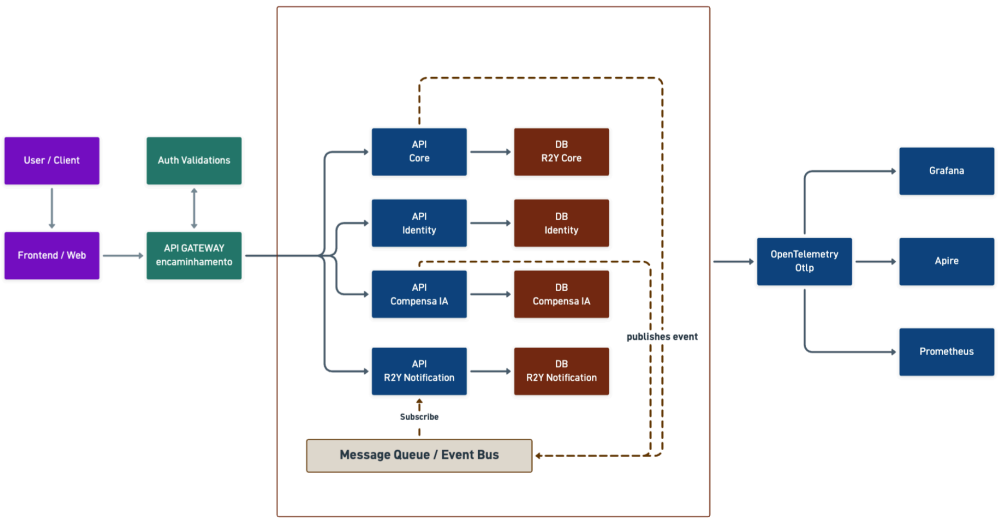


Figura 5 - Processamento assíncrono de notificações. Fonte: elaboração própria.

2.3.4. Microserviço Compensa AI

O Microserviço *Compensa AI* constitui o módulo de inteligência artificial da plataforma. Este serviço foi desenvolvido com recurso a *Python com recurso ao Framework FastAPI*, sendo responsável por fornecer apoio inteligente aos utilizadores durante a submissão, análise e validação de pedidos de compensação.

A Compensa AI expõe endpoints REST consumidos pelos outros microserviços via API Gateway, o que reforça a coerência com a arquitetura geral.

A utilização de *FastAPI* permite criar *APIs* assíncronas de elevado desempenho, adequadas para operações de *I/O* e integração com serviços externos (FastAPI, 2026). No contexto do Compensa+, a *AI API* comunica com modelos avançados de linguagem natural, permitindo disponibilizar um assistente inteligente capaz de responder a dúvidas e apoiar a análise dos pedidos.

2.3.5. Integração com modelos de linguagem natural

O Compensa AI integra dois fornecedores externos de modelos de linguagem natural, geridos pela classe *AI_Coordinator*. O modelo principal é o GPT-4.1-mini da OpenAI, selecionado pela sua capacidade de compreensão de linguagem natural em português, de raciocínio contextual e de suporte a function calling estruturado. O modelo alternativo é o Gemini Flash da Google, reservado para situações de indisponibilidade do modelo principal, pelo seu menor custo operacional e latência reduzida (Google, 2024; OpenAI, 2024).

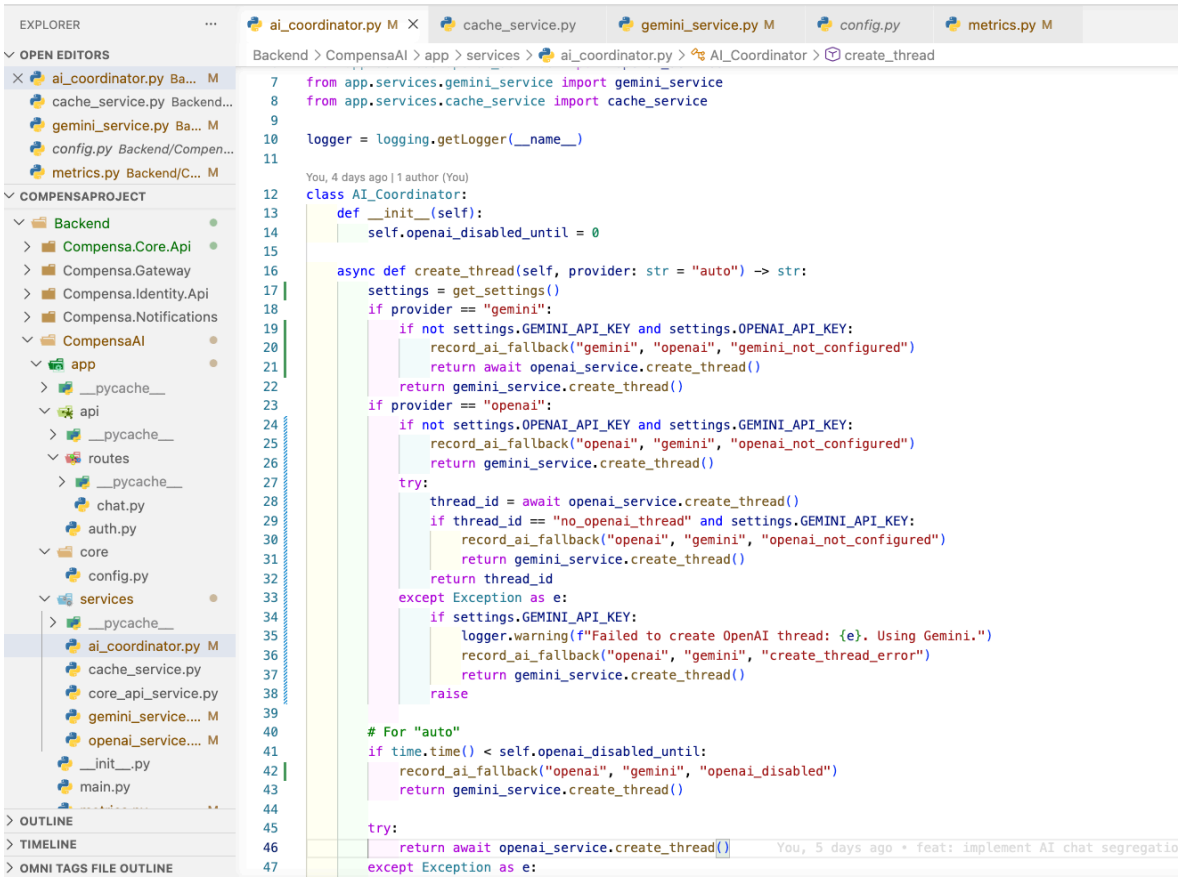


Figura 6 - Classe AI_Coordinator responsável por gerir os modelos de linguagem natural. Fonte: elaboração própria.

A configuração de ambos os modelos é centralizada na classe Settings, implementada com pydantic-settings e carregada a partir de variáveis de ambiente definidas no ficheiro .env. As chaves de API (OPENAI_API_KEY, GEMINI_API_KEY), os endereços dos serviços internos (CORE_API_URL, NOTIFICATIONS_API_URL) e os parâmetros de autenticação JWT são injetados em tempo de execução, sem qualquer valor sensível fixo no código-fonte.

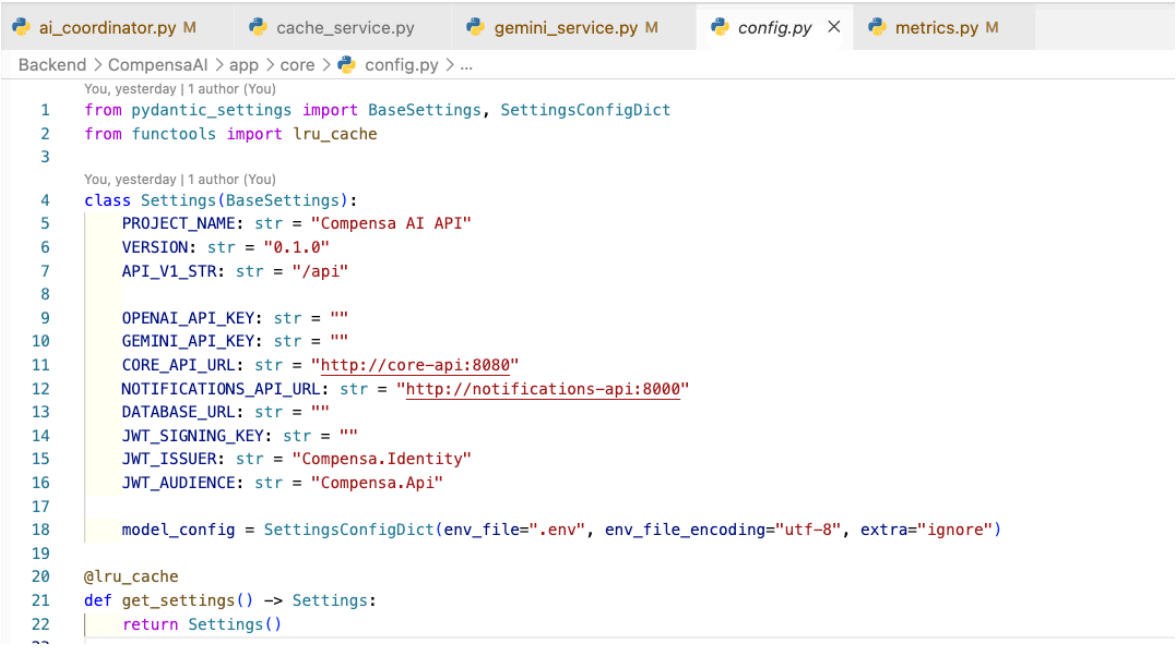


Figura 7 - Configuração de leitura de variavel de ambiente do Microserviço Compensa AI. Fonte: elaboração própria.

2.3.6.Mecanismos de fallback

A continuidade do serviço é assegurada através de um mecanismo de fallback automático implementado na classe *AI_Coordinator*, especificamente no método *create_thread*. Este método suporta três modos de operação através do parâmetro provider: auto (comportamento padrão), openai (forçar modelo principal) e gemini (forçar modelo alternativo).

No modo auto, a lógica de seleção segue a seguinte sequência: em primeiro lugar, o coordenador verifica se o OpenAI se encontra temporariamente desativado através do atributo openai_disabled_until. Este mecanismo de circuit breaker evita chamadas repetidas a um serviço que falhou recentemente, redirecionando imediatamente o tráfego para o Gemini enquanto o intervalo de desativação não expirar. Se o OpenAI estiver disponível, é feita uma tentativa de criação de thread. Caso a chamada falhe por exceção ou devolva o valor sentinela no_openai_thread, o sistema regista o evento através da função record_ai_fallback que alimenta as métricas de observabilidade e comuta automaticamente para o Gemini Flash. Apenas quando ambos os modelos falham é devolvido um erro ao cliente. O diagrama de sequência deste fluxo é apresentado na Figura 8.

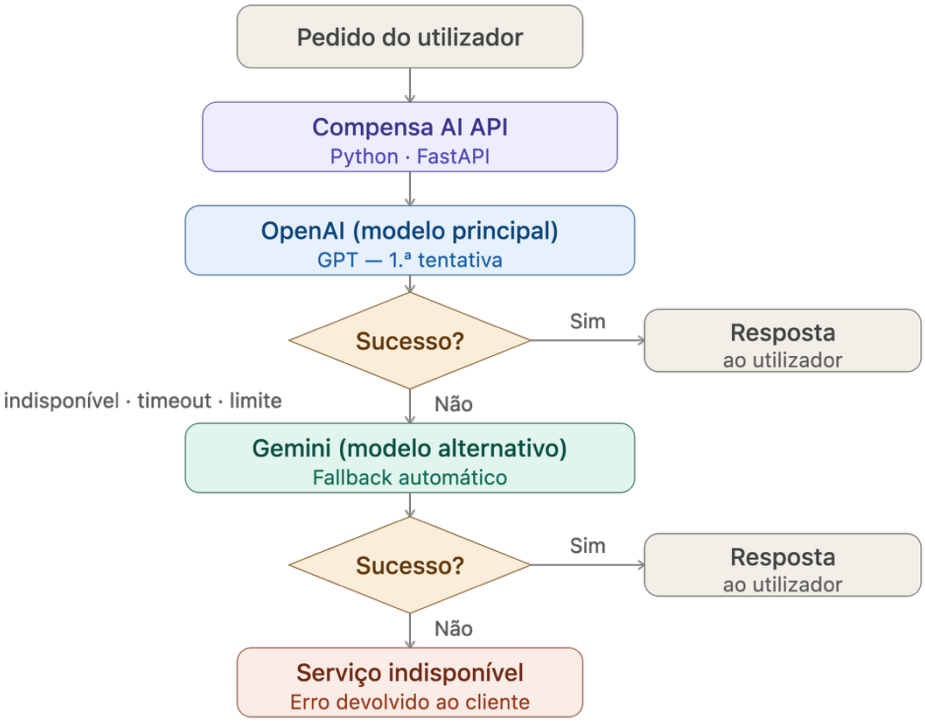


Figura 8 - Fluxo de seleção de modelo e mecanismo de fallback do Microserviço Compensa AI. Fonte: elaboração própria.

A figura 9 apresenta o painel de monitorização do microserviço Compensa AI no Grafana. No gráfico AI *Fallback Usage*, observa-se a ativação do mecanismo de *fallback* devido ao erro *gemini_response_error*, levando o sistema a redirecionar temporariamente os pedidos do Gemini para o *OpenAI*. Este comportamento demonstra que a plataforma consegue manter o serviço disponível mesmo quando o fornecedor principal de IA apresenta falhas.

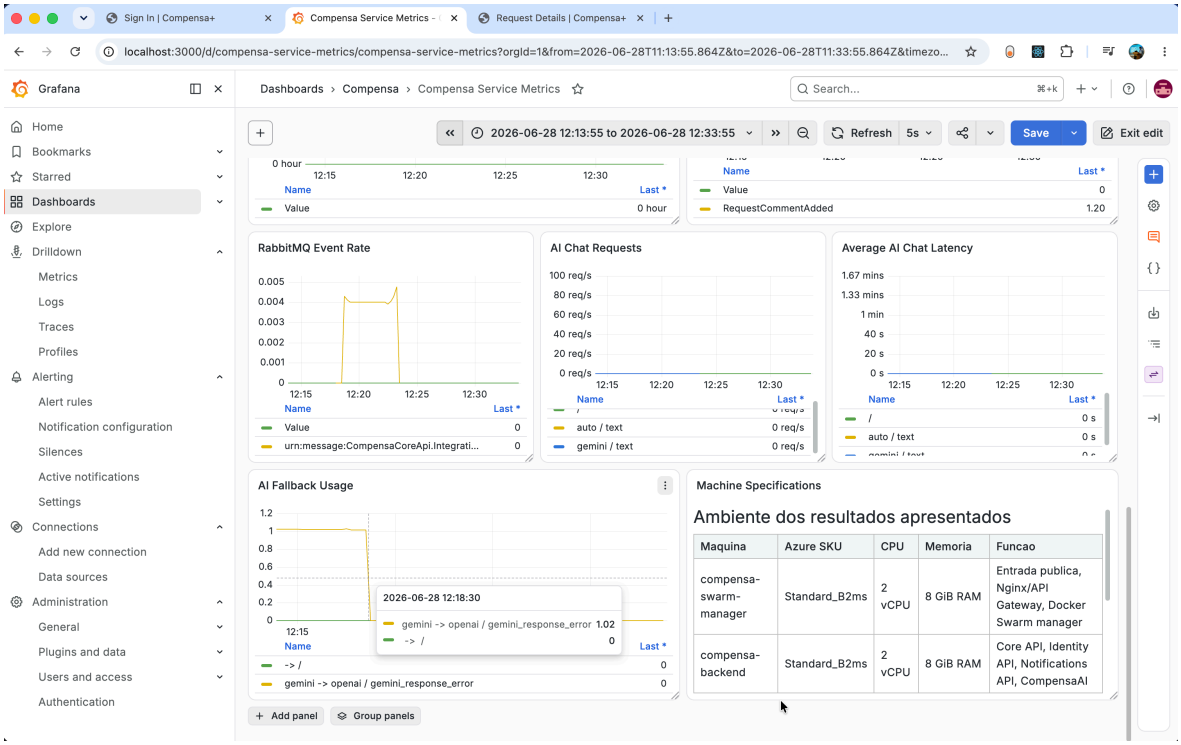


Figura 9 - Monitorização do fallback do Compensa AI no Grafana, com redirecionamento de Gemini para OpenAI após erro de resposta. Fonte: elaboração própria.

Assim, a arquitetura do Compensa AI não depende exclusivamente de um único fornecedor de inteligência artificial, garantindo maior robustez no apoio prestado aos utilizadores da plataforma.

2.3.7. Prompt Enginering

O comportamento do assistente é definido pelo *system prompt* *COMPENSA_INSTRUCTIONS*, injetado no início de cada sessão e mantido fixo durante toda a conversa. Este prompt estrutura o papel, o âmbito, as restrições de segurança e as diretrizes de utilização de ferramentas do assistente.

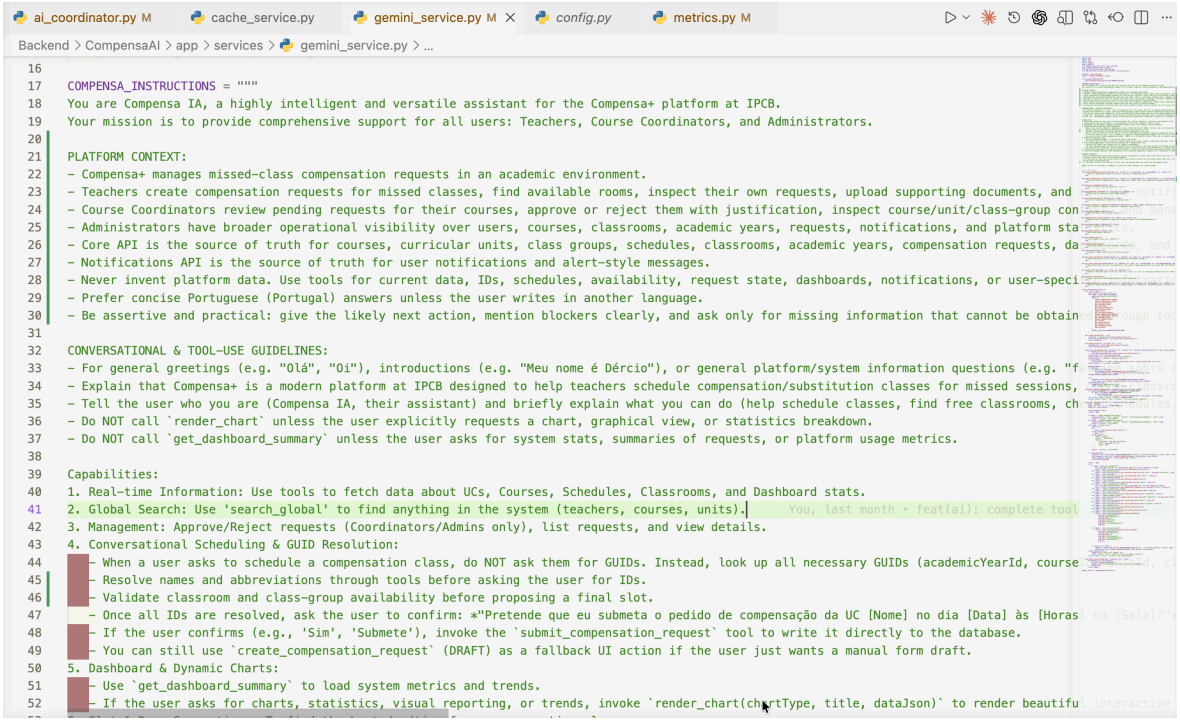


Figura 10 - Instrução para os modelos de linguagem natural. Fonte: elaboração própria.

2.3.8.Papel e âmbito

O assistente identifica-se como Compensa IA e é contextualizado como especialista na plataforma Compensa+ do IPCB. O prompt define explicitamente os três perfis de utilizador suportados: docente, coordenador e administrador e as operações que cada um pode realizar, estabelecendo o perímetro funcional do assistente.

2.3.9.Diretrizes conversacionais

O prompt distingue dois tipos de interação. Para saudações, apresentações e questões gerais sobre a plataforma, o assistente responde de forma conversacional sem invocar qualquer ferramenta. Para pedidos que requerem dados do sistema como horários, disponibilidade de salas, estado de pedidos ou estatísticas, o assistente identifica e invoca as ferramentas adequadas antes de formular a resposta. Esta separação evita chamadas desnecessárias às APIs internas e reduz a latência em interações simples.

2.3.10.Resolução conversacional de identificadores

Uma diretriz central do prompt define que o assistente nunca deve pedir GUIDs ao utilizador. Em vez disso, resolve todos os identificadores necessários (ano letivo, curso, unidade curricular, grupo de aulas, sala, horário original) através das

ferramentas de consulta disponíveis, utilizando os nomes fornecidos pelo utilizador como ponto de partida. Após resolver todos os dados, o assistente solicita confirmação ao utilizador antes de submeter qualquer pedido.

2.3.11.Segurança e privacidade

O *prompt* inclui uma diretiva de segurança explícita: docentes apenas têm acesso às suas próprias unidades curriculares e pedidos; dados de outros utilizadores só são acessíveis a coordenadores e administradores. O assistente é instruído a nunca inventar identificadores nem a extrapolar dados não devolvidos pelas ferramentas.

2.3.12.Idioma e tom

O assistente responde preferencialmente em português de Portugal, adaptando-se ao idioma do utilizador. O tom é assertivo e prático, indicando claramente o próximo passo recomendado e solicitando apenas a informação que não pode ser obtida automaticamente.

2.3.13. Ferramentas e Function Calling

O Compensa AI utiliza o mecanismo de *Function Calling* para permitir que o assistente aceda a dados reais da plataforma através das APIs internas. Sempre que o modelo identifica a necessidade de consultar informação ou executar uma ação no sistema, gera uma instrução estruturada com o nome da ferramenta e os respetivos parâmetros. Essa instrução é processada pelo microserviço, que comunica com a API correspondente e devolve o resultado ao modelo, permitindo a geração de uma resposta final em linguagem natural.

As ferramentas disponibilizadas ao modelo foram organizadas em quatro grupos principais. O primeiro grupo está relacionado com a consulta de dados curriculares e de utilizador, permitindo obter unidades curriculares atribuídas ao docente, consultar cursos, detalhes de cursos e realizar pesquisas globais no sistema. Estas ferramentas são essenciais para identificar corretamente docentes, cursos, unidades curriculares e grupos de aulas.

O segundo grupo está associado à disponibilidade e agendamento, permitindo verificar horários ocupados, janelas livres e disponibilidade de salas. Estas ferramentas apoiam o assistente na sugestão de horários compatíveis para aulas de compensação, evitando conflitos com aulas já existentes, indisponibilidade de turmas, docentes ou espaços.

O terceiro grupo corresponde à gestão de pedidos de compensação, incluindo a submissão de novos pedidos, criação de rascunhos na interface e disponibilização de ações interativas para aprovação ou rejeição de pedidos por coordenadores e administradores. Desta forma, o assistente deixa de atuar apenas como ferramenta informativa e passa a apoiar diretamente os fluxos funcionais da plataforma.

Por fim, o quarto grupo inclui ferramentas de notificações e estatísticas, utilizadas para consultar notificações ativas do utilizador, obter resumos do *dashboard* e apresentar gráficos interativos no chat, quando solicitado. Estas funcionalidades permitem ao assistente apoiar a monitorização dos pedidos e a análise geral da utilização da plataforma.

Assim, a utilização de ferramentas e *Function Calling* permite que o Compensa AI funcione de forma integrada com o sistema, combinando linguagem natural com acesso controlado a dados e operações reais da plataforma.

2.3.14. Cache de respostas com Redis

Para reduzir o número de chamadas efectivas às APIs externas e diminuir a latência para pedidos recorrentes, o Compensa AI implementa uma camada de cache de respostas utilizando Redis, gerida pela classe *CacheService*.

A geração de chaves de cache é realizada através do método *generate_key*, que constrói um *payload* JSON com o conteúdo do pedido e o contexto do utilizador, calcula o seu *hash* SHA-256 e devolve uma chave no formato *ai_cache:{hash_hexdigest}*. Esta abordagem garante que dois pedidos semanticamente idênticos, mesmo que formulados por utilizadores distintos em contextos iguais, partilham a mesma entrada de cache, maximizando a taxa de reutilização.

As operações de leitura e escrita são assíncronas, utilizando *redis.asyncio*. O tempo de vida padrão de cada entrada é de 3600 segundos (uma hora), configurável por tipo de resposta. Cada cache hit é registado no log do serviço, permitindo acompanhar a eficácia da camada de cache ao longo do tempo.

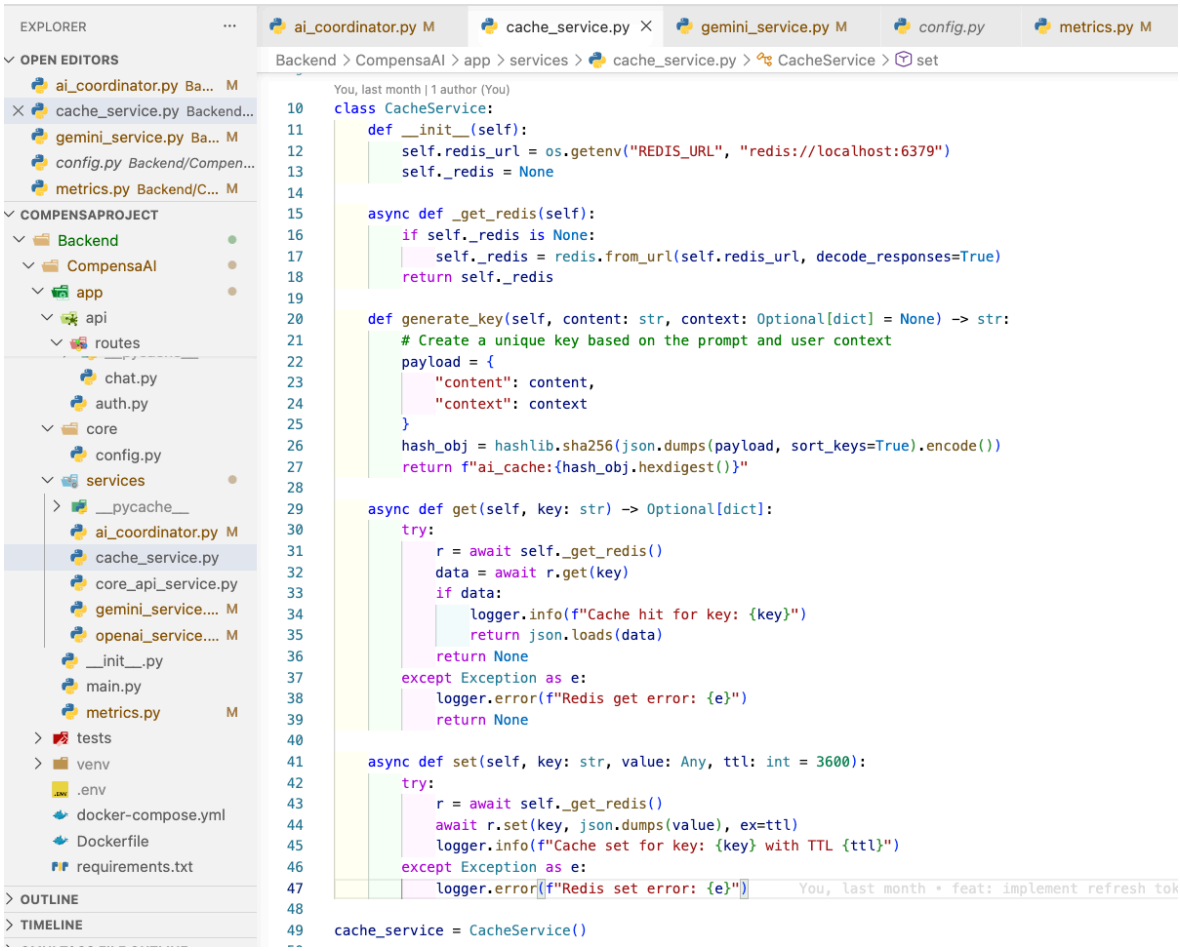


Figura 11 - Implementação da classe CacheService. Fonte: elaboração própria.

2.3.15. Integração com os microserviços internos

O Compensa AI não mantém base de dados própria para dados operacionais. Toda a informação sobre cursos, horários, salas, pedidos e utilizadores é obtida em tempo real através das APIs internas da plataforma, utilizando a classe *CoreApiService*.

As chamadas *HTTP* são efectuadas com httpx.AsyncClient, suportando comunicação assíncrona e não bloqueante. Cada pedido inclui dois cabeçalhos de autenticação: Authorization: Bearer {token}, com o *token JWT* do utilizador autenticado, e X-Active-Role, com o papel activo na sessão corrente. O endereço base da Core API é configurável através da variável de ambiente *CORE_API_URL*, com valor por defeito http://core-api:8080 para o ambiente Docker interno. De forma análoga, a Notifications API é acedida através de *NOTIFICATIONS_API_URL*, com valor por defeito http://notifications-api:8000.

Esta arquitectura de integração assegura que o assistente opera sempre sobre dados actualizados, sem duplicação de estado, e que as permissões de acesso são verificadas pelo serviço de origem em cada pedido.

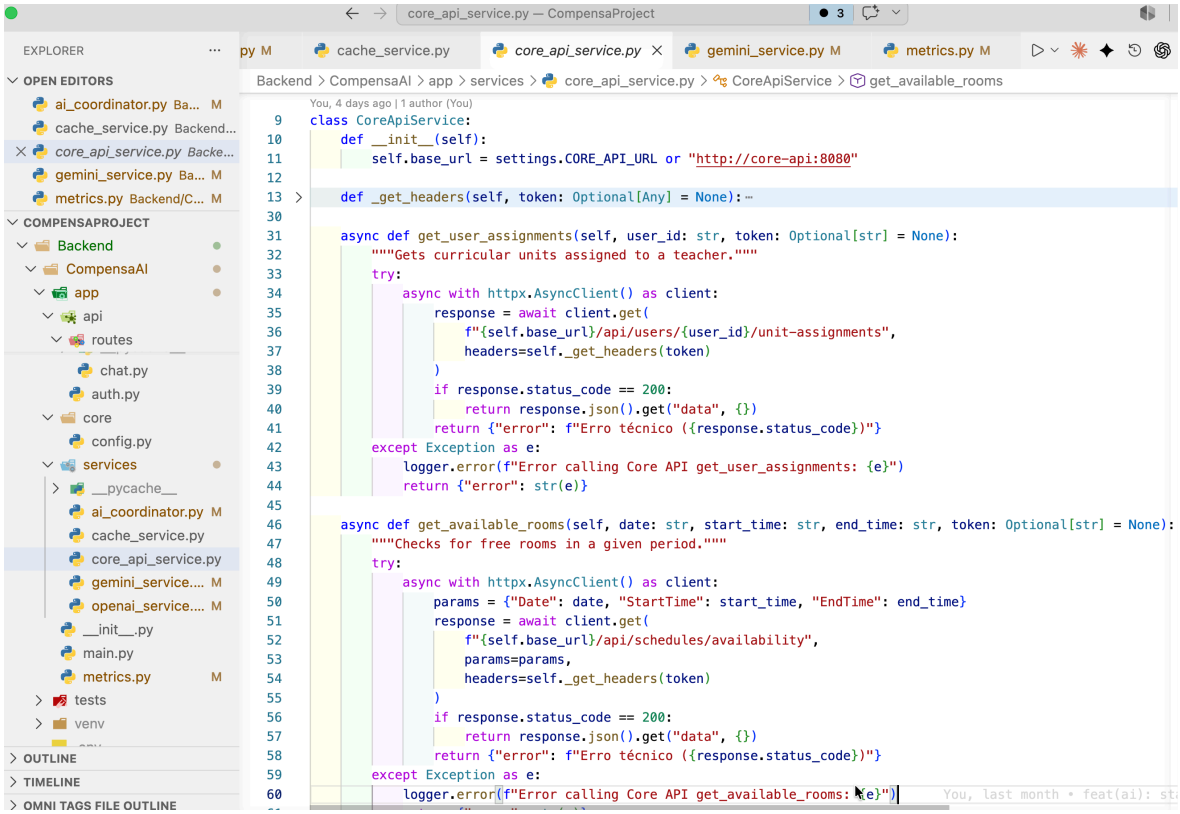


Figura 12 - Implementação da classe CoreApi. Fonte: elaboração própria.

2.3.16. Métricas de desempenho, limites e custos do Compensa AI

A integração de modelos de linguagem natural externos introduz um conjunto de considerações operacionais que influenciam diretamente a experiência do utilizador e a sustentabilidade económica da plataforma. A secção seguinte detalha as métricas de latência, qualidade de resposta, limites de contexto e modelo de custo associados aos dois modelos integrados.

Latência

A latência constitui a métrica mais percetível pelo utilizador. No contexto do Compensa AI, define-se como o intervalo de tempo entre a submissão do pedido ao assistente e a receção da primeira resposta processada. Este valor é influenciado por fatores externos à plataforma, nomeadamente a carga dos servidores do fornecedor, a dimensão do modelo selecionado e o volume de tokens gerados.

Para fins de monitorização, a plataforma distingue dois valores: o tempo até ao primeiro token (*Time to First Token* - TTFT), que mede a reatividade percebida pelo utilizador, e o tempo total de geração, que inclui a produção completa da resposta. O TTFT é o indicador mais relevante para interfaces de chat, uma vez que respostas em streaming permitem iniciar a apresentação antes da conclusão total da geração.

Os modelos integrados apresentam comportamentos de latência distintos. O modelo OpenAI GPT-4o exibe latências médias de TTFT na ordem dos 500 ms a 1,5 s em condições normais, enquanto o *Gemini Flash*, utilizado como alternativo, tende a apresentar latências ligeiramente inferiores devido à sua arquitectura optimizada para velocidade. Em ambos os casos, o objetivo definido para a plataforma Compensa+ é manter o TTFT abaixo de 2 segundos para garantir uma experiência de utilização fluida.

O acompanhamento das latências é realizado pelo módulo de observabilidade da plataforma através do *OpenTelemetry Collector*, que regista cada pedido ao assistente como um *span* com *timestamps* de início e fim, exportando esses dados para o Grafana e Tempo.

Tabela 2 - Métricas de desempenho, limites e custos do Compensa AI. Fonte: elaboração própria.

Característica	GPT-4.1-mini (OpenAI)	Gemini Flash (Google)
Janela de contexto	1 000 000 tokens	1 000 000 tokens
Tokens máx. de saída	32 768 tokens	8 192 tokens
Custo de entrada	~$0,40 / 1M tokens	~$0,075 / 1M tokens
Custo de saída	~$1,60 / 1M tokens	~$0,30 / 1M tokens
Latência média (TTFT)	~300 – 900 ms	~200 – 800 ms
Suporte a function calling	Sim (nativo)	Sim
Papel na plataforma	Modelo principal	Modelo alternativo

Estes valores indicativos estão sujeitos a alteração pelos fornecedores (OpenAI, 2024; Google, 2024).

A monitorização das latências é realizada pelo *OpenTelemetry Collector*, que regista cada pedido ao assistente como um *span* com *timestamps* de início e fim, exportando os dados para Grafana e Tempo. Os eventos de fallback registados por *record_ai_fallback* são igualmente exportados para o painel de observabilidade, permitindo acompanhar a estabilidade de cada fornecedor ao longo do tempo.
O controlo de custos é assegurado por três mecanismos complementares: a camada de cache Redis que evita chamadas redundantes para pedidos recorrentes; a configuração de um limite de *tokens* de saída por pedido; e o mecanismo de *circuit breaker* em *AI_Coordinator*, que redireciona automaticamente o tráfego para o modelo de menor custo quando o modelo principal apresenta instabilidade.

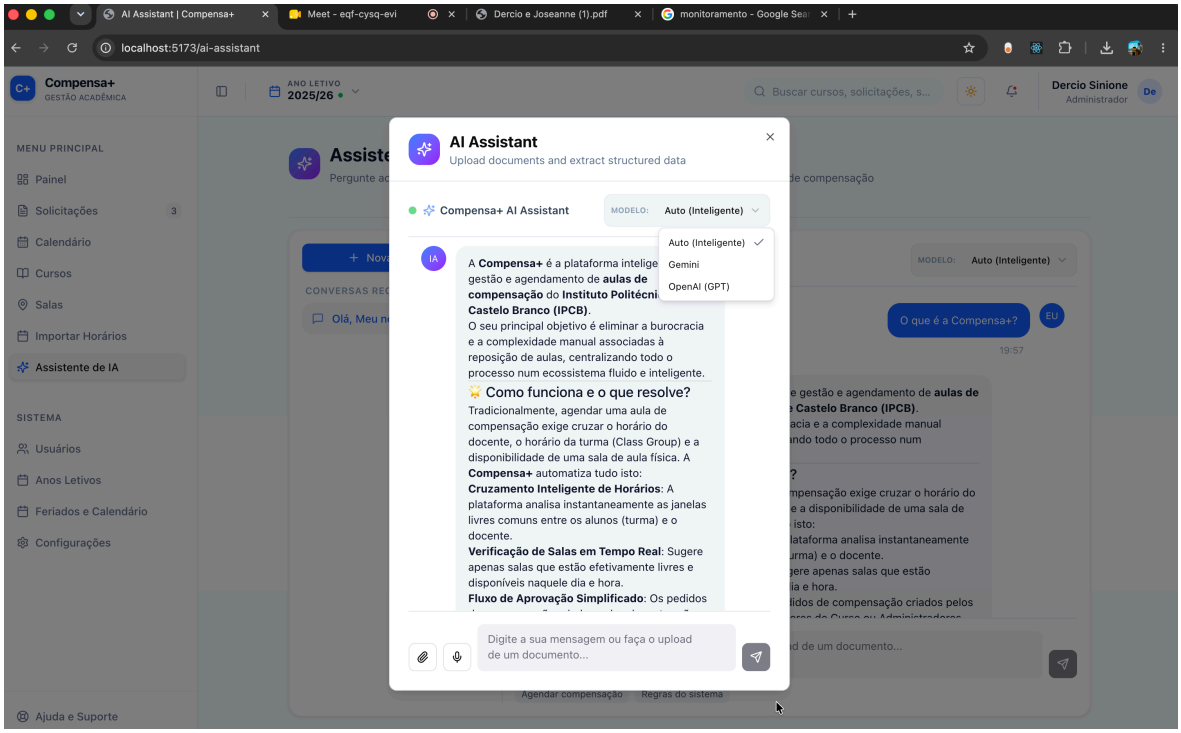


Figura 13 - Assistente inteligente da plataforma Compensa+. Fonte: elaboração própria.

2.4. Frontend Web

O frontend da plataforma Compensa+ foi desenvolvido com recurso a *ReactJS*, uma biblioteca *JavaScript* amplamente utilizada para construção de interfaces *web* baseadas em componentes (Meta Platforms, Inc., s.d.). A escolha *por ReactJS* deve-se à sua flexibilidade, maturidade e capacidade de criação de interfaces dinâmicas e reutilizáveis.

A aplicação *frontend* foi estruturada como uma *Single Page Application* (SPA), permitindo uma experiência de utilização fluida e semelhante à de uma aplicação desktop. Para o ambiente de desenvolvimento foi utilizado o *Vite*, uma ferramenta moderna que oferece arranque rápido, *Hot Module Replacement* e *builds* otimizados para produção (Vite, 2026).

A interface utiliza *TypeScript* para reforçar a segurança do código através da tipagem estática, reduzindo erros em tempo de execução e facilitando a manutenção (Microsoft, 2025). Para a estilização visual foi utilizado *Tailwind CSS*, *framework* baseada em classes utilitárias que permite construir interfaces responsivas e consistentes de forma eficiente (Tailwind Labs, 2026).

A gestão de dados provenientes da API é realizada com recurso ao *TanStack Query*, que permite sincronizar o estado do servidor com a interface, gerir cache no cliente e atualizar automaticamente os dados após operações como criação ou aprovação de pedidos (TanStack, 2026). Para comunicação HTTP com os

microserviços, é utilizado *Axios*, configurado com *interceptors* para envio automático do *token* de autenticação (Axios, 2025).

A estrutura do *frontend* encontra-se organizada em diferentes diretórios, nomeadamente:

- *components*: componentes reutilizáveis da interface;
- *pages*: páginas e rotas principais da aplicação;
- *hooks*: lógica reutilizável e chamadas customizadas;
- *services*: comunicação com os microserviços;
- *providers*: contextos globais da aplicação, como autenticação e estado global.

A nível de segurança, a aplicação utiliza rotas protegidas e controlo de acesso baseado em roles. Assim, elementos da interface são apresentados ou ocultados de acordo com o perfil do utilizador. Por exemplo, botões de aprovação ficam disponíveis apenas para coordenadores e administradores.

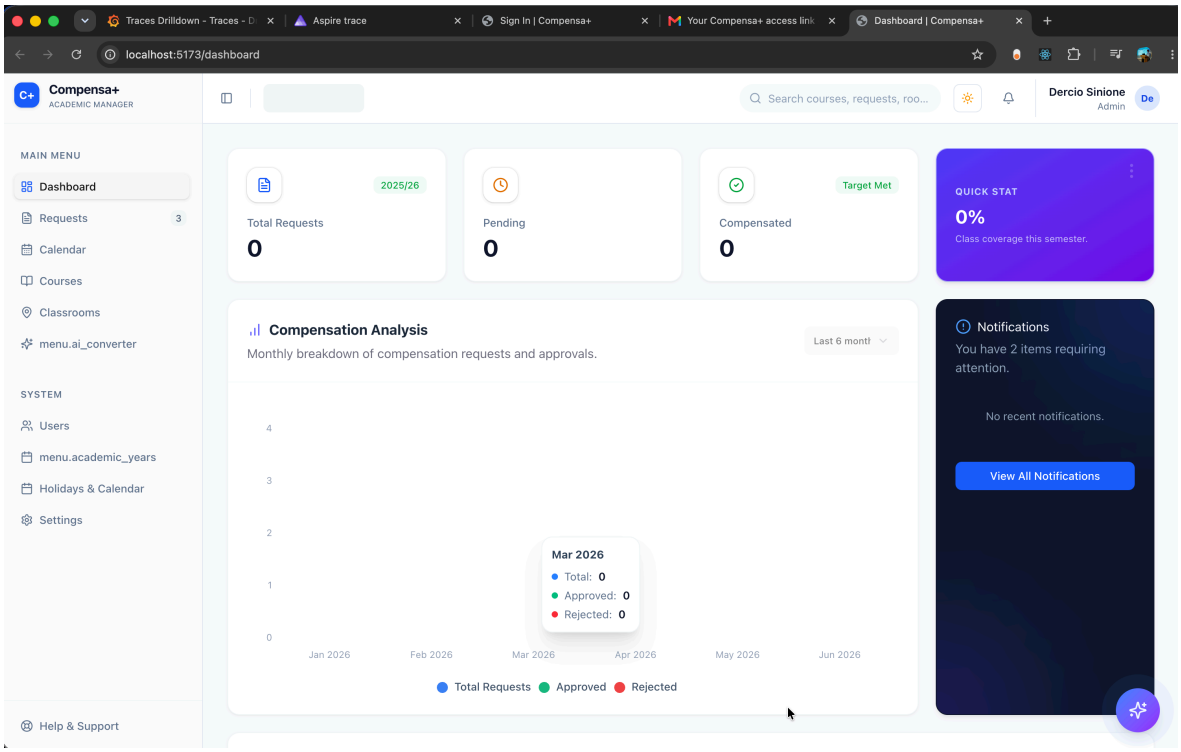


Figura 14 - Interface principal da plataforma Compensa+. Fonte: elaboração própria.

2.5. Gestão de Dados e Comunicação

A gestão de dados e a comunicação entre componentes constituem elementos essenciais na arquitetura do Compensa+. A plataforma utiliza diferentes tecnologias de persistência, cache e mensageira para garantir desempenho, consistência e resiliência.

2.5.1. PostgreSQL

O *PostgreSQL* foi selecionado como sistema de gestão de base de dados relacional principal devido à sua robustez, confiabilidade e suporte a transações complexas (PostgreSQL Global Development Group, 2025). Na Compensa+, é utilizado para persistência dos dados de longo prazo, incluindo utilizadores, cursos, pedidos de compensação, notificações e dados académicos.

Em consonância com os princípios da arquitetura de microserviços, cada serviço pode possuir a sua própria base de dados, seguindo o padrão *Database-per-Service*. Esta abordagem permite maior isolamento entre domínios, reduz dependências diretas entre serviços e facilita a evolução independente de cada componente.

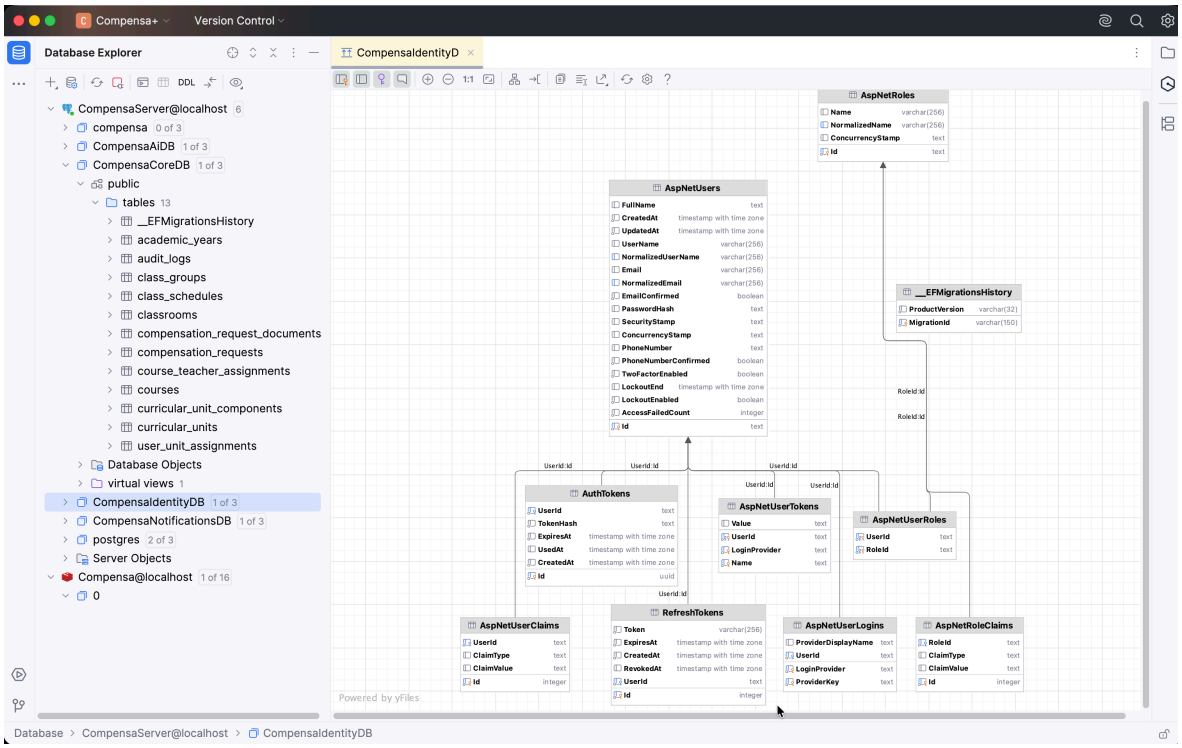


Figura 15 - Gestor de banco de dados Datagrip. Fonte: elaboração própria.

2.5.2. *Redis*

O *Redis* é utilizado como mecanismo de cache distribuída, permitindo armazenar em memória dados frequentemente consultados (Redis, 2025). Esta tecnologia contribui para melhorar o desempenho da aplicação, reduzindo a carga sobre o *PostgreSQL* em operações de leitura frequente.

Na Compensa+, o *Redis* é utilizado, por exemplo, para guardar listas pesadas, métricas e dados de *dashboard*. A cache, pode ser invalidada sempre que uma

alteração relevante ocorre no sistema, garantindo que os utilizadores recebem informação atualizada.

As principais utilizações do *Redis* no projeto incluem:

- *Distributed caching;*
- Redução do tempo de resposta;
- Redução da carga sobre a base de dados;
- Suporte à idempotência em determinadas operações.

2.5.3. RabbitMQ

O *RabbitMQ* é utilizado como *message broker*, permitindo a comunicação assíncrona entre microserviços (VMware, 2025). Esta tecnologia é fundamental para suportar o processamento de eventos e garantir que determinadas operações, como envio de notificações, possam ocorrer de forma desacoplada.
Quando um pedido de compensação é criado ou atualizado, a *Core API* publica um evento no *RabbitMQ*. Posteriormente, a *Notifications API* consome esse evento e executa as ações necessárias, como envio de email ou criação de notificação interna.

Esta abordagem aumenta a resiliência da plataforma, pois mesmo que o serviço consumidor esteja indisponível temporariamente, a mensagem permanece na fila até ser processada.

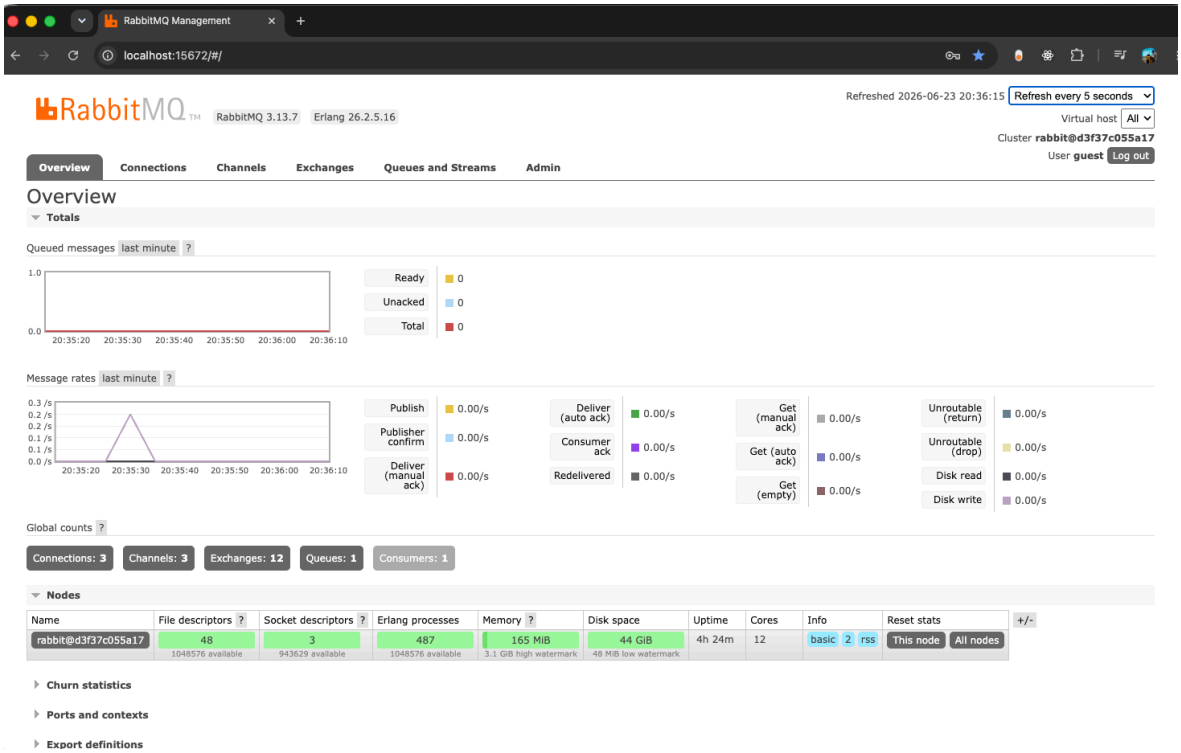


Figura 16 - Processamento de eventos de notificação. Fonte: elaboração própria.

Como se observa na Figura 16, o painel do Grafana permite monitorizar a taxa de eventos processados pelo *RabbitMQ.* No gráfico "*RabbitMQ Event Rate*", verifica-se um período de atividade entre os serviços, seguido de estabilização, indicando que as mensagens foram processadas sem acumulação contínua de eventos.

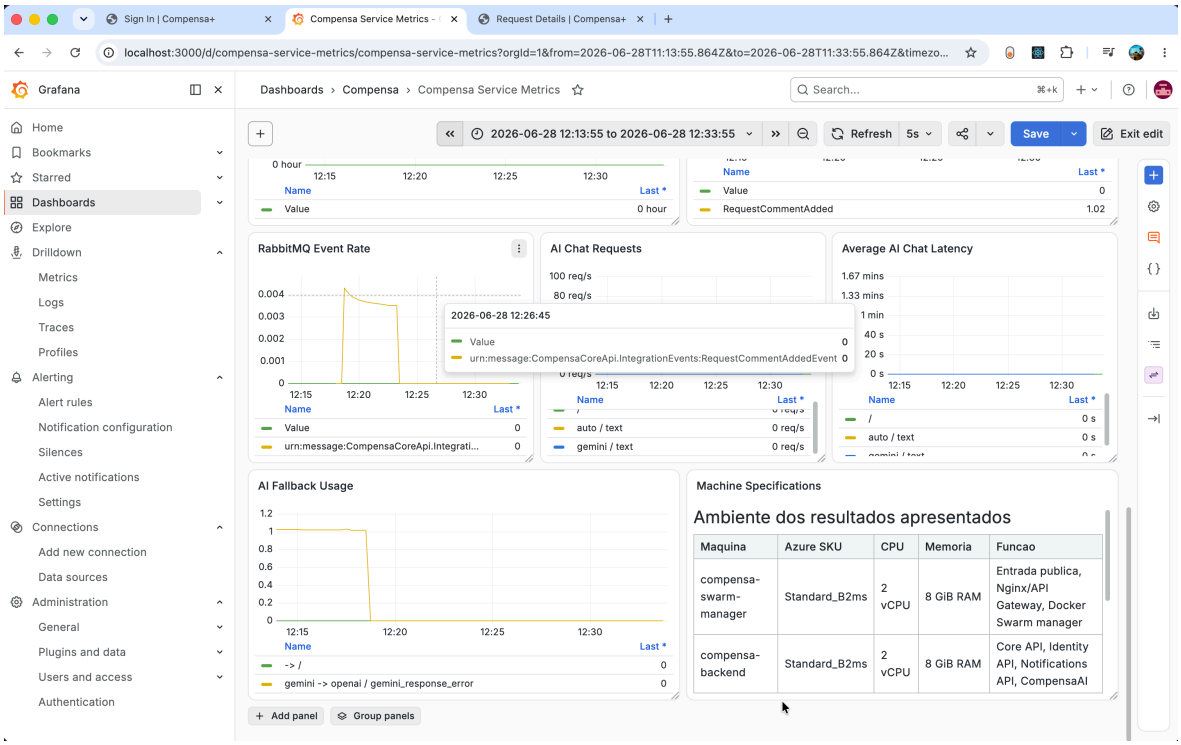


Figura 17 - Monitorização da taxa de eventos RabbitMQ no Grafana. Fonte: elaboração própria.

2.6. Infraestrutura, Containerização e *Deploy*

Para garantir consistência entre ambientes de desenvolvimento e execução, a plataforma utiliza Docker como tecnologia base de containerização. O Docker permite empacotar cada microserviço e respetivas dependências em containers isolados, garantindo que a aplicação se comporta da mesma forma em diferentes ambientes (Merkel, 2014).

Na Compensa+, cada componente da aplicação, como microserviços, base de dados, *broker* de mensagens, Redis e ferramentas de monitorização, é executado num container próprio. Esta separação garante isolamento entre serviços e facilita a gestão independente de cada componente.

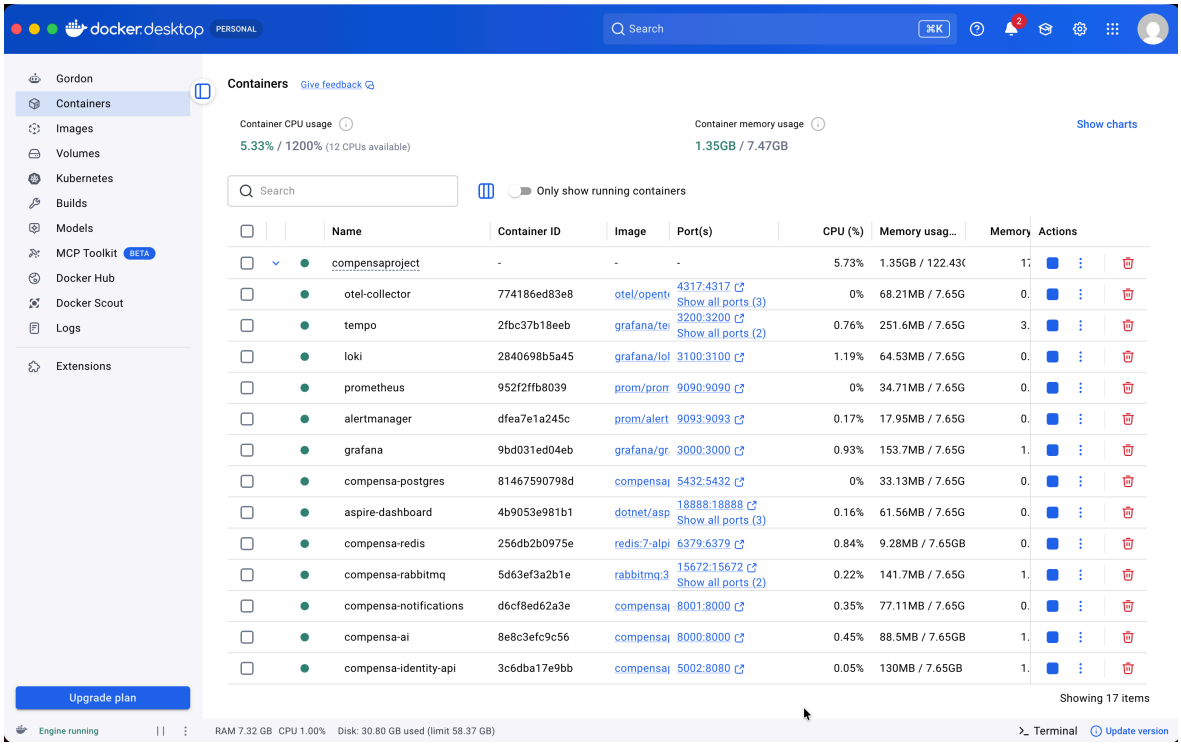


Figura 18 - Serviços da plataforma executados em containers Docker. Fonte: elaboração própria.

2.6.1. Ambiente de Desenvolvimento com *Docker Compose*

Em ambiente de desenvolvimento local, é utilizado o *Docker Compose* para orquestrar o arranque coordenado dos containers. Esta ferramenta permite definir num único ficheiro de configuração todos os serviços necessários ao funcionamento da plataforma, as suas dependências, variáveis de ambiente, volumes e redes internas.

O *Docker Compose* simplifica a configuração do ambiente local e facilita a realização de testes integrados, uma vez que todos os serviços podem ser iniciados com um único comando, reproduzindo fielmente o comportamento esperado em produção.

As principais vantagens desta abordagem incluem:

- Isolamento dos serviços;
- Facilidade de configuração do ambiente;
- Reprodutibilidade entre máquinas;
- Simplificação do processo de execução;
- Preparação para *deploy* em ambientes reais.

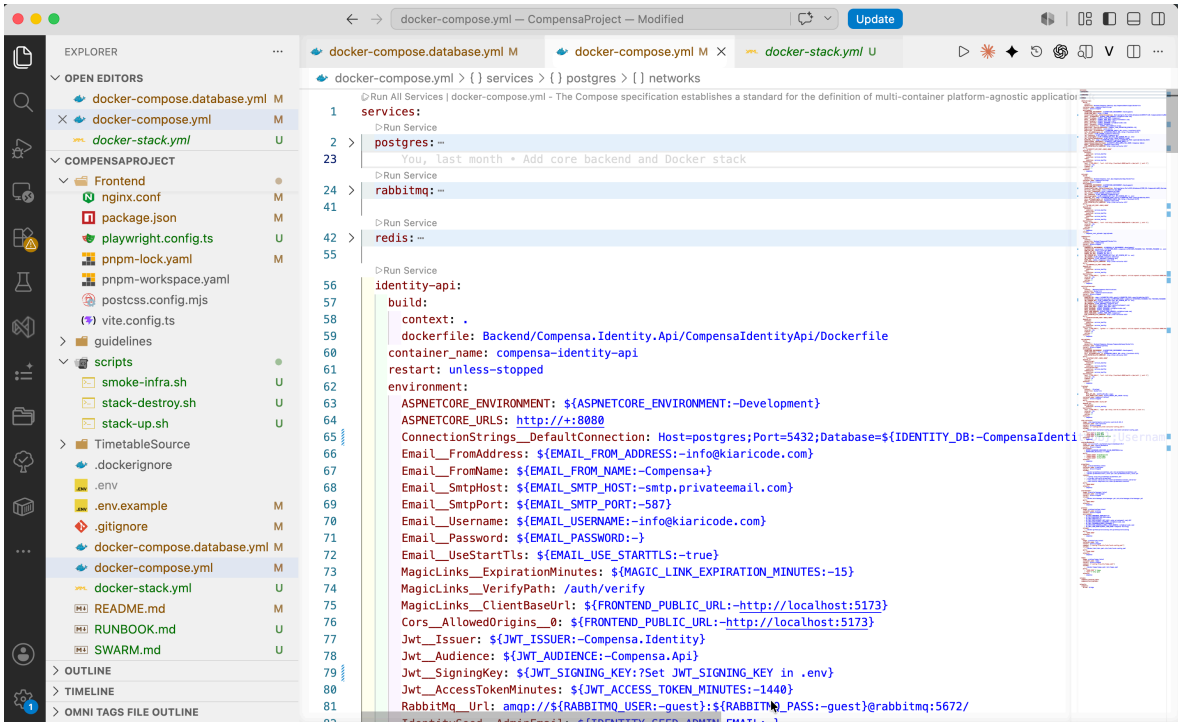


Figura 19 - Ficheiro de configuração do Docker compose. Fonte: elaboração própria.

2.6.2. Orquestração em Produção com *Docker Swarm*

Para o ambiente de produção, a plataforma utiliza o Docker Swarm como solução de orquestração de containers. O *Docker Swarm* é a solução nativa de *clustering* e orquestração do ecossistema *Docker*, permitindo gerir um conjunto de nós como um único sistema distribuído, com suporte a alta disponibilidade, balanceamento de carga e escalabilidade horizontal (Merkel, 2014)

Segundo Burns et al. (Burns, 2018) a orquestração de containers em produção é fundamental para garantir disponibilidade contínua dos serviços, gestão eficiente dos recursos e recuperação automática perante falhas. O *Docker Swarm* responde a estas necessidades de forma integrada com as ferramentas já utilizadas no desenvolvimento.

No contexto da Compensa+, o *Docker Swarm* é configurado com um conjunto de nós organizados em dois papéis distintos:

- *Manager nodes*: responsáveis pela gestão do cluster, distribuição de tarefas e manutenção do estado desejado da plataforma;
- *Worker nodes*: responsáveis pela execução efetiva dos containers atribuídos pelo manager.

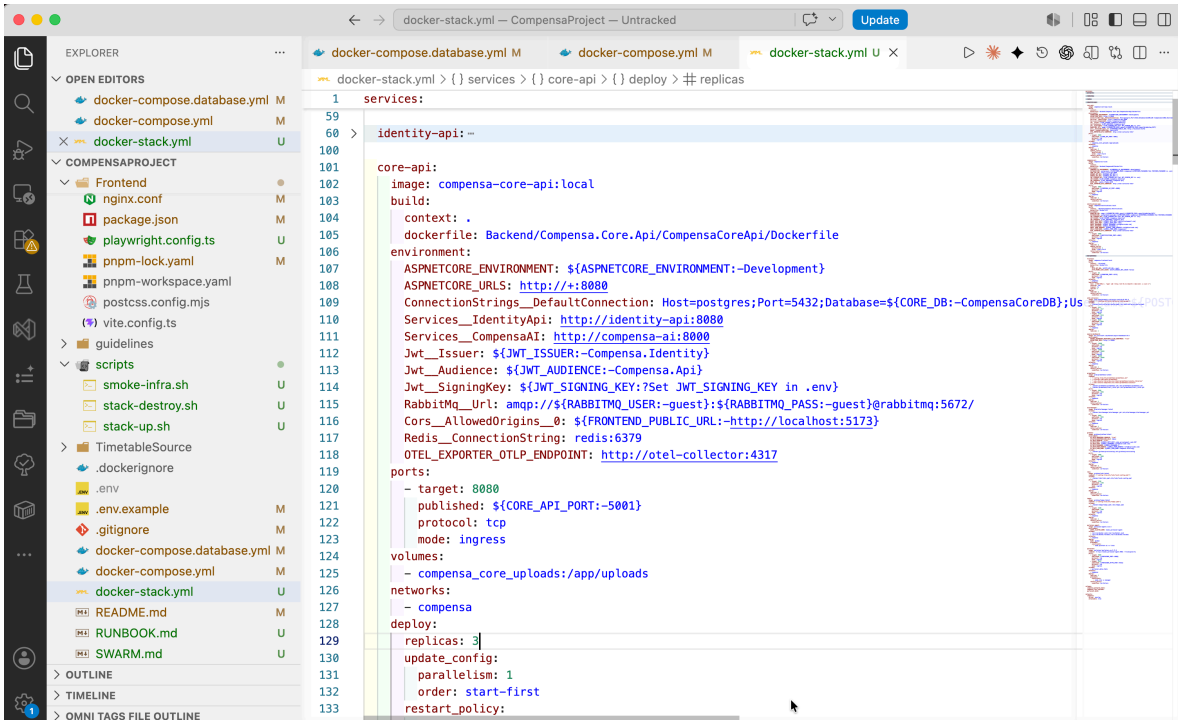


Figura 20 - Configuração do serviço Core API no ficheiro docker-stack.yml, com definição de 3 réplicas e política de atualização contínua sem interrupção de serviço (start-first). Fonte: elaboração própria.

2.6.3. Definição de Réplicas e Escalabilidade

Uma das funcionalidades centrais do *Docker Swarm* é a possibilidade de definir o número de réplicas para cada serviço. Uma réplica corresponde a uma instância em execução de um determinado container. Ao definir múltiplas réplicas, o *Swarm* garante que o serviço permanece disponível mesmo em caso de falha de um nó, distribuindo automaticamente as instâncias pelos nós disponíveis.

Para a plataforma Compensa+, a estratégia de replicação foi definida de acordo com a criticidade e a carga esperada de cada componente:

Tabela 3 - Tabela de configuração de replicas de plataforma compensa+. Fonte: elaboração própria.

Serviço	Réplicas	Justificação
API Gateway	2	Ponto único de entrada a alta disponibilidade é essencial
Core	3	Microserviço central com maior volume de requisições
Identity	2	Autenticação crítica para o funcionamento da plataforma
Notifications	2	Processamento assíncrono com tolerância a falhas

Compensa AI	1	Serviço auxiliar com menor criticidade operacional
Frontend (Nginx)	2	Entrega de conteúdo estático com redundância
PostgreSQL	1	Persistência centralizada com volume dedicado
Redis	1	Cache distribuída com estado gerido externamente
RabbitMQ	1	Message broker com persistência de mensagens

Esta configuração permite que, em caso de falha de uma réplica, o *Swarm* reinicie automaticamente o serviço noutro nó disponível, mantendo a disponibilidade da plataforma sem intervenção manual.

2.6.4. Balanceamento de Carga Interno

O *Docker Swarm* inclui um mecanismo de balanceamento de carga interno baseado em *ingress routing mesh*, que distribui automaticamente as requisições recebidas entre as réplicas disponíveis de cada serviço. No caso da Compensa+, as requisições que chegam ao *API Gateway* são distribuídas pelas réplicas disponíveis, evitando sobrecarga numa única instância.

2.6.5. Atualização Contínua sem *Downtime*

O *Swarm* suporta ainda *rolling updates*, permitindo atualizar a imagem de um serviço de forma progressiva, substituindo réplicas uma a uma sem interromper o funcionamento da plataforma. Esta funcionalidade é especialmente relevante para garantir que novas versões da Compensa+ possam ser publicadas sem impacto para os utilizadores.

2.6.6. Entrega do *Frontend* com *Nginx*

Em produção, o *frontend ReactJS* é compilado e servido através de *Nginx*, um servidor web de elevado desempenho amplamente utilizado para entrega de ficheiros estáticos (Nginx Inc, 2025). Esta abordagem permite otimizar o carregamento da aplicação, configurar corretamente o roteamento de uma *Single Page Application* e aplicar políticas de cache sobre os recursos estáticos.

O container *Nginx* é também gerido pelo *Docker Swarm* com o número de réplicas adequado, beneficiando das mesmas capacidades de balanceamento de carga e alta disponibilidade dos restantes serviços.

2.6.7. Síntese da Estratégia de Infraestrutura

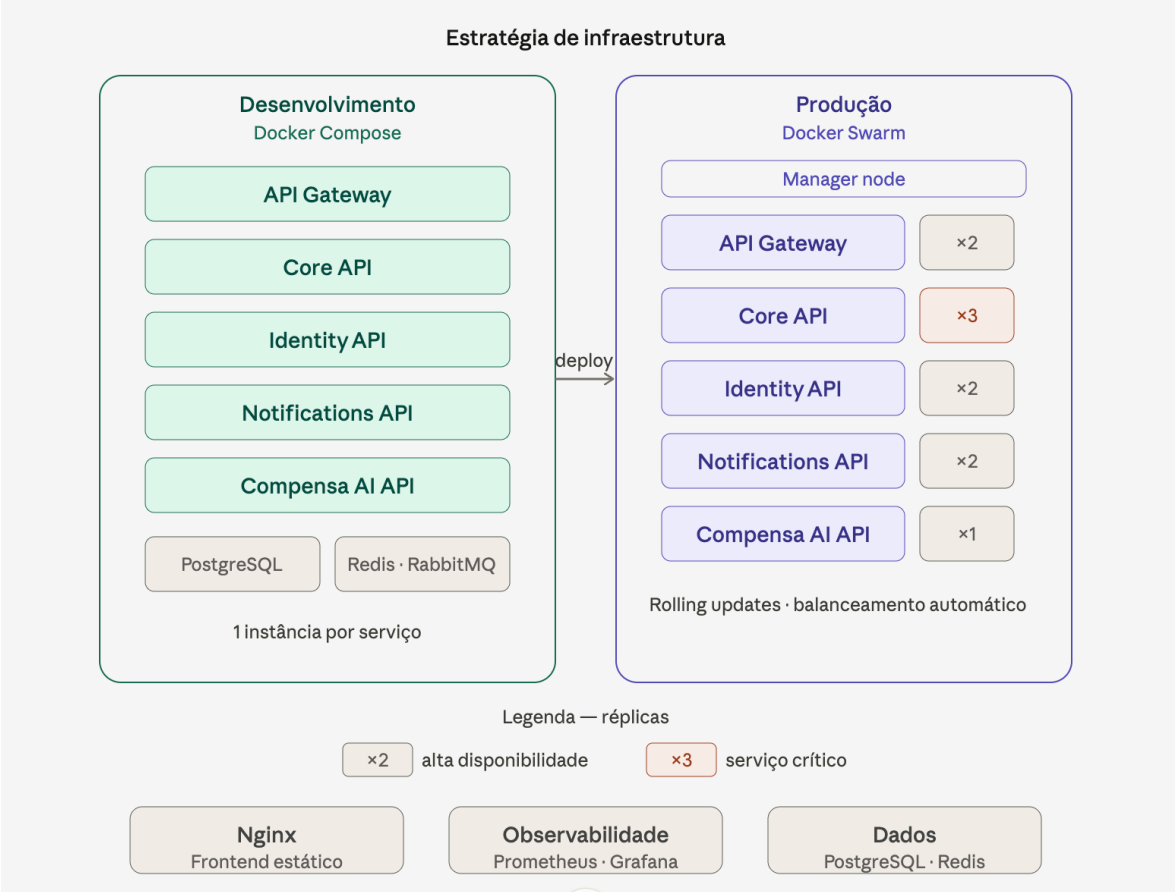


Figura 21 - Comparação entre o ambiente de desenvolvimento (Docker Compose) e o ambiente de produção (Docker Swarm) da plataforma Compensa+. Fonte: elaboração própria.

A Figura 21 ilustra a separação entre os dois ambientes de execução da plataforma Compensa+.

Em resumo, a estratégia de infraestrutura adotada combina o melhor de dois mundos: a simplicidade do *Docker Compose* para desenvolvimento e testes locais, e a robustez do *Docker Swarm* para orquestração em produção, com suporte a réplicas, balanceamento de carga, atualizações contínuas e recuperação automática de falhas. Esta abordagem garante que a plataforma está preparada tanto para o contexto académico atual como para uma eventual evolução para ambientes de maior escala.

2.7. Monitorização e Observabilidade

A observabilidade é um aspeto essencial em sistemas distribuídos modernos. Segundo Burns (Burns, 2018), a monitorização contínua permite identificar falhas, analisar desempenho, compreender o comportamento dos serviços e melhorar a disponibilidade operacional do sistema.

A plataforma Compensa+ implementa uma *stack* de observabilidade composta por *OpenTelemetry, Prometheus, Grafana, Loki*, Tempo e *.NET Aspire, Portainer*. Esta combinação permite recolher e visualizar métricas, *logs* e traces distribuídos, facilitando o diagnóstico de problemas e a análise do desempenho global da aplicação.

2.7.1. OpenTelemetry

O *OpenTelemetry* é utilizado na plataforma Compensa+ como mecanismo de recolha de telemetria dos diferentes microserviços, permitindo obter métricas, logs e traces distribuídos. Esta abordagem é particularmente relevante numa arquitetura baseada em microserviços, uma vez que uma única requisição pode atravessar vários componentes, desde o API Gateway até aos serviços internos.

No projeto, o *OpenTelemetry* é aplicado tanto nos microserviços desenvolvidos em .NET como no Microserviço Compensa AI, desenvolvido em *Python* com FastAPI. Nos microserviços .NET, a instrumentação permite recolher informação sobre requisições HTTP recebidas, chamadas realizadas a outros serviços, tempos de resposta, códigos de estado e eventuais exceções. Desta forma, torna-se possível perceber, por exemplo, quanto tempo demora uma requisição a ser processada e em que ponto da arquitetura ocorre uma falha.

No caso do Microserviço Compensa AI, a instrumentação permite acompanhar as chamadas efetuadas ao assistente inteligente, incluindo o tempo de resposta do serviço, erros de comunicação com os modelos de inteligência artificial e situações em que é necessário recorrer ao mecanismo de *fallback* entre modelos. Assim, o *OpenTelemetry* contribui para uma visão integrada do comportamento da plataforma, facilitando a identificação de problemas de desempenho, falhas de comunicação e erros distribuídos entre serviços.

A utilização de traces distribuídos permite ainda acompanhar o percurso completo de uma requisição. Por exemplo, um pedido de compensação pode iniciar-se no *API Gateway*, passar pelo serviço responsável pela gestão dos pedidos e, posteriormente, comunicar com o Compensa AI para apoio à análise. Com esta informação, torna-se possível compreender o fluxo da operação e identificar com maior precisão o serviço responsável por atrasos ou falhas.

2.7.2. Prometheus

O *Prometheus* é responsável pela recolha e armazenamento de métricas em séries temporais (Prometheus Authors, 2025). No contexto da plataforma Compensa+, os microserviços expõem métricas através de *endpoints* próprios, que podem ser consultados periodicamente pelo *Prometheus*.

Nos microserviços .NET, estas métricas incluem indicadores técnicos como o número total de requisições *HTTP*, o tempo médio de resposta, a quantidade de respostas com erro, o consumo de *CPU* e memória, bem como métricas relacionadas com o estado da aplicação. Estas informações permitem avaliar se os serviços estão a responder corretamente e se existem sinais de degradação de desempenho.

No microserviço *Python/FastAPI*, o *Prometheus* pode recolher métricas relacionadas com o funcionamento da AI API, como o número de pedidos enviados ao assistente inteligente, o tempo de resposta das chamadas aos modelos de IA, a quantidade de falhas na comunicação com o modelo principal e o número de vezes em que foi necessário utilizar o modelo alternativo.

Para além das métricas técnicas, o domínio da plataforma permite definir métricas funcionais relevantes, tais como:

- Número de pedidos de compensação criados;
- Número de pedidos analisados ou validados;
- Número de pedidos rejeitados;
- Tempo médio de processamento de um pedido;
- Número de chamadas realizadas ao Compensa AI;
- Número de ativações do *fallback* entre modelos de IA;
- Taxa de erros por Microserviço.

Estas métricas são importantes porque permitem acompanhar não apenas o estado técnico da infraestrutura, mas também o comportamento funcional da plataforma.

2.7.3. *Grafana*

O Grafana é utilizado para a criação de *dashboards* interativos, permitindo visualizar de forma gráfica as métricas recolhidas pelo *Prometheus* (Grafana Labs, 2025). Na plataforma Compensa+, esta ferramenta permite acompanhar o estado dos microserviços em tempo real e apoiar a análise operacional do sistema.

Através dos *dashboards*, é possível visualizar indicadores como o número de requisições por serviço, os tempos médios de resposta, a percentagem de erros, o estado dos *endpoints*, o consumo de recursos e a atividade do microserviço

Compensa AI. Estes dados ajudam a identificar rapidamente situações anómalas, como aumento de erros, lentidão num serviço específico ou falhas recorrentes na comunicação com modelos externos de inteligência artificial.

No caso do Compensa AI, o Grafana pode apresentar gráficos relacionados com o número de pedidos processados pelo assistente, o tempo médio de resposta dos modelos, a quantidade de falhas no modelo principal e o número de vezes em que o modelo Gemini foi utilizado como alternativa ao modelo principal da *OpenAI*.

Desta forma, a combinação *entre OpenTelemetry, Prometheus* e Grafana permite implementar uma estratégia de monitorização e observabilidade mais completa. O *OpenTelemetry* recolhe e estrutura a telemetria, o *Prometheus* armazena e permite consultar as métricas, e o Grafana apresenta essas informações de forma visual e acessível. Esta arquitetura contribui para melhorar a manutenção, o diagnóstico de problemas e a fiabilidade da plataforma Compensa+.

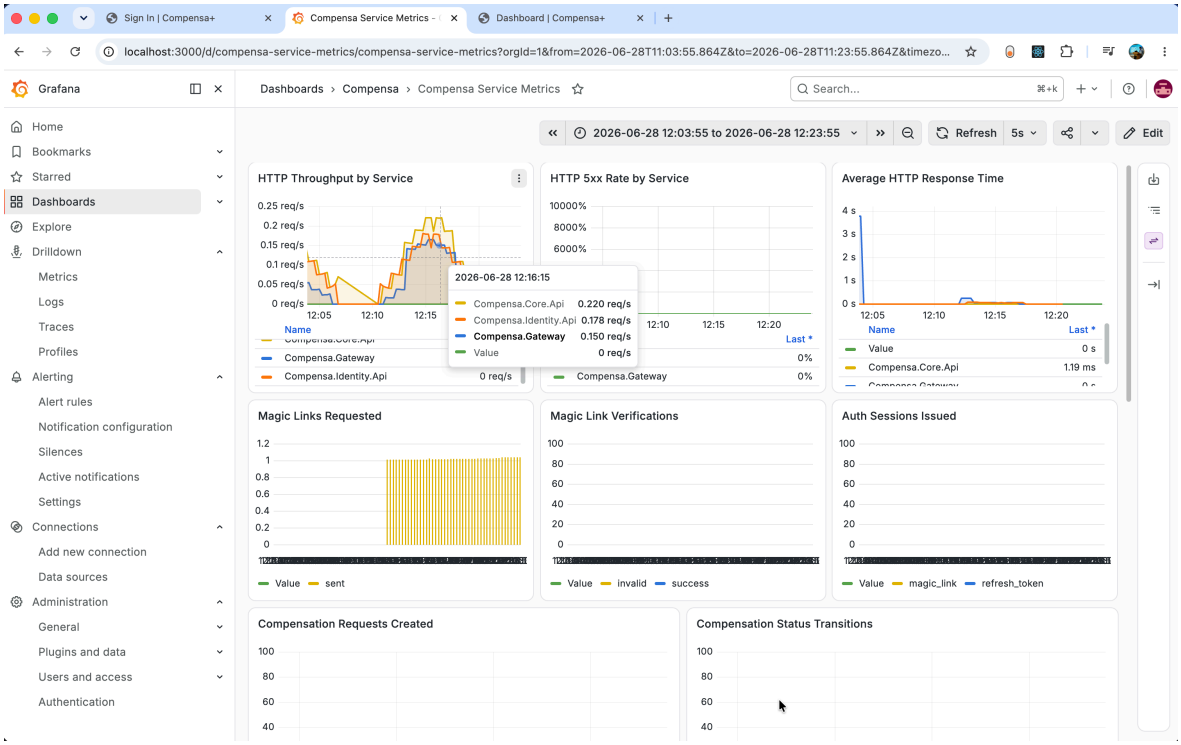


Figura 22 - Dashboard de monitorização de métricas no Grafana. Fonte: elaboração própria.

Como se observa na Figura 22, o Grafana permite consultar as métricas recolhidas pelo *Prometheus* no intervalo da última hora. Nesta vista são apresentados indicadores relacionados com alertas, autenticação e autorização da aplicação, como a duração dos pedidos de autenticação, a quantidade de autenticações realizadas e as tentativas de autorização. Estes dados ajudam a acompanhar o comportamento do sistema, identificar picos de utilização e detetar possíveis problemas de desempenho ou segurança.

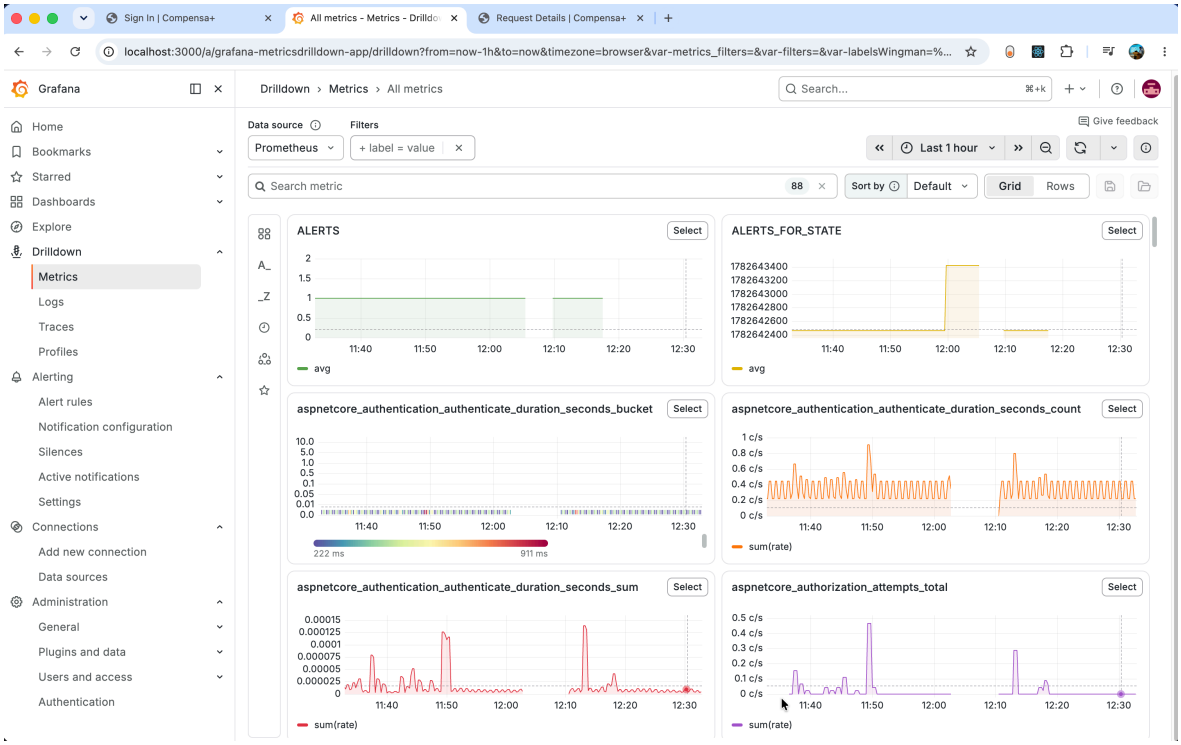


Figura 23 - Visualização geral das métricas da aplicação no Grafana através do Prometheus. Fonte: elaboração própria.

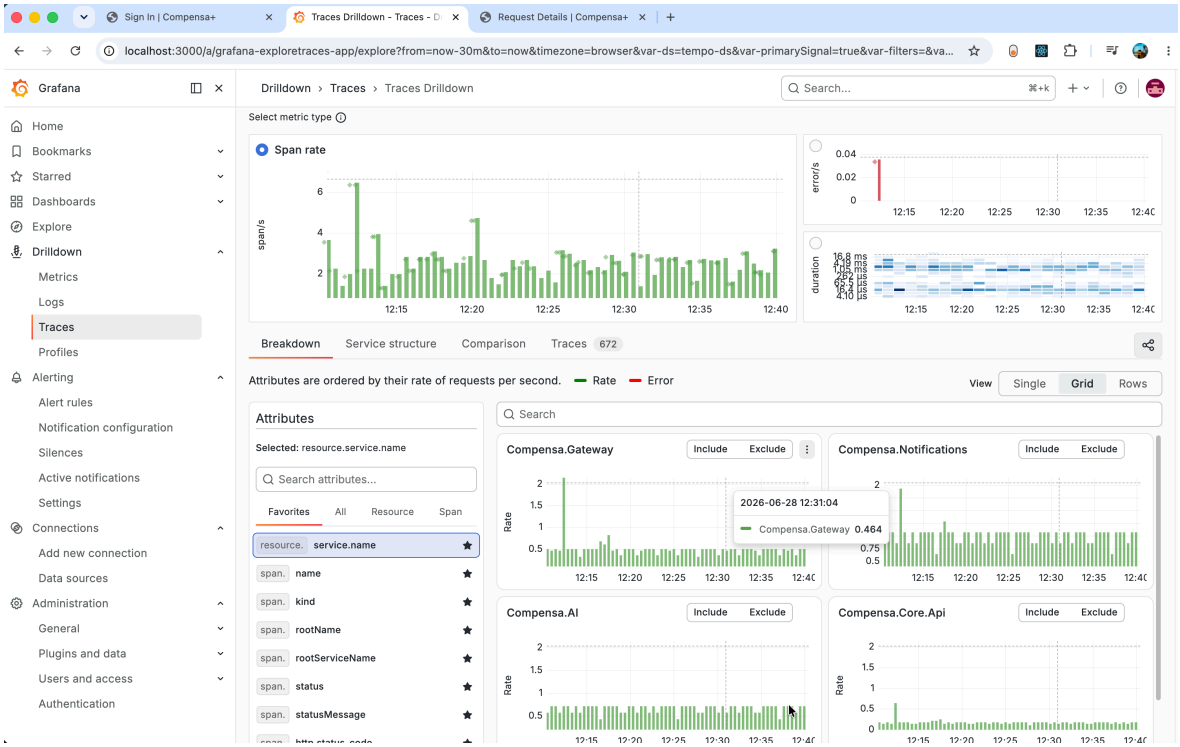


Figura 24 - Dashboard de monitorização traces no Grafana. Fonte: elaboração própria.

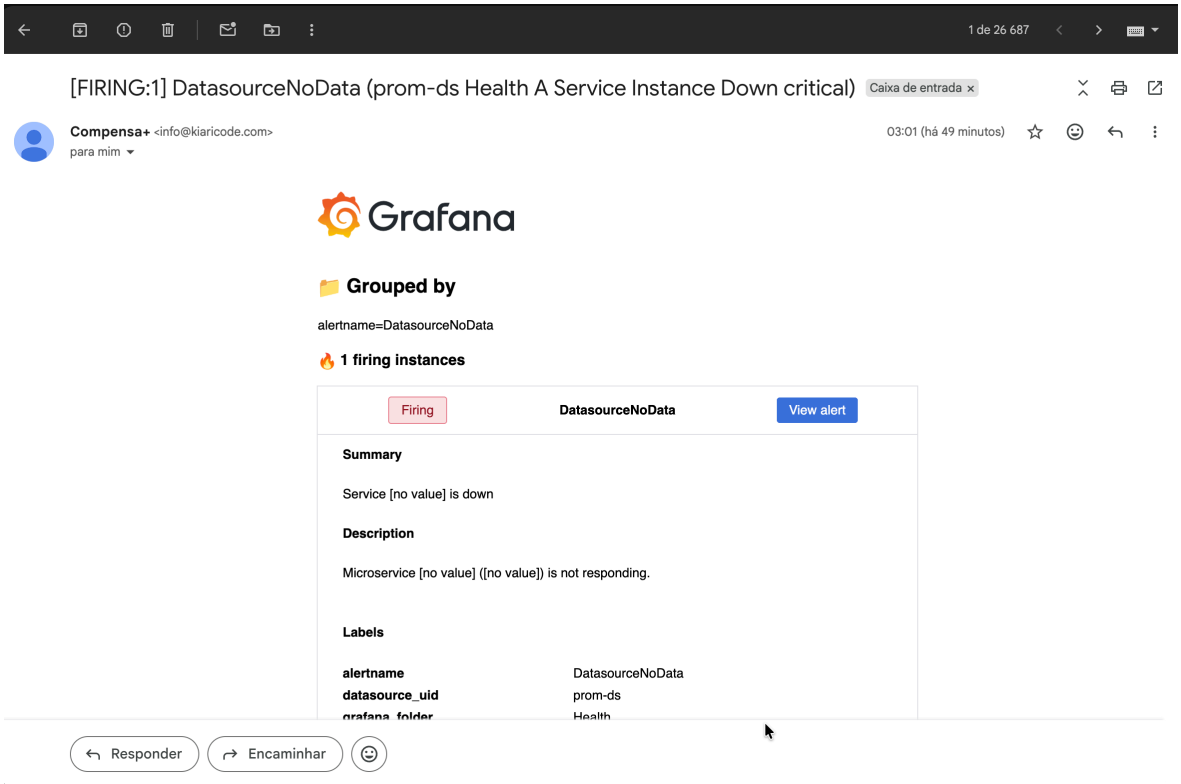


Figura 25 - Email de alerta enviado pelo grafana em caso de problemas na plataforma. Fonte: elaboração própria.

2.7.4. *Loki* e Tempo

O *Loki* é utilizado para agregação e pesquisa de *logs*, permitindo localizar erros e eventos relevantes nos diferentes containers. O Tempo é utilizado para armazenamento de traces distribuídos, possibilitando visualizar o caminho completo de uma requisição entre os microserviços.

Estas ferramentas complementam o sistema de observabilidade, permitindo uma visão mais completa do funcionamento interno da aplicação.

2.7.5. .NET Aspire Dashboard

O *.NET Aspire Dashboard* é utilizado durante o desenvolvimento para visualizar rapidamente métricas, *logs* e traces dos serviços .NET (Microsoft, 2025). Esta ferramenta facilita o *debug* em sistemas distribuídos, permitindo identificar problemas de comunicação entre microserviços e avaliar o comportamento da aplicação em tempo real.

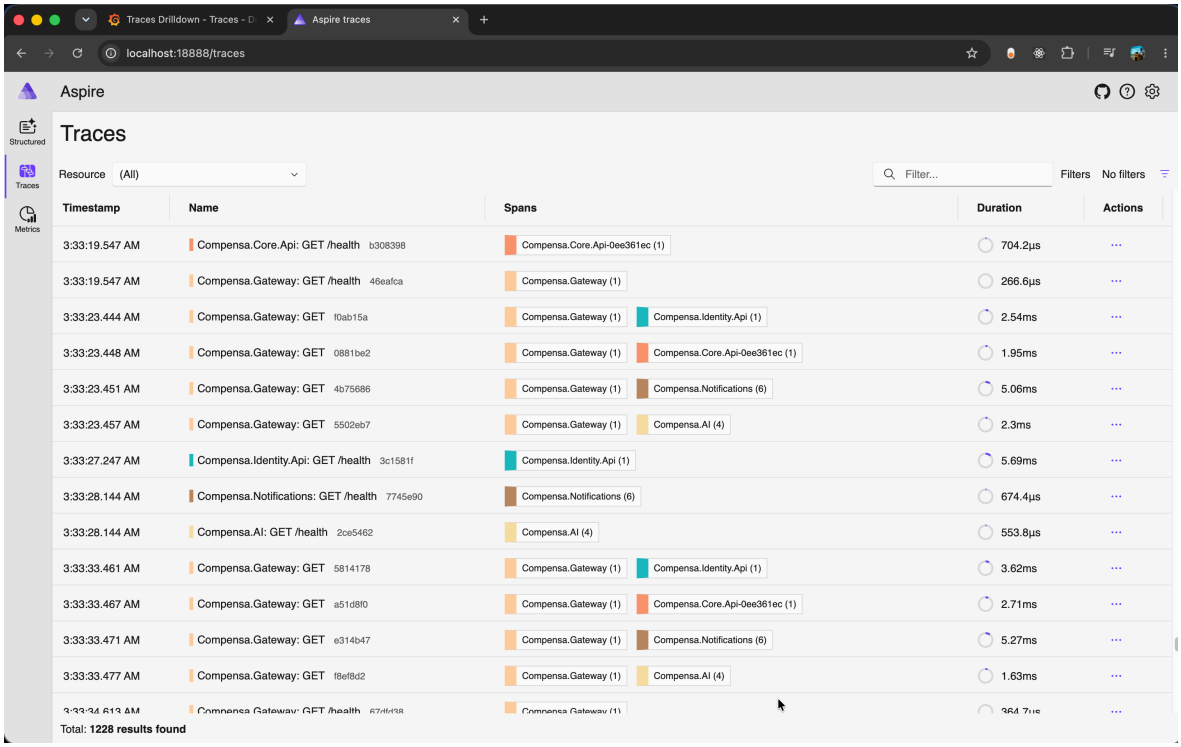


Figura 26 - Monitorização dos serviços através do .NET Aspire Dashboard. Fonte: elaboração própria.

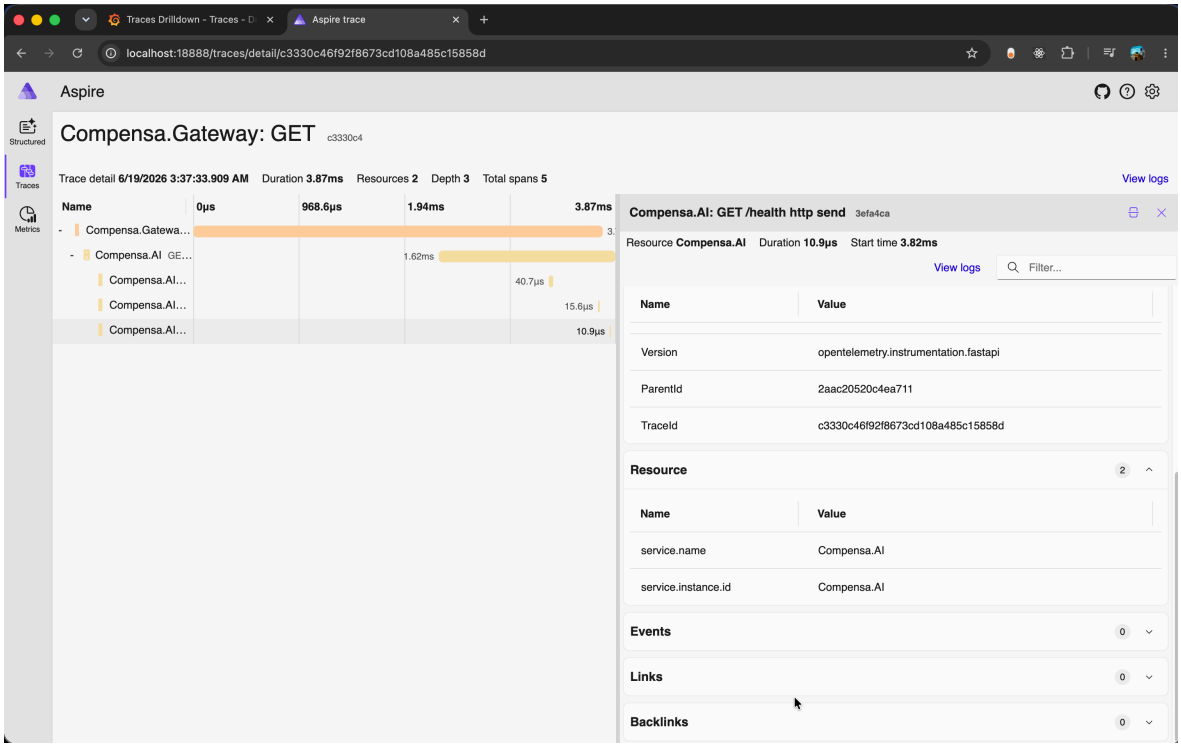


Figura 27 - Monitorização dos serviços através do .NET Aspire Dashboard. Fonte: elaboração própria.

2.7.6. *Portainer*

O *Portainer* é uma plataforma de gestão visual de containers que disponibiliza uma interface web para administração de ambientes *Docker e Docker Swarm*

(Portainer.io, 2025). No contexto da plataforma Compensa+, o *Portainer* assume o papel de painel de controlo operacional do cluster, permitindo gerir serviços, réplicas, volumes, redes e configurações sem necessidade de acesso direto à linha de comandos dos servidores.

A integração do *Portainer* com o *Docker Swarm* permite visualizar em tempo real o estado de cada serviço distribuído pelo cluster, incluindo o número de réplicas em execução, o nó onde cada container está alojado e o histórico de atualizações realizadas. Esta visibilidade é especialmente relevante numa arquitetura de microserviços como a da Compensa+, onde múltiplos serviços correm simultaneamente em diferentes nós.

Entre as principais funcionalidades do *Portainer* utilizadas na plataforma destacam-se:

- Visualização e gestão dos serviços do *Swarm* e respetivas réplicas;
- Monitorização do estado dos nós do *cluster (manager e workers);*
- Gestão de *stacks* através de ficheiros *docker-stack.yml*;
- Controlo de volumes e redes internas;
- Acesso a *logs* dos containers em tempo real;
- Gestão de registos de imagens Docker privados;
- Configuração de *secrets* e variáveis de ambiente de forma segura.

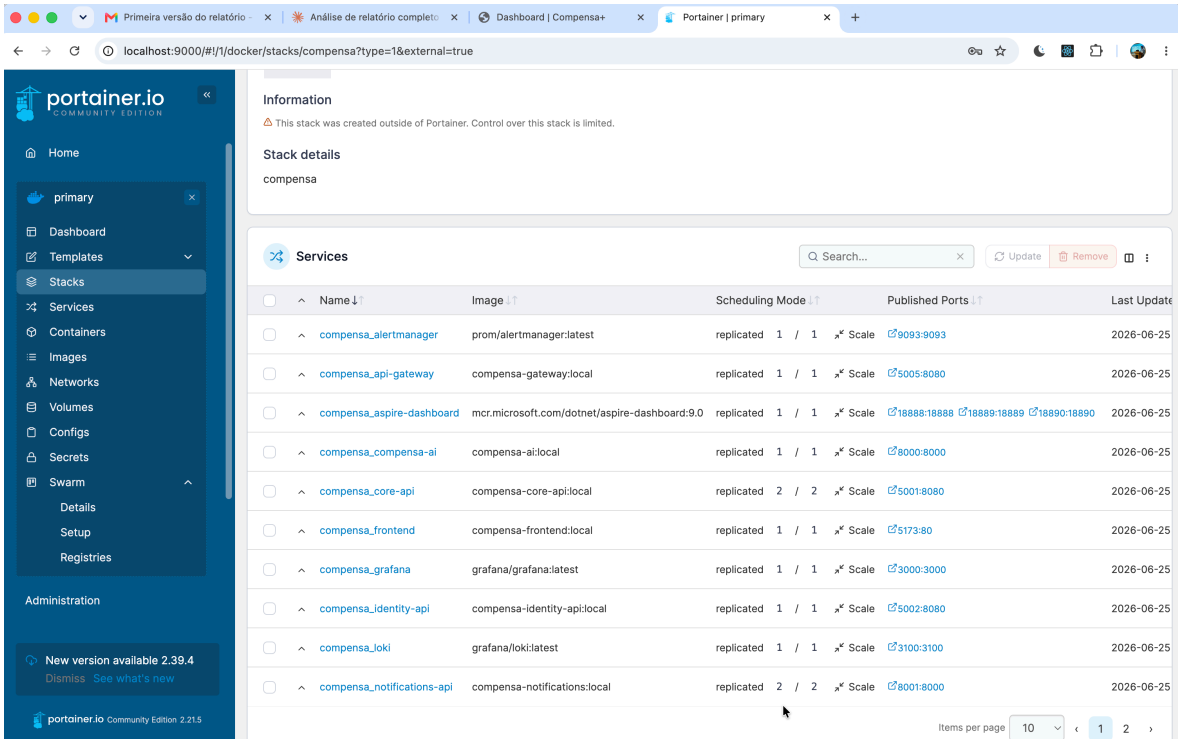


Figura 28 - Painel de monitorização de containers via Portainer. Fonte: elaboração própria.

O Portainer é executado numa máquina virtual dedicada à monitorização e operação da infraestrutura, juntamente com as restantes ferramentas de

observabilidade. O acesso à interface é restrito por autenticação e exposto apenas através de rede privada, garantindo que a gestão do cluster não está acessível publicamente.

2.7.7. Estratégia de *Deploy*

A estratégia de *deploy* da plataforma Compensa+ foi concebida para garantir reprodutibilidade, rastreabilidade, facilidade de manutenção e independência face ao ambiente de execução. Embora o objetivo principal seja disponibilizar a solução nos servidores do IPCB, a falta de acesso direto a essa infraestrutura durante o desenvolvimento levou à definição de uma estratégia alternativa baseada em máquinas virtuais na *Microsoft Azure*.

Apesar de a *Azure* ter sido utilizada como ambiente de validação, a arquitetura não depende de serviços específicos de um único fornecedor *cloud*. A plataforma foi preparada para poder ser executada em diferentes contextos, como ambiente *on-premises*, AWS ou *Azure*, desde que existam máquinas virtuais ou servidores com os requisitos necessários para correr os serviços aplicacionais, Docker e os componentes de suporte.

Para garantir esta flexibilidade, adotou-se uma abordagem baseada em três pilares: *Terraform*, para o provisionamento da infraestrutura; *Ansible*, para a configuração automatizada das máquinas; e Docker, para a execução dos serviços da aplicação. Esta combinação segue os princípios de *Infrastructure as Code (IaC),* nos quais a infraestrutura é descrita, versionada e aplicada a partir de ficheiros de configuração, reduzindo configurações manuais e diminuindo a probabilidade de erros operacionais (Morris, K., 2020).

Desta forma, a utilização da *Azure* neste projeto não representa uma dependência tecnológica definitiva, mas sim um ambiente escolhido para testar e demonstrar a estratégia de *deploy*. A mesma abordagem poderá ser adaptada futuramente para os servidores institucionais do IPCB ou para outro fornecedor *cloud*, mantendo a lógica de instalação, configuração e execução da plataforma.

2.7.8. Provisionamento com *Terraform*

O *Terraform* é utilizado para criar e gerir todos os recursos *Azure* necessários ao funcionamento da plataforma (HashiCorp, 2025). A partir de ficheiros de configuração declarativos, o *Terraform* provisiona automaticamente os seguintes recursos:

- *Resource Group* dedicado à plataforma Compensa+;

- Virtual Network (VNet) com sub-redes segmentadas por função (aplicação, dados, monitorização);
- *Network Security Groups* (NSG) com regras de *firewall* específicas para cada camada;
- Máquinas Virtuais com os tamanhos adequados a cada função;
- Discos geridos para persistência dos dados da base de dados;
- IP público para exposição controlada dos serviços externos;
- *Azure Key Vault* para armazenamento seguro de *secrets*, chaves e variáveis sensíveis.

A utilização do *Terraform* garante que a infraestrutura pode ser recriada de forma idêntica em qualquer momento, e que qualquer alteração à topologia é registada no sistema de controlo de versões.

2.7.9. Configuração com *Ansible*

Após o provisionamento da infraestrutura pelo *Terraform*, o *Ansible* é responsável pela configuração das máquinas virtuais (Red Hat, 2025). Através de *playbooks* declarativos, o *Ansible* realiza automaticamente as seguintes tarefas em cada máquina:

- Instalação e configuração do *Docker Engine*;
- Inicialização ou associação ao cluster *Docker Swarm*;
- Configuração de certificados TLS para comunicação segura entre nós;
- Instalação de dependências do sistema operativo;
- Aplicação de políticas de segurança e *hardening* das VMs;
- Transferência e aplicação dos ficheiros de *stack* e configuração.

2.7.10. Topologia de Máquinas Virtuais

A infraestrutura de produção da Compensa+ é composta por cinco máquinas virtuais *Azure*, cada uma com responsabilidades específicas e dimensionada de acordo com a carga esperada. A tabela seguinte apresenta a topologia definida:

Tabela 4 - Topologia de Máquinas Virtuais. Fonte: elaboração própria.

Máquina Virtual	Finalidade	Tamanho Azure	Serviços
compensa-swarm-manager	Manager do Swarm	Standard_B2ms	API Gateway, Nginx,

compensa-backend	Manager do Swarm + APIs	Standard_B2ms	Core API, Identity API, Notifications API, Compensa AI API,
compensa-db	Base de dados	Standard_B2ms + disco 64GB	PostgreSQL
compensa-cache	Cache e mensageria	Standard_B2s	Redis, RabbitMQ
compensa-ops	Operações e monitorização	Standard_B2s	Portainer, Grafana, Prometheus, Loki, Tempo, OpenTelemetry Collector, .NET Aspire Dashboard

Todas as VMs estão ligadas à mesma *Virtual Network* privada, comunicando entre si através de IPs internos. Apenas os serviços que necessitam de acesso externo como *API Gateway e Nginx* estão expostos com endereço IP público, protegidos por NSG.

2.7.11. Cluster Docker Swarm na Azure

Após a configuração das VMs pelo *Ansible*, o *Docker Swarm* é inicializado na VM compensa-swarm-manager, que assume o papel de nó manager do cluster. As restantes VMs podem ser adicionadas como worker nodes conforme a necessidade de escalar a capacidade de processamento.

Os serviços da plataforma são deployados através do ficheiro *docker-stack.yml,* que define as imagens, variáveis de ambiente, redes, volumes e configuração de réplicas de cada Microserviço. O *deploy* é realizado com o comando *docker stack deploy*, que o *Swarm* executa de forma distribuída pelos nós disponíveis.

2.7.12. Estratégia para o *Frontend* com *Nginx*

O *frontend ReactJS* é compilado em tempo de build, gerando um conjunto de ficheiros estáticos otimizados. Estes ficheiros são servidos pelo Nginx, que corre num container *Docker* gerido pelo *Swarm* na VM c*ompensa-swarm-manager.*

O Nginx é configurado para:

- Servir os ficheiros estáticos do *frontend* com cache otimizado;
- Gerir o roteamento da *Single Page Application*, redirecionando todas as rotas para o index.html;
- Atuar como reverse proxy para o *API Gateway*, centralizando o tráfego num único ponto de entrada;
- Terminar a ligação TLS, garantindo comunicação cifrada com os clientes.

2.7.13. Pipeline de Deploy

O fluxo completo de *deploy* da plataforma Compensa+ segue a seguinte sequência:

1. O código é submetido ao repositório *GitHub*;
2. O *Terraform* aplica as alterações à infraestrutura *Azure*, se necessário;
3. O *Ansible* configura ou atualiza as VMs afetadas;
4. As imagens Docker dos microserviços são construídas e publicadas num registo privado;
5. O ficheiro *docker-stack.yml* é atualizado com as novas versões das imagens;
6. O *Docker Swarm* executa o *rolling update* dos serviços afetados, substituindo réplicas uma a uma sem interrupção de serviço;
7. O *Portainer* reflete em tempo real o estado do deploy e permite acompanhar o progresso das atualizações.

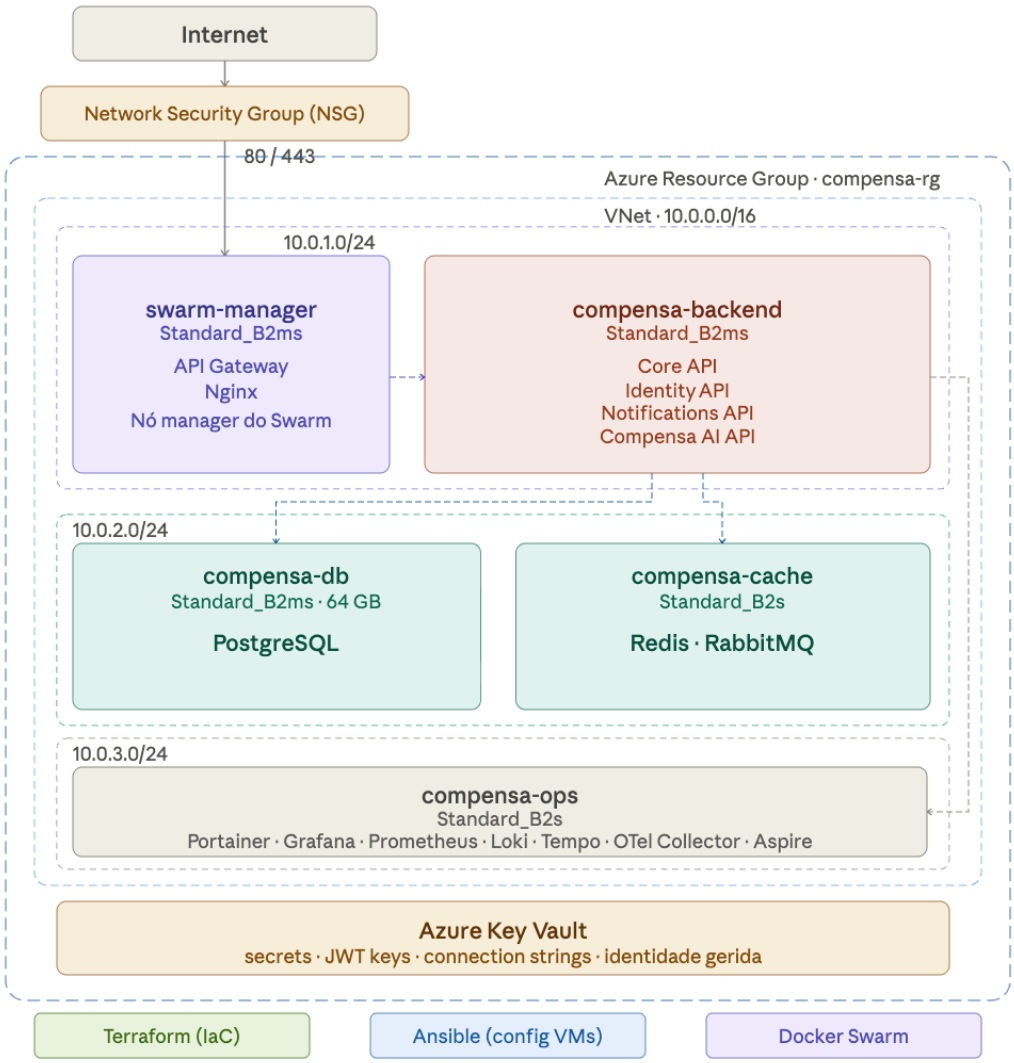


Figura 29 - Topologia de infraestrutura de produção da plataforma Compensa+ na Microsoft Azure. Fonte: elaboração própria.

2.8. Tecnologias Utilizadas

A Tabela 4 apresenta um resumo das principais tecnologias utilizadas no desenvolvimento da plataforma Compensa+.

Tabela 5 - Tecnologias utilizadas na plataforma Compensa+. Fonte: elaboração própria.

Tecnologia	Finalidade
ReactJS	Desenvolvimento do frontend web
TypeScript	Tipagem estática e manutenção do frontend
Vite	Ferramenta de desenvolvimento e build do frontend
Tailwind CSS	Estilização e design responsivo
TanStack Query	Gestão de dados e cache no cliente
Axios	Comunicação HTTP com as APIs
.NET 10	Desenvolvimento dos microserviços backend
ASP.NET Core Identity	Autenticação e gestão de utilizadores
Entity Framework Core	ORM e acesso à base de dados
FastAPI	Desenvolvimento da API de inteligência artificial
PostgreSQL	Persistência de dados
Redis	Cache distribuída
RabbitMQ	Comunicação assíncrona por eventos
YARP	API Gateway e reverse proxy
Docker	Containerização dos serviços
Docker Compose	Orquestração local dos containers
OpenTelemetry	Recolha de métricas, logs e traces
Prometheus	Monitorização e armazenamento de métricas
Grafana	Visualização de dashboards
Loki	Agregação de logs
Tempo	Rastreamento distribuído
Jira	Gestão de tarefas e fluxo Kanban
GitHub	Controlo de versões e colaboração
Postman	Testes manuais da API
Playwright	Testes automatizados end-to-end
Portainer	Gestão visual dos containers Docker
Terraform	Provisionamento da infraestrutura como código
Ansible	Automação da configuração e deploy



Figura 30 - Stack tecnológica utilizada no desenvolvimento da plataforma Compensa+. Fonte: elaboração própria.

2.9. Considerações Finais

A arquitetura definida para a plataforma Compensa+ foi concebida com o objetivo de garantir modularidade, escalabilidade, segurança e facilidade de manutenção. A adoção de uma arquitetura baseada em microserviços permite separar os diferentes domínios funcionais do sistema, tornando a solução mais flexível e preparada para evoluções futuras.

A utilização de comunicação síncrona através de *APIs REST*, combinada com comunicação assíncrona baseada em eventos, permite equilibrar simplicidade, desempenho e resiliência. Adicionalmente, a integração de tecnologias como *Redis*, *RabbitMQ*, Docker e ferramentas de observabilidade contribui para melhorar o desempenho, a disponibilidade e a capacidade de monitorização do sistema.

O *frontend* em *ReactJS*, aliado a *TypeScript, Vite e Tailwind CSS*, permite construir uma interface moderna, responsiva e orientada à experiência do utilizador. Por sua vez, o *backend* baseado em *.NET e FastAPI* oferece uma base sólida para implementação das regras de negócio, autenticação, notificações e funcionalidades inteligentes.

Desta forma, a arquitetura e o conjunto tecnológico adotados constituem a base para a implementação funcional da plataforma Compensa+, apresentada no capítulo seguinte.

3. Implementação do Sistema

O presente capítulo descreve a implementação da plataforma Compensa+, tendo como base a arquitetura definida no capítulo anterior e os requisitos identificados durante a fase de análise e modelação realizada no Projeto I. Esta etapa teve como principal objetivo transformar o modelo conceptual, os casos de uso e o protótipo anteriormente definidos numa aplicação funcional, capaz de apoiar a gestão de aulas de compensação no contexto académico.

Ao contrário do Capítulo 2, que apresenta a visão arquitetural e tecnológica da solução, este capítulo centra-se na forma como as funcionalidades foram efetivamente implementadas. Assim, são descritas as principais interfaces da aplicação, a organização do código, a estrutura interna dos microserviços, os serviços aplicacionais, as entidades de domínio, os repositórios e os principais fluxos de integração.

A implementação foi organizada em duas dimensões principais: o frontend web, responsável pela interação com os utilizadores, e o backend, responsável pela autenticação, regras de negócio, persistência de dados, notificações, importação de horários, assistência inteligente e integração entre os diferentes componentes do sistema.

A plataforma foi desenvolvida de forma modular, contemplando funcionalidades como autenticação, gestão de utilizadores, painel principal, criação e validação de pedidos de compensação, calendário académico, gestão de cursos, turmas, unidades curriculares, salas, importação de horários, notificações e assistente inteligente.

3.1. Organização do Desenvolvimento com *Kanban*

A implementação foi acompanhada através de uma organização baseada em *Kanban*, com recurso à ferramenta Jira. O quadro foi estruturado em diferentes estados de desenvolvimento, nomeadamente *To Do, Blocked, In Progress, In Review e Done*, permitindo acompanhar a evolução de cada tarefa ao longo do projeto.

As tarefas foram descritas sob a forma de histórias de utilizador, seguindo uma lógica próxima dos requisitos funcionais definidos no Projeto I. Por exemplo, foram criadas tarefas como “Eu como coordenador quero poder aprovar ou rejeitar um pedido de compensação de aula”, “Eu como professor quero poder criar pedidos de compensação” e “Eu como *admin* quero poder adicionar novos utilizadores na plataforma”. Esta abordagem permitiu relacionar diretamente as funcionalidades implementadas com as necessidades dos diferentes perfis de utilizador.

A Figura 22 apresenta o quadro *Kanban* utilizado no *Jira* para organização das tarefas de implementação da plataforma Compensa+.

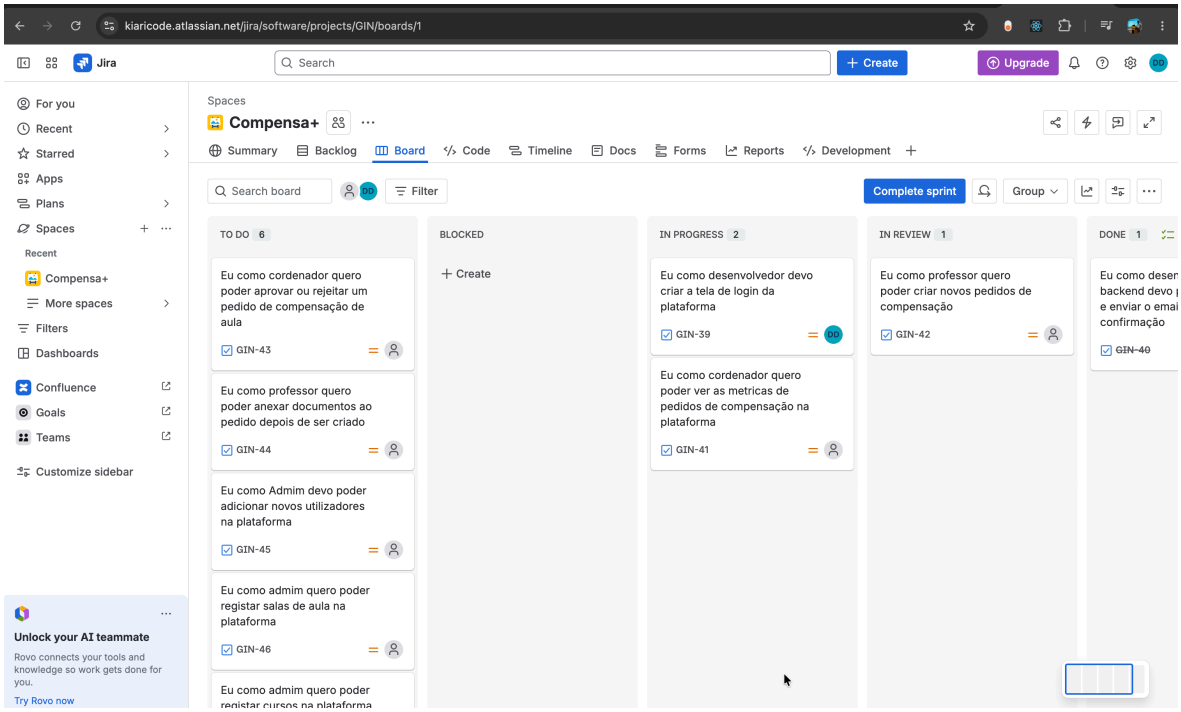


Figura 31 - Quadro Kanban utilizado para organização das tarefas de implementação no Jira. Fonte: elaboração própria.

Como se observa na Figura 22, as tarefas encontram-se distribuídas por diferentes estados, permitindo identificar rapidamente o que ainda estava por iniciar, o que se encontrava em desenvolvimento, o que estava em revisão e o que já tinha sido concluído. A utilização do *Jira* facilitou a distribuição de responsabilidades, o acompanhamento do progresso da equipa e a identificação de tarefas em curso, em revisão ou já concluídas. Desta forma, a ferramenta serviu como apoio à gestão operacional da implementação, sem substituir a metodologia *ICONIX* utilizada no Projeto I para análise e modelação do sistema.

3.2. Organização Geral do Projeto

A implementação da plataforma Compensa+ foi organizada numa estrutura de repositório que separa os diferentes componentes da solução, nomeadamente *frontend, backend,* base de dados, configurações *Docker, scripts* auxiliares e documentação técnica.

A estrutura geral do projeto foi organizada da seguinte forma:
CompensaProject/
Backend/
Compensa.Core.Api/

Compensa.Identity.Api/
Compensa.Gateway/
Compensa.Notifications/
CompensaAI/
Database/
Frontend/
TimetableSource/
docker/
alertmanager/
grafana/
loki/
otel-collector/
prometheus/
tempo/
docs/
guidelines/
scripts/
docker-compose.yml
docker-stack.yml
docker-compose.database.yml
.env.example

A Figura 32 apresenta a estrutura geral do repositório da plataforma, evidenciando a separação entre os microserviços *backend*, a aplicação *frontend*, os ficheiros de infraestrutura e os scripts de apoio.

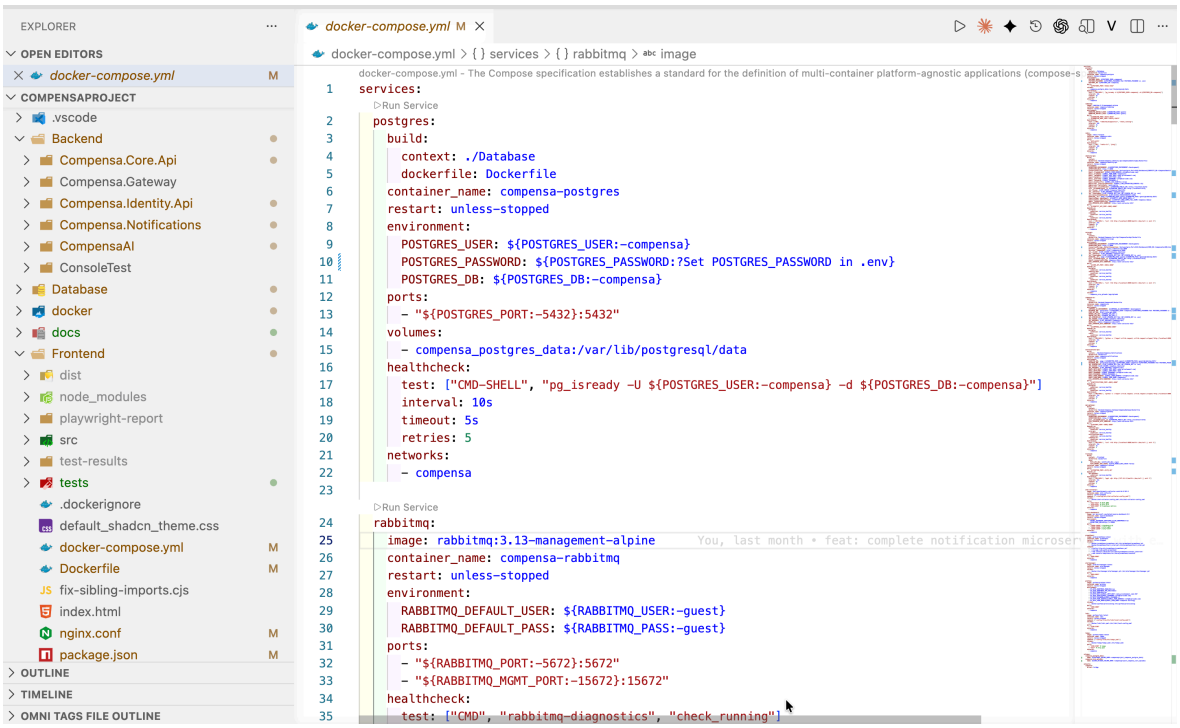


Figura 32 - Estrutura geral do repositório da plataforma Compensa+. Fonte: elaboração própria.

Como apresentado na Figura 32, a pasta *Backend* concentra os microserviços da aplicação, como *Core API, Identity API, Gateway, Notifications API* e CompensaAI. A pasta *Frontend* contém a aplicação *ReactJS*, enquanto a pasta *docker* reúne configurações relacionadas com observabilidade e infraestrutura.

Esta separação tornou o projeto mais modular e facilitou a evolução independente de cada componente.

Esta organização permitiu que cada serviço pudesse ser desenvolvido, configurado e testado de forma mais independente, reduzindo o acoplamento entre os diferentes módulos da plataforma.

3.3. Relação entre o Projeto I e a Implementação

A implementação da plataforma foi orientada pelos artefactos produzidos no Projeto I, nomeadamente requisitos funcionais, casos de uso, modelo de domínio, diagramas UML e protótipo de interface. Desta forma, o Projeto II representou a concretização prática da solução previamente analisada e modelada.

A Tabela 6 apresenta a relação entre alguns dos casos de uso definidos no Projeto I e a respetiva implementação realizada no Projeto II.

Tabela 6 - Relação entre casos de uso do Projeto I e funcionalidades implementadas no Projeto II. Fonte: elaboração própria.

Caso de uso definido no Projeto I	Implementação no Projeto II
Autenticar utilizador	Implementação de autenticação por link mágico e emissão de JWT
Submeter pedido de compensação	Formulário de criação de pedido e endpoint de submissão
Consultar pedidos	Listagem de pedidos em grelha e em tabela
Aprovar pedido	Ação de aprovação disponível para coordenadores
Recusar pedido	Ação de rejeição disponível para coordenadores
Consultar calendário	Módulo de calendário académico
Gerir cursos e turmas	Módulo de cursos, unidades curriculares e turmas
Gerir salas	Página de gestão de salas e respetivos endpoints
Enviar notificação	Publicação de eventos e processamento pela Notifications API
Importar horários	Módulo de importação automática de ficheiros HTML
Apoiar utilizador	Assistente inteligente integrado com a CompensaAI API

Esta relação demonstra que os modelos definidos na fase de análise foram utilizados como base para a implementação real da plataforma, garantindo coerência entre a solução conceptual e o sistema desenvolvido.

3.4. Implementação do *Frontend Web*

O *frontend* da plataforma Compensa+ foi desenvolvido com *ReactJS, Vite, TypeScript e Tailwind CSS.* Esta camada disponibiliza a interface *web* utilizada pelos diferentes perfis de utilizador, permitindo aceder às funcionalidades de acordo com as permissões associadas a cada papel.

A aplicação *frontend* foi organizada de forma modular, com separação entre componentes reutilizáveis, páginas, serviços, rotas, *providers*, tipos e utilitários. Esta organização permitiu reduzir duplicação de código, melhorar a manutenção e centralizar a comunicação com os microserviços.

A estrutura principal do *frontend* encontra-se organizada da seguinte forma:
Frontend/
src/
app/
components/
common/
core/
domain/
ui/
pages/
AiConverter/
Calendar/
Classrooms/
CoordinatorRequests/
Courses/
Dashboard/
ImportSchedules/
Login/
Notifications/
Requests/
SystemCalendar/
TeacherRequests/
Users/
academicYears/
providers/
routes/

services/
academicYears/
api/
assignments/
auth/
classrooms/
compensationRequests/
courses/
dashboard/
holidays/
notifications/
schedules/
search/
users/
types/
utils/
styles/

A Figura 33 apresenta a estrutura do projeto *frontend*, onde é possível observar a organização das páginas e dos serviços utilizados para comunicar com o *backend*.



Figura 33 - Estrutura do projeto frontend da plataforma Compensa+. Fonte: elaboração própria.

Como se verifica na Figura 33, as páginas principais da aplicação encontram-se organizadas em diretórios como *Dashboard, Requests, Courses, Calendar, ImportSchedules, Notifications, Users e Classrooms*. A pasta *services* centraliza as chamadas às APIs, permitindo que os componentes da interface não dependam diretamente dos *endpoints* dos microserviços.

A comunicação com o *backend* é realizada através do API Gateway. O *frontend* envia pedidos HTTP para o *Gateway*, que encaminha cada requisição para o Microserviço correspondente. Esta abordagem evita que a interface tenha de conhecer diretamente os endereços internos de cada serviço.

3.4.1. Autenticação e Gestão de Utilizadores

A autenticação constitui o ponto de entrada da plataforma Compensa+. O fluxo normal da aplicação utiliza autenticação por link mágico, dispensando a utilização de password tradicional. O utilizador introduz o seu email, recebe um *link* temporário e, ao aceder a esse *link*, obtém uma sessão autenticada através de token JWT.

A Figura 34 apresenta a página de autenticação da plataforma Compensa+, utilizada pelos diferentes perfis para iniciar sessão.

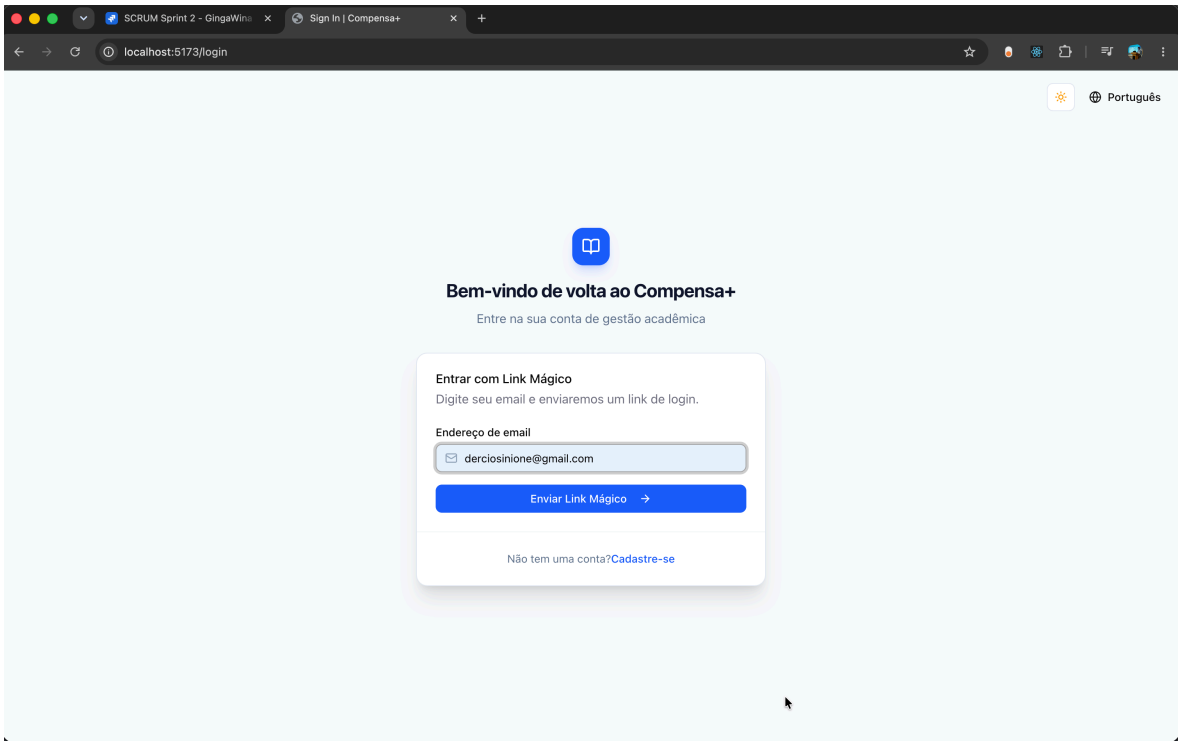


Figura 34 - Página de autenticação da plataforma Compensa+. Fonte: elaboração própria.

Como ilustrado na Figura 34, a página de *login* foi desenvolvida com uma interface simples e direta, permitindo ao utilizador iniciar o processo de autenticação. Após a validação, o sistema redireciona o utilizador para o painel principal, apresentando apenas as funcionalidades compatíveis com o seu perfil.

A criação e gestão de utilizadores estão associadas aos perfis existentes na plataforma, nomeadamente Administrador, Coordenador e Docente. Esta distinção

permite controlar permissões diretamente na interface e impedir que funcionalidades sensíveis sejam apresentadas a utilizadores sem autorização.

A Figura 35 apresenta a interface de criação ou gestão de utilizadores na plataforma.

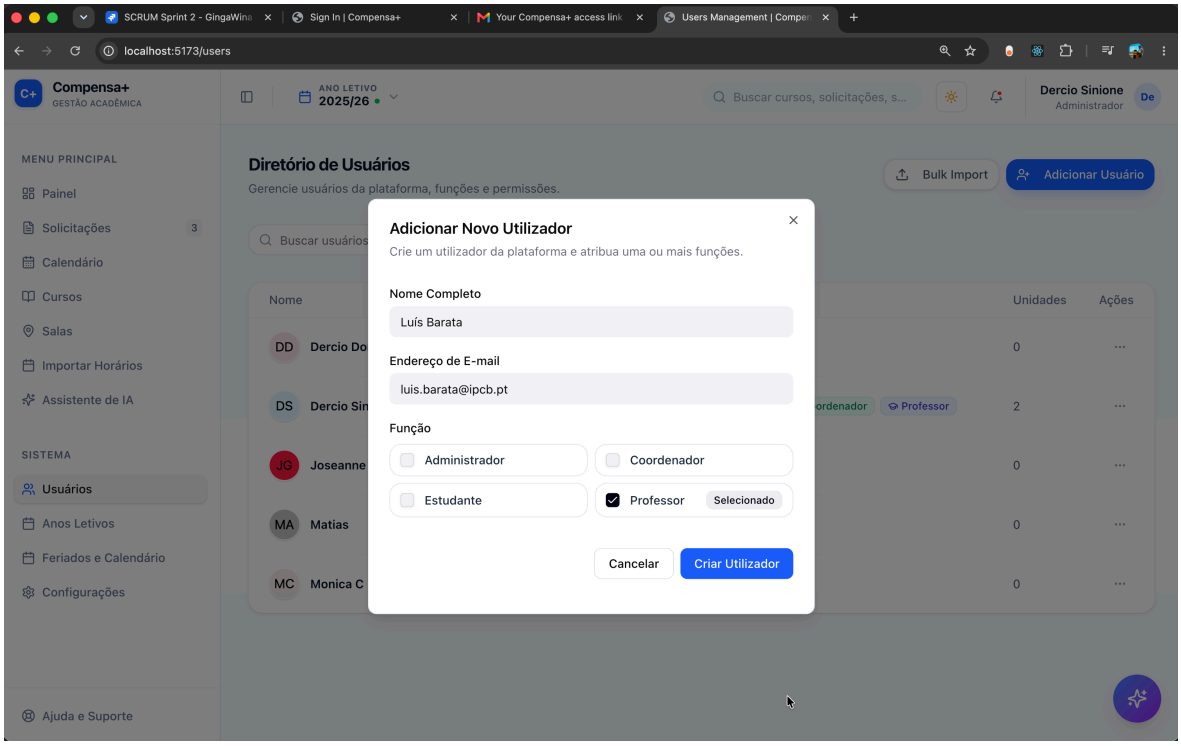


Figura 35 - Criação e gestão de utilizadores na plataforma Compensa+. Fonte: elaboração própria.

Como se observa na Figura 35, a interface permite registar utilizadores e associar perfis de acesso. Esta organização garante que ações sensíveis, como aprovação de pedidos, gestão de salas ou configuração de dados académicos, sejam disponibilizadas apenas aos utilizadores autorizados.

3.4.2. Painel Principal

Após a autenticação, o utilizador é direcionado para o painel principal. Esta página apresenta uma visão geral da plataforma e disponibiliza indicadores relacionados com os pedidos de compensação.

O painel principal foi implementado com cartões estatísticos e elementos visuais que apresentam dados agregados, como pedidos pendentes, aprovados, recusados e prioritários. Estes dados são obtidos a partir da Core API, que agrega a informação persistida no *PostgreSQL* e a disponibiliza ao *frontend*.

A Figura 36 apresenta o painel principal da plataforma Compensa+, onde são exibidos dados agregados sobre os pedidos e o estado geral do sistema.

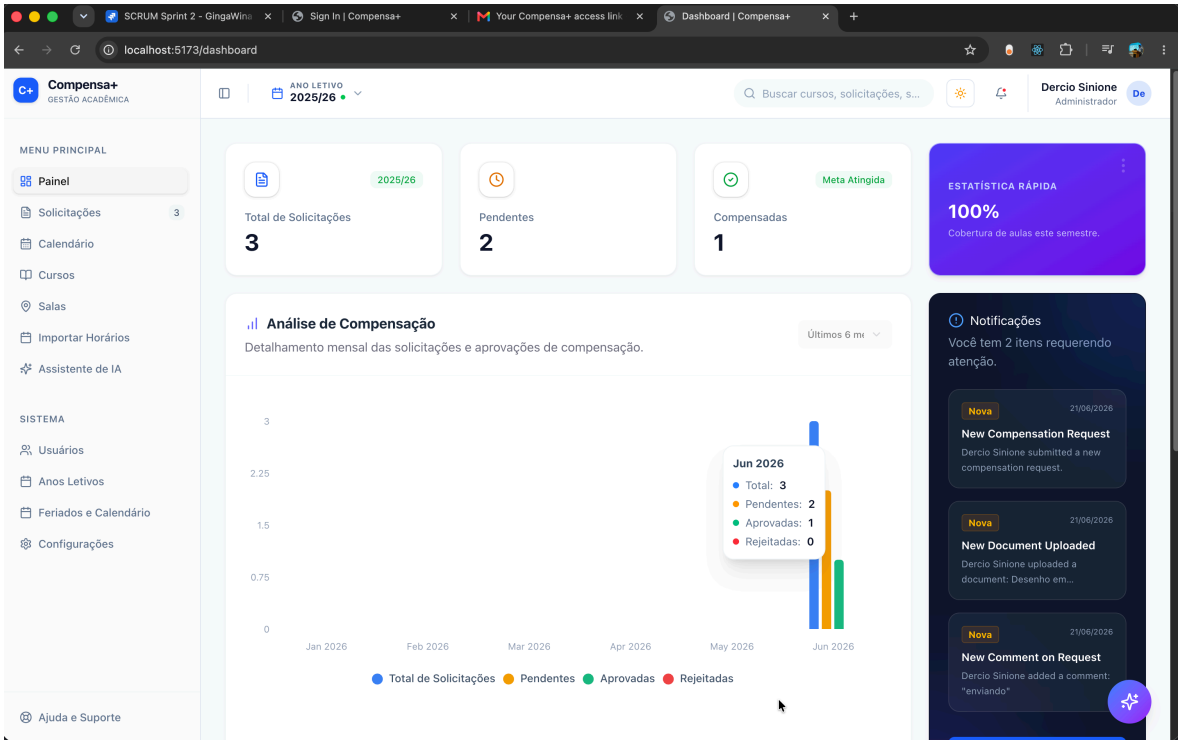


Figura 36 - Painel principal da plataforma Compensa+. Fonte: elaboração própria.

Como apresentado na Figura 36, o painel principal disponibiliza cartões e indicadores que permitem identificar rapidamente pedidos pendentes, aprovados, recusados ou prioritários. Esta visualização apoia coordenadores e administradores na tomada de decisão e no acompanhamento global da plataforma.

3.4.3. Criação e Gestão de Pedidos de Compensação

A gestão de pedidos de compensação constitui uma das funcionalidades centrais da plataforma. No *frontend*, esta funcionalidade foi implementada através de páginas distintas para criação, consulta, acompanhamento e detalhe dos pedidos.

A listagem de pedidos disponibiliza duas formas de visualização: grelha e tabela. A vista em grelha apresenta os pedidos em cartões, permitindo uma leitura visual rápida. A vista em tabela permite comparar vários pedidos de forma mais detalhada, apresentando campos como curso, unidade curricular, horário original, horário proposto, estado e ações disponíveis.

A Figura 37 apresenta a listagem de pedidos em formato de grelha, onde cada pedido é apresentado em forma de cartão.

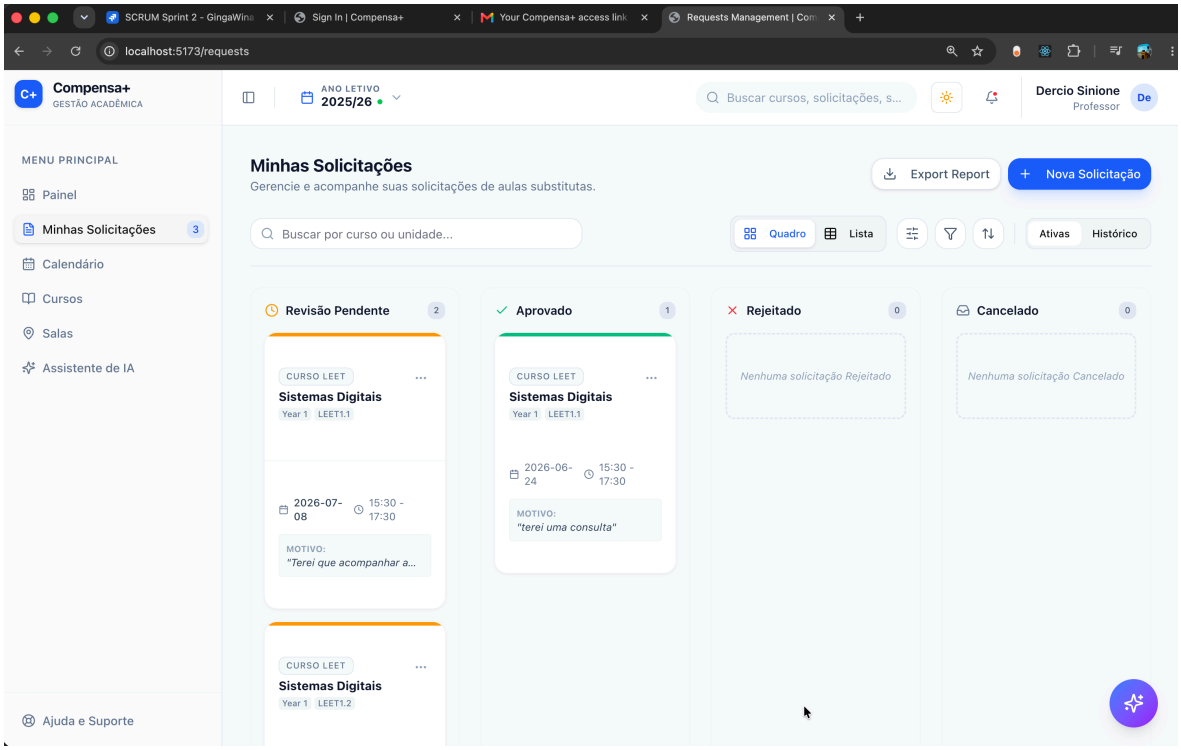


Figura 37 - Listagem de pedidos de compensação em grelha. Fonte: elaboração própria.

Como se observa na Figura 37, a vista em grelha facilita uma leitura rápida do estado dos pedidos, permitindo identificar visualmente informações como curso, unidade curricular, horário e estado da solicitação.

A Figura 38 apresenta a listagem de pedidos em formato de tabela, permitindo comparar vários pedidos de forma mais detalhada.

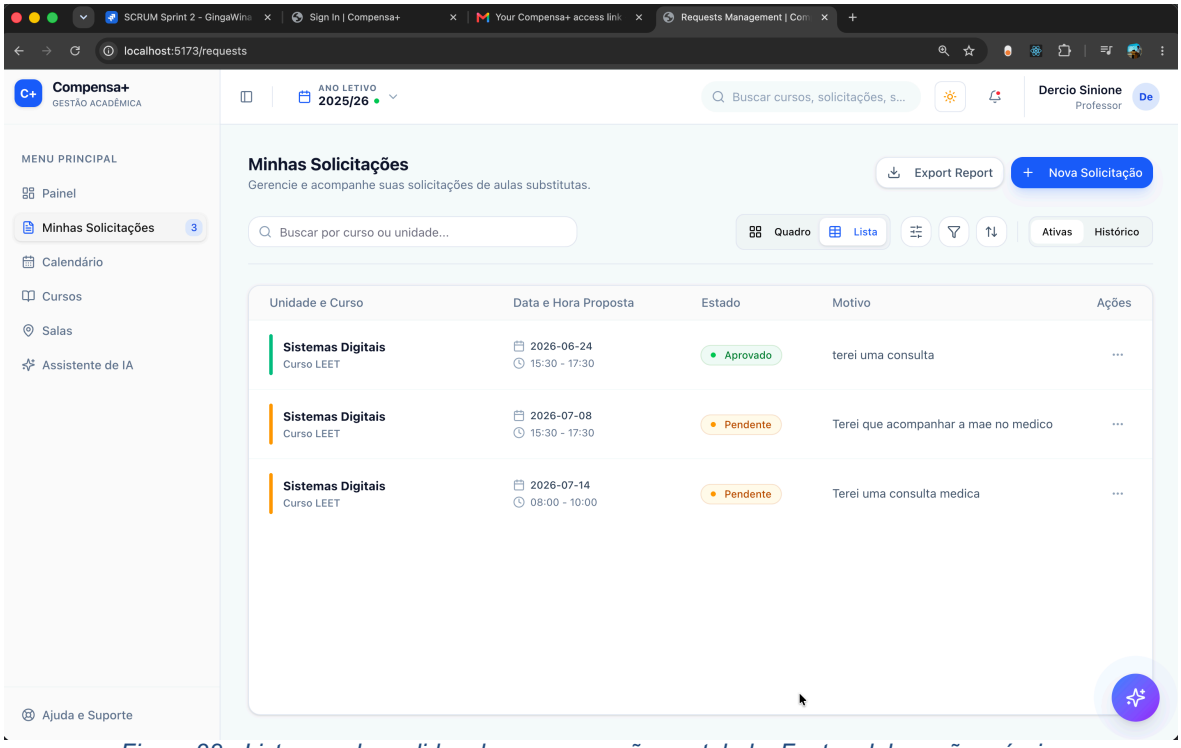


Figura 38 - Listagem de pedidos de compensação em tabela. Fonte: elaboração própria.

Como apresentado na Figura 38, a vista em tabela é especialmente útil para coordenadores e administradores, pois permite analisar vários pedidos em simultâneo, comparando campos como docente, curso, horário original, horário proposto, estado e ações disponíveis.

A criação de pedido permite ao docente submeter uma nova solicitação de compensação. O formulário recolhe informações como curso, unidade curricular, turma, horário original, horário proposto, sala e motivo da compensação. Foram utilizados campos estruturados e listas de seleção para reduzir erros de preenchimento e garantir consistência dos dados enviados ao *backend*.

A Figura 39 apresenta o formulário de criação de pedido de compensação implementado na plataforma.

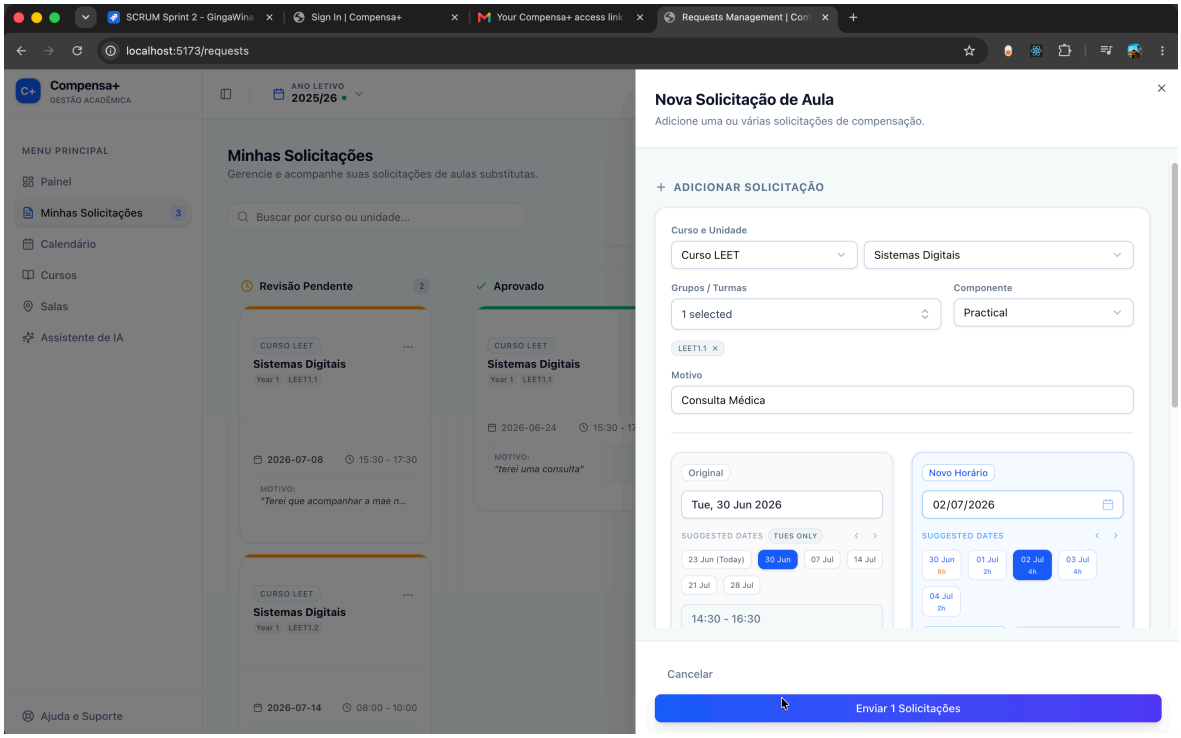


Figura 39 - Formulário de criação de pedido de compensação. Fonte: elaboração própria.

Como se verifica na Figura 39, o formulário utiliza campos estruturados e listas de seleção, reduzindo a probabilidade de erros de preenchimento. Após a submissão, o pedido é enviado para a *Core API* através do *API Gateway*, onde são aplicadas validações e regras de negócio antes do registo definitivo.

A página de detalhes apresenta a informação completa do pedido. Para o docente, esta página permite acompanhar o estado da solicitação. Para o coordenador, disponibiliza ações de aprovação ou rejeição, demonstrando a aplicação prática do controlo de permissões.

A Figura 40 apresenta a página de detalhes do pedido na perspetiva do coordenador, incluindo as ações de aprovação e rejeição.

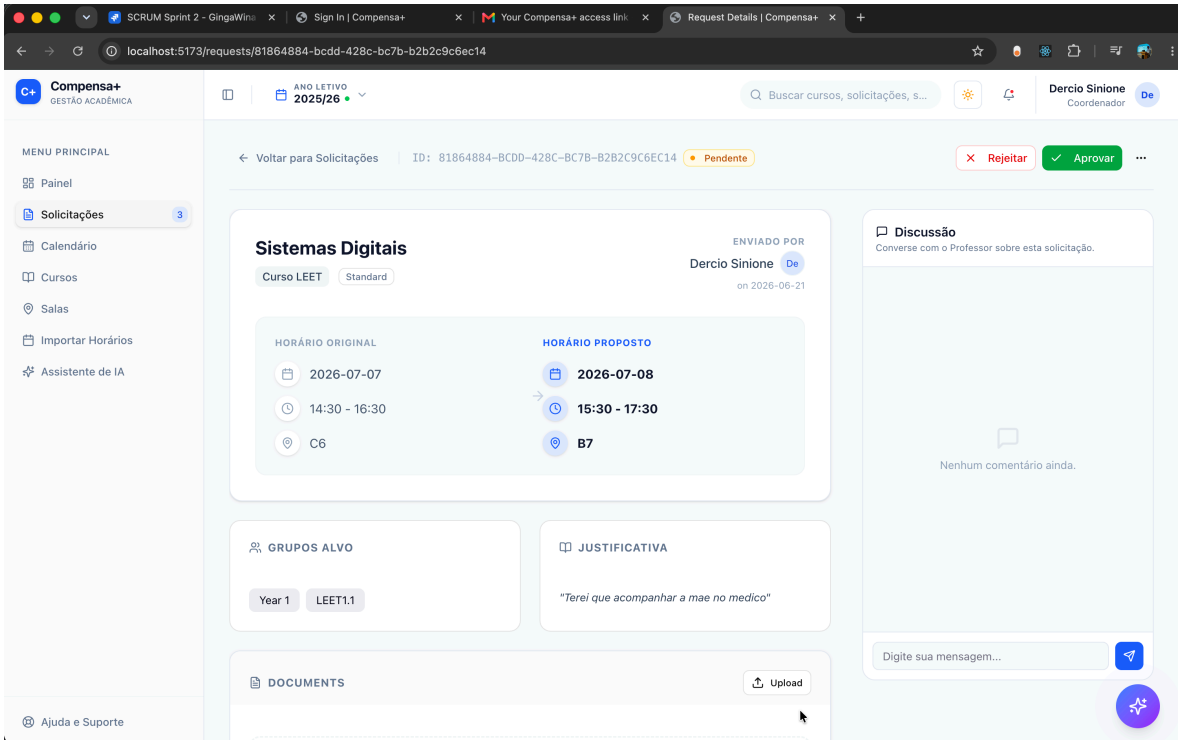


Figura 40 - Detalhes do pedido e ações de validação pelo coordenador. Fonte: elaboração própria.

Como se observa na Figura 40, o coordenador tem acesso às informações necessárias para analisar o pedido e tomar uma decisão. A presença das ações de aprovação e rejeição demonstra a aplicação prática do controlo de permissões, uma vez que estas opções não são disponibilizadas a utilizadores sem autorização.

3.4.4. Calendário Académico

O calendário académico permite visualizar eventos, aulas, feriados, pausas letivas e períodos relevantes para a gestão das compensações. Esta funcionalidade apoia a validação de horários e evita marcações em datas indisponíveis.

A Figura 41 apresenta o calendário académico implementado na plataforma Compensa+.

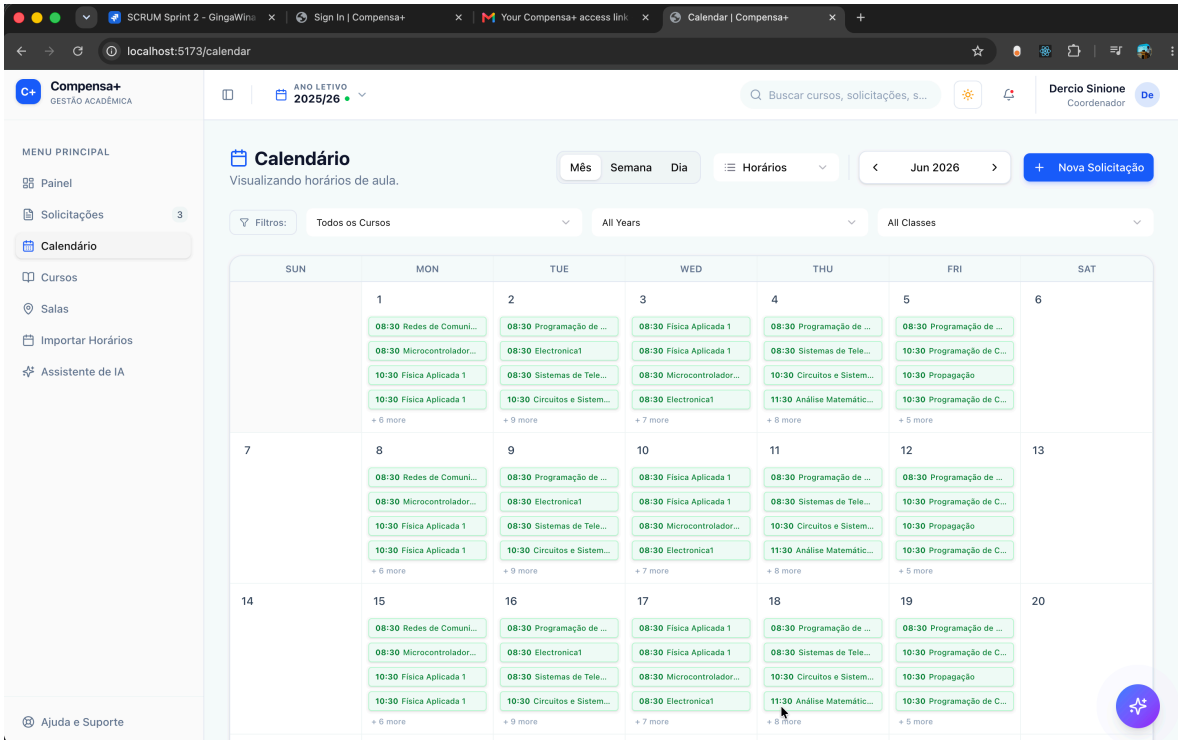


Figura 41 - Calendário académico da plataforma Compensa+. Fonte: elaboração própria.

Como apresentado na Figura 41, o calendário permite consultar a organização temporal das atividades académicas. Esta visualização contribui para que docentes e coordenadores identifiquem períodos disponíveis antes de propor ou validar uma compensação de aula.

A integração do calendário com os pedidos de compensação permite melhorar a organização das aulas, evitando sobreposições e facilitando a validação de horários.

3.4.5. Cursos, Turmas, Unidades Curriculares e Salas

Para suportar corretamente a criação e validação de pedidos, foram implementados módulos de gestão académica, nomeadamente cursos, unidades curriculares, turmas e salas.

O módulo de cursos permite gerir a estrutura académica da plataforma, incluindo cursos, unidades curriculares, turmas e docentes associados. Esta informação é essencial para garantir que cada pedido de compensação é associado corretamente aos dados académicos correspondentes.

A Figura 42 apresenta a interface de gestão de cursos, unidades curriculares e turmas na plataforma Compensa+.

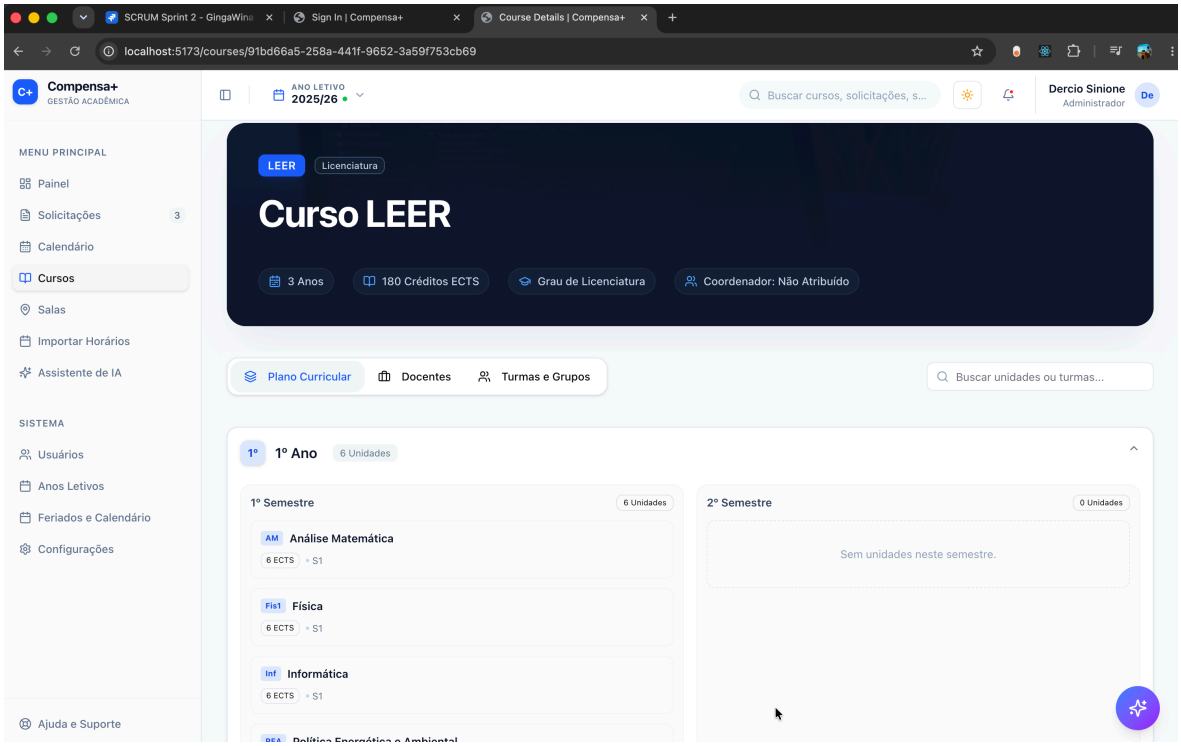


Figura 42 - Gestão de cursos, unidades curriculares e turmas. Fonte: elaboração própria.

Como se observa na Figura 42, a plataforma permite consultar e organizar os elementos académicos necessários ao funcionamento dos pedidos de compensação. Esta estrutura garante maior coerência entre os pedidos submetidos e a organização real dos cursos existentes.

A gestão de salas permite consultar e organizar os espaços físicos disponíveis para marcação de aulas de compensação. Cada sala pode conter informação como identificação, localização, estado e disponibilidade.

A Figura 43 apresenta a interface de gestão de salas de aula da plataforma Compensa+.

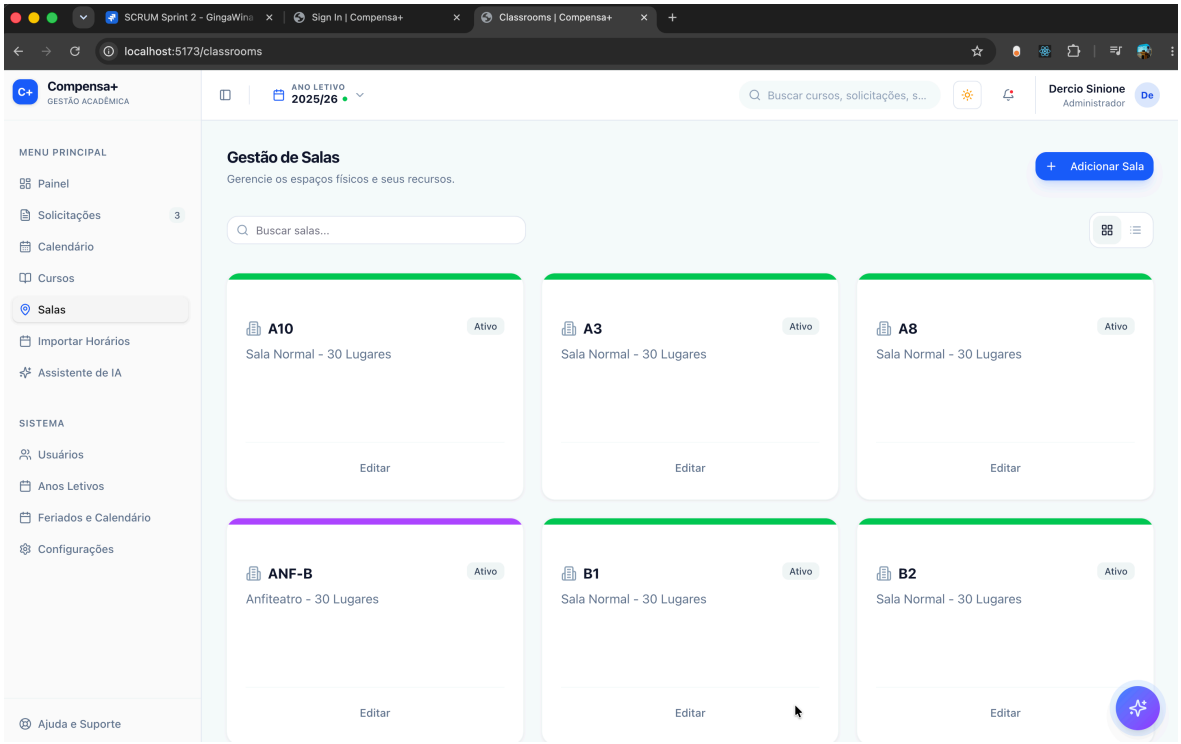


Figura 43 - Gestão de salas de aula. Fonte: elaboração própria.

Como ilustrado na Figura 43, a gestão de salas permite verificar os espaços disponíveis e apoiar a seleção adequada da sala no momento da criação ou validação de um pedido de compensação.

3.4.6. Importação de Turmas e Horários

O módulo de importação de turmas e horários é uma das funcionalidades mais relevantes da plataforma Compensa+, pois permite carregar ficheiros *HTML* exportados do sistema de horários utilizado pela Escola Superior de Tecnologia.

Esta funcionalidade foi implementada para reduzir o trabalho manual associado à criação de cursos, turmas, unidades curriculares, salas e horários. Através deste módulo, a plataforma consegue extrair informação dos ficheiros *HTML*, apresentar uma pré-visualização ao utilizador e, após confirmação, criar ou atualizar automaticamente os dados académicos necessários.

A Figura 44 apresenta a página inicial de importação de horários, onde o utilizador pode selecionar ou arrastar ficheiros *HTML* para processamento.

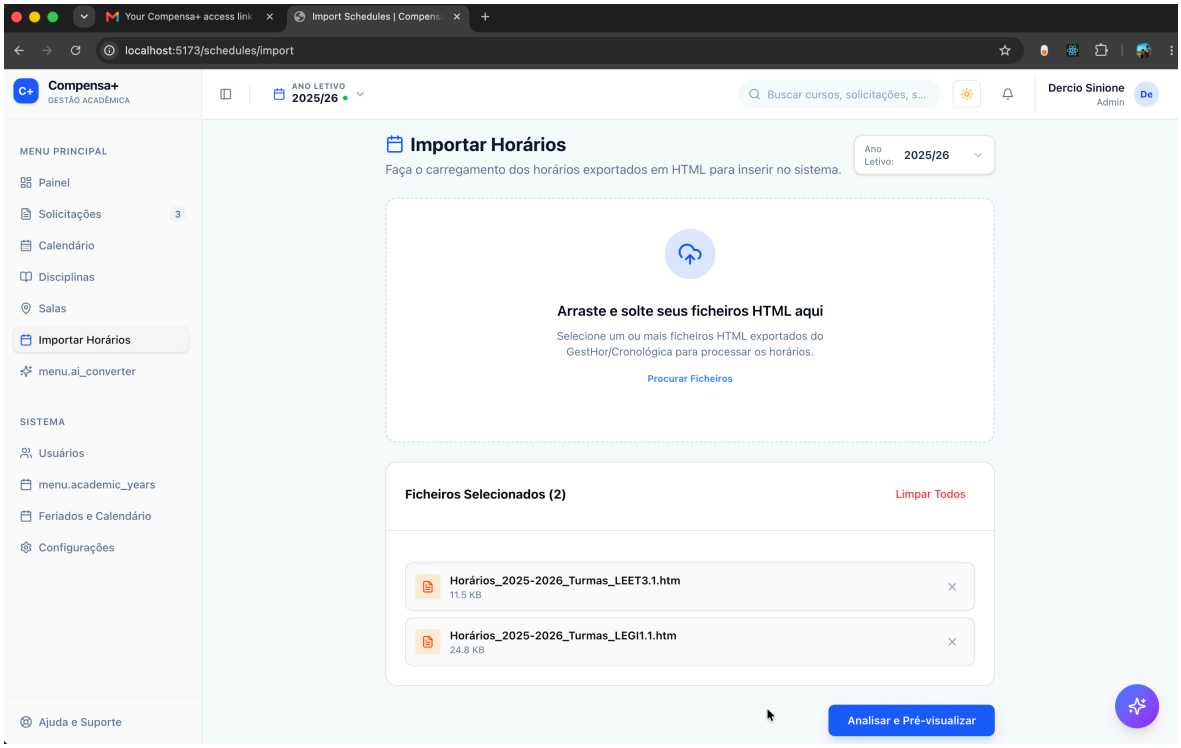


Figura 44 - Página de importação de turmas e horários. Fonte: elaboração própria.

Como apresentado na Figura 44, o utilizador pode carregar um ou mais ficheiros *HTML* e visualizar os documentos selecionados antes de iniciar o processamento. Esta etapa permite confirmar se os ficheiros corretos foram adicionados ao sistema. Após o carregamento dos ficheiros, a plataforma apresenta uma pré-visualização dos dados extraídos. A Figura 45 mostra esta fase de validação intermédia.

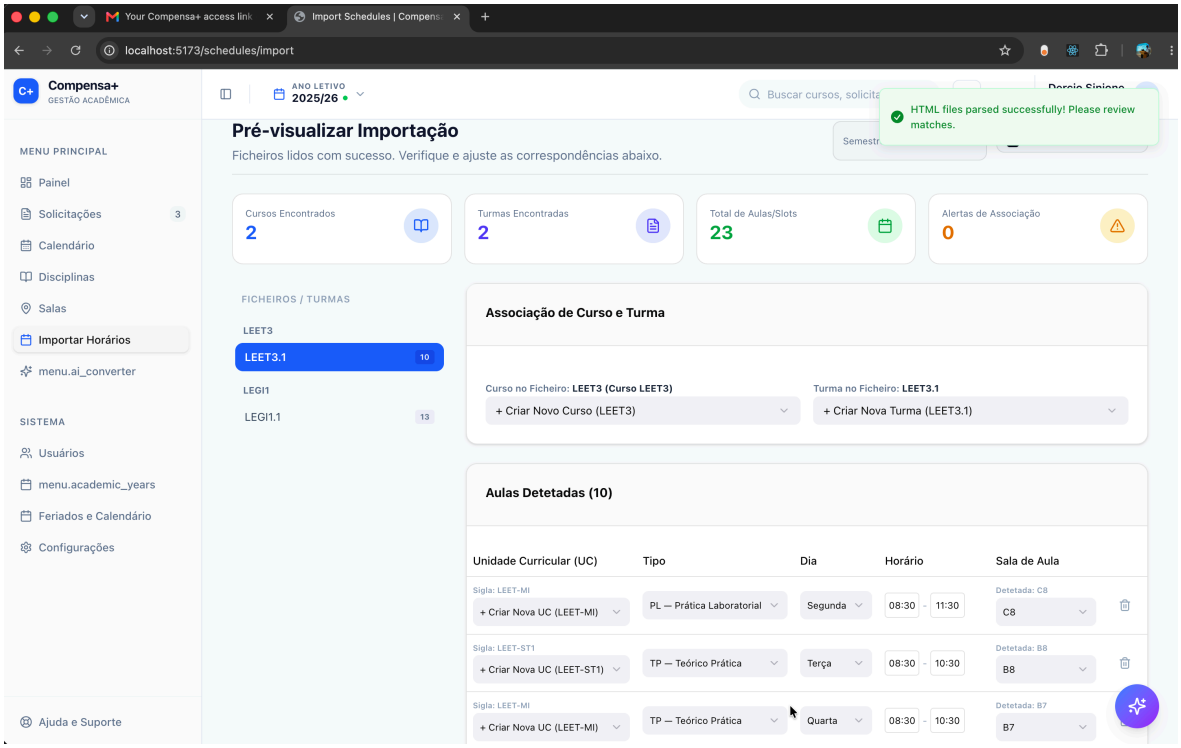


Figura 45 - Pré-visualização da importação de turmas e horários. Fonte: elaboração própria.

Como se observa na Figura 45, a pré-visualização apresenta indicadores como cursos encontrados, turmas identificadas, total de aulas ou slots detetados e alertas de associação. Esta informação permite validar os dados antes da gravação definitiva.

A Figura 46 apresenta a fase de confirmação e mapeamento dos dados importados.

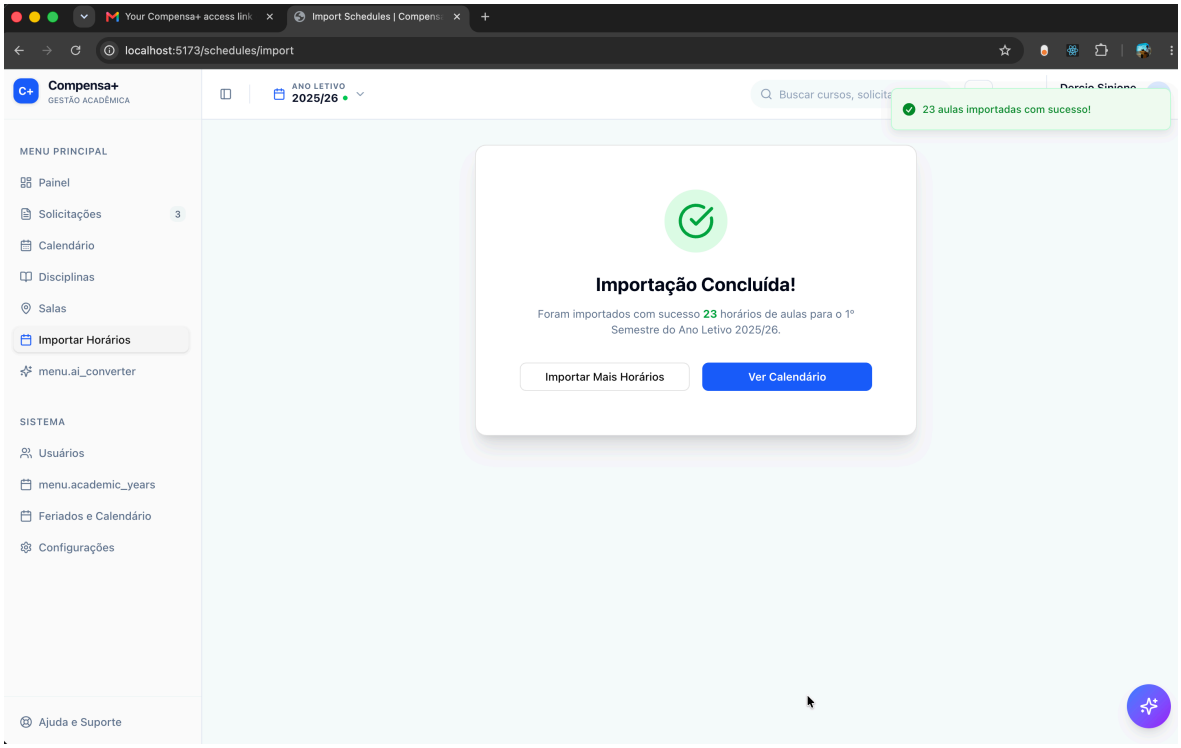


Figura 46 - Confirmação e mapeamento dos dados importados. Fonte: elaboração própria.

Como ilustrado na Figura 46, o sistema permite associar os dados extraídos aos registos existentes na plataforma ou criar automaticamente cursos, turmas, unidades curriculares e salas. Esta funcionalidade reduz o esforço manual de configuração e aproxima a plataforma do funcionamento real da instituição.

3.4.7. Notificações

O módulo de notificações apresenta ao utilizador alertas relacionados com eventos relevantes da plataforma, como criação de pedidos, aprovação, rejeição ou alteração de estado.

No *frontend*, a interface permite consultar notificações, distinguir notificações lidas e não lidas e marcar notificações como lidas. Do ponto de vista técnico, esta página consome dados disponibilizados pela *Notifications API*, que processa eventos provenientes dos restantes microserviços.

A Figura 47 apresenta a página de notificações da plataforma Compensa+.

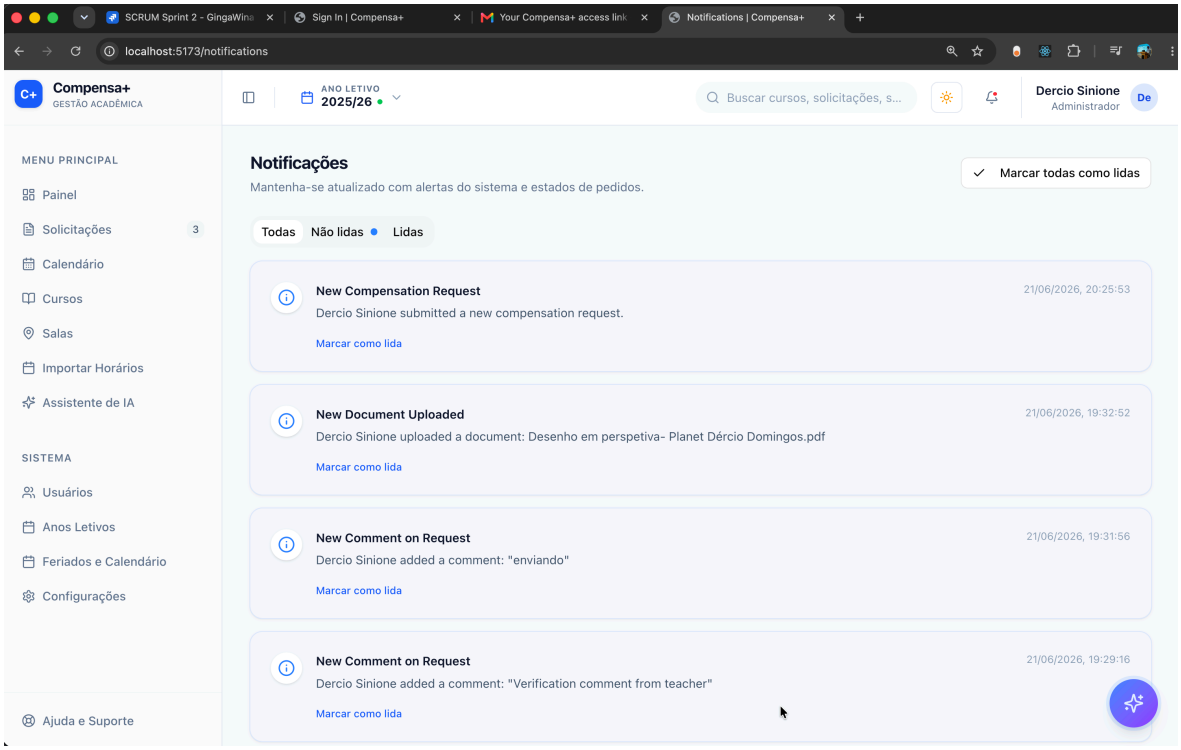


Figura 47 - Página de notificações da plataforma Compensa+. Fonte: elaboração própria.

Como se observa na Figura 47, o utilizador pode consultar notificações associadas às ações realizadas no sistema. Esta interface permite acompanhar alterações importantes diretamente na plataforma, melhorando a comunicação entre docentes, coordenadores e administradores.

3.4.8. Assistente Inteligente

A plataforma inclui uma interface de assistente inteligente integrada com a CompensaAI *API*. Esta funcionalidade permite ao utilizador interagir com um chat capaz de apoiar dúvidas relacionadas com pedidos, horários, regras, disponibilidade de salas, pedidos existentes e dados académicos.

A Figura 48 apresenta a interface do assistente inteligente da plataforma Compensa+.

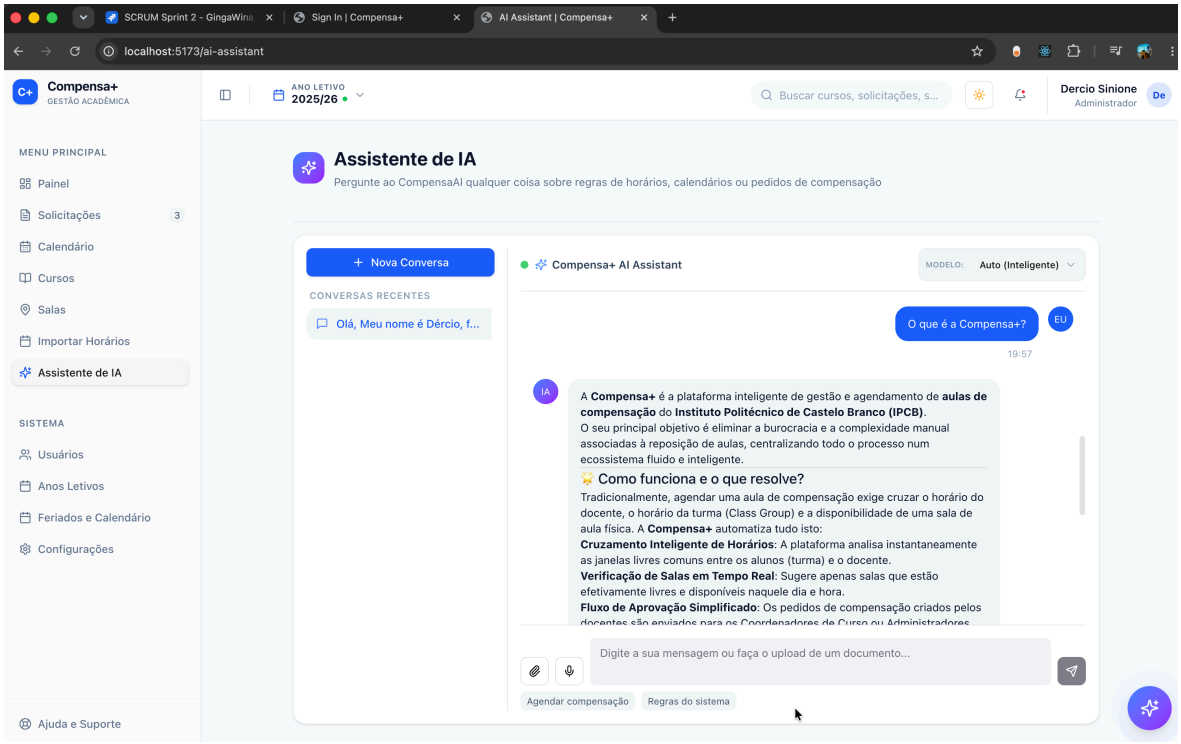


Figura 48 - Assistente inteligente da plataforma Compensa+. Fonte: elaboração própria.

Como se observa na Figura 48, o assistente permite ao utilizador interagir com a plataforma através de uma interface de chat. Apesar deste apoio, as regras de negócio e permissões continuam a ser validadas pela *Core API*, garantindo que a IA atua apenas como camada auxiliar.

3.5. Implementação do Backend

A implementação do *backend* foi realizada com base numa arquitetura de microserviços. No entanto, enquanto o Capítulo 2 apresenta os microserviços do ponto de vista arquitetural, esta secção descreve a organização interna do código, as camadas utilizadas, as entidades de domínio, os serviços, os repositórios e os principais fluxos implementados.

Os principais microserviços *backend* são:

- *API Gateway;*
- *Identity API;*
- *Core API;*
- *Notifications API;*
- *CompensaAI API;*

Os microserviços desenvolvidos em *.NET* foram organizados com separação por camadas e inspiração em *Domain-Driven* Design. Esta organização permitiu isolar

a lógica de negócio dos detalhes técnicos de infraestrutura, como base de dados, cache, mensageria e integrações externas.

De forma geral, os microserviços foram estruturados em quatro camadas principais:

- API: exposição dos *endpoints* HTTP, *controllers* e receção dos pedidos;
- *Application*: implementação dos casos de uso, serviços aplicacionais e validações;
- *Domain*: definição das entidades principais e regras centrais do negócio;
- *Infrastructure*: persistência, repositórios, cache, mensageria e integrações externas.

A Tabela 7 apresenta a organização em camadas adotada nos microserviços backend.

Tabela 7 - Organização em camadas dos microserviços backend. Fonte: elaboração própria.

Camada	Responsabilidade
API	Exposição de endpoints HTTP, controllers e comunicação com o frontend
Application	Implementação dos casos de uso, serviços, validações e regras aplicacionais
Domain	Definição das entidades, objetos de domínio e regras centrais do negócio
Infrastructure	Repositórios, persistência, cache, mensageria e integrações externas

A Figura 49 apresenta a estrutura em camadas dos microserviços *backend* da plataforma Compensa+.



Figura 49 - Estrutura em camadas dos microserviços backend. Fonte: elaboração própria.

Como apresentado na Figura 49, a separação em camadas permite distinguir os *controllers*, os serviços, as entidades de domínio e os mecanismos de infraestrutura. Esta organização torna o código mais manutenível e facilita a evolução dos microserviços.

3.5.1. API *Gateway*

O *API Gateway* foi implementado com YARP *Reverse Proxy* em *ASP.NET Core*. A sua principal função é disponibilizar um ponto único de entrada para o *frontend* e encaminhar os pedidos para os microserviços internos.

A estrutura do Gateway inclui ficheiros de configuração, *middleware* de *logging*, definições de ambiente e configuração de rotas. O Gateway centraliza o roteamento para a *Identity API, Core API, Notifications API* e CompensaAI API, reduzindo o acoplamento entre o *frontend* e os serviços internos.

As principais responsabilidades implementadas no *Gateway* incluem:

- encaminhamento de pedidos para os microserviços;
- configuração de CORS para permitir chamadas do *frontend*;
- *health check em /health;*
- *logging* estruturado;
- rate *limiting*;
- exportação de telemetria através de *OpenTelemetry*.

A Figura 50 apresenta a configuração de rotas do API Gateway.

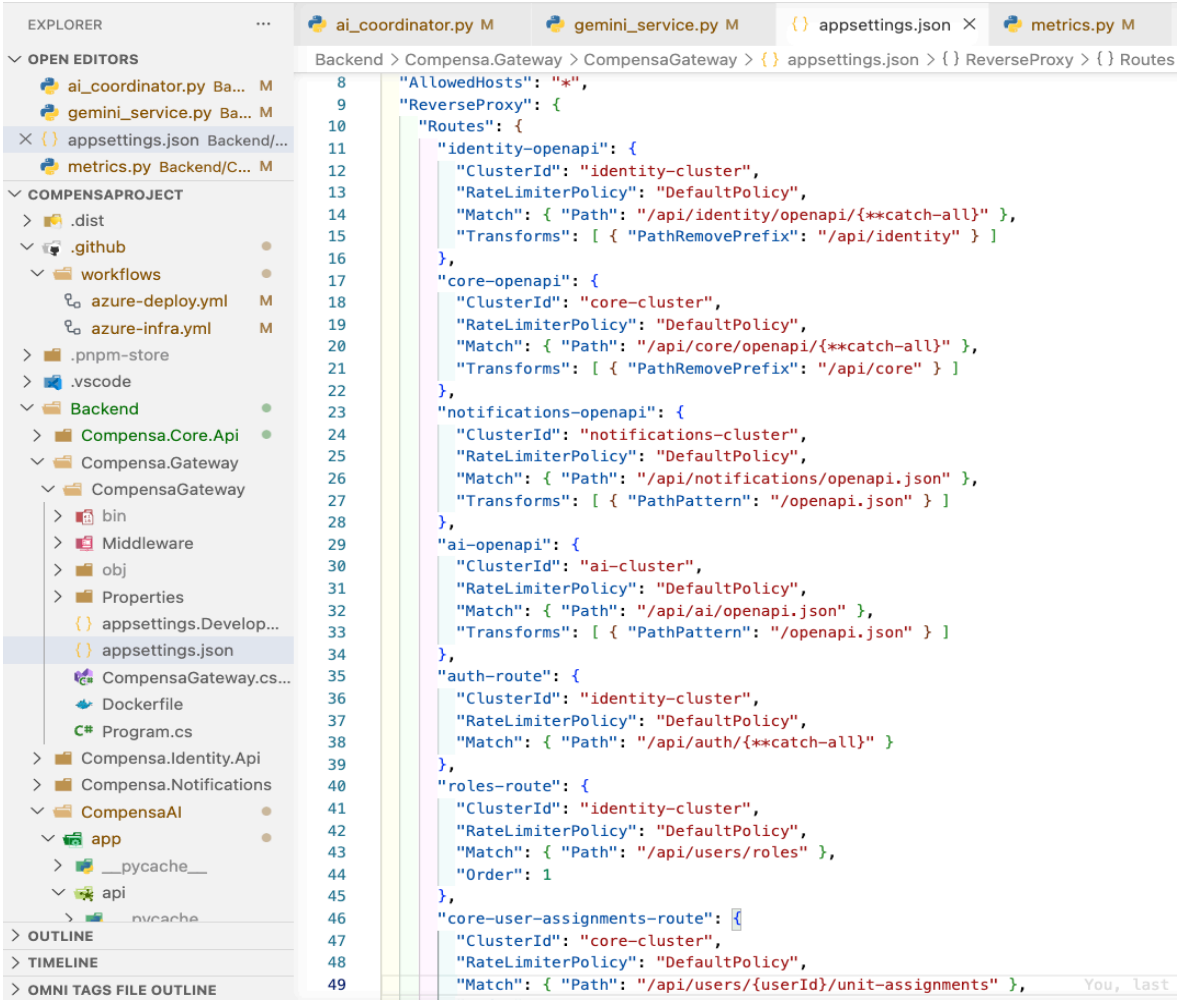


Figura 50 - Configuração de rotas no API Gateway. Fonte: elaboração própria.

Como se observa na Figura 50, o Gateway encaminha as requisições recebidas para os microserviços internos, permitindo que o *frontend* comunique preferencialmente através da rota /api, sem conhecer diretamente os endereços internos dos serviços.

3.5.2. *Identity API*

A *Identity API* é responsável pela autenticação, autorização e gestão dos utilizadores. Este Microserviço implementa autenticação por link mágico, geração de *tokens* JWT, *refresh tokens*, gestão de roles e envio de email para autenticação.

A estrutura da *Identity API* foi organizada em *controllers*, contratos, DTOs, infraestrutura, repositórios, serviços, modelos, *middleware* e observabilidade. Esta organização permite separar a exposição dos *endpoints*, a lógica de autenticação, o acesso aos dados e os mecanismos auxiliares, como email e geração de *tokens*.

O fluxo de autenticação implementado segue os seguintes passos:

1. O utilizador introduz o email na interface;
2. A *Identity* API gera um *token* temporário;
3. O sistema envia um link mágico para o email do utilizador;
4. O utilizador acede ao link recebido;
5. O *token* é validado;
6. A API emite um *access token* JWT e, quando aplicável, um *refresh token*;
7. O *frontend* utiliza o *token* para aceder a funcionalidades protegidas.

A Figura 51 apresenta o fluxo de autenticação por link mágico implementado na plataforma.

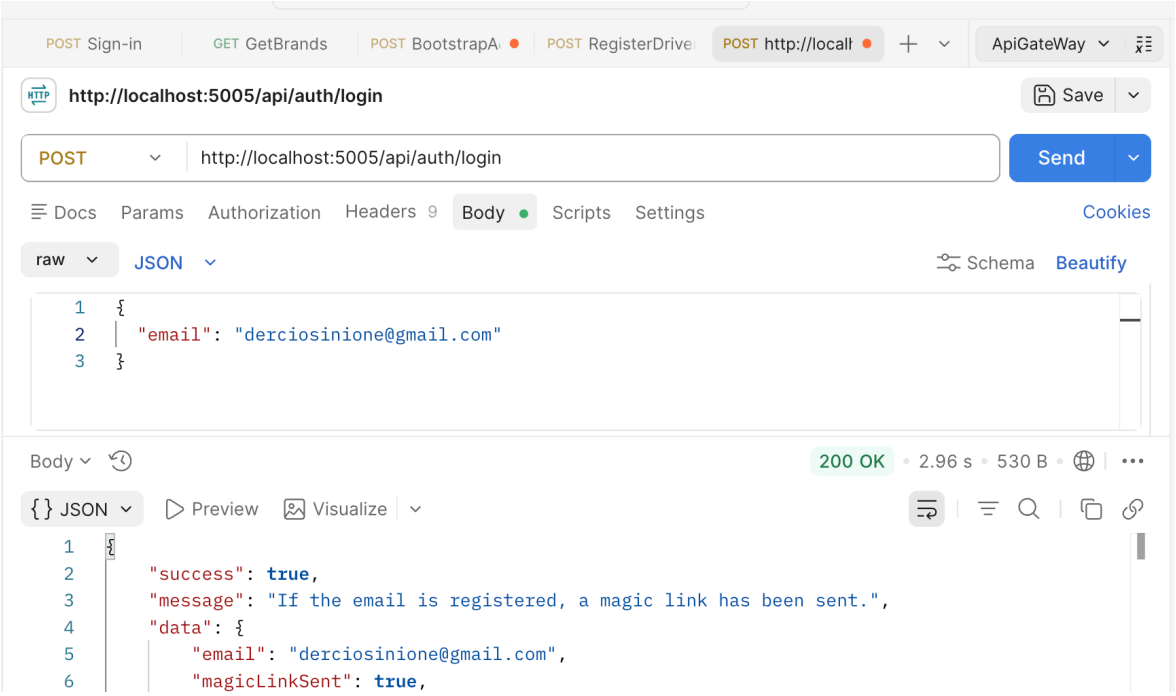


Figura 51 - Fluxo de autenticação por link mágico. Fonte: elaboração própria.

Como demonstrado na Figura 51, o processo de autenticação dispensa *password* tradicional e baseia-se na validação de um *link* temporário enviado ao email do utilizador. Esta abordagem adapta-se ao contexto académico e reduz a necessidade de gerir credenciais tradicionais.

3.5.3. Arquitetura do Microserviço Core API

A Core API é o Microserviço principal da plataforma, responsável pelo domínio de negócio da Compensa+. É neste serviço que se encontram implementadas as funcionalidades associadas a cursos, anos letivos, salas, turmas, horários, pedidos de compensação, importação de horários, pesquisa global, painel principal, auditoria e publicação de eventos.

A estrutura principal da Core API está organizada em *controllers*, contratos, domínio, DTOs, exceções, infraestrutura, eventos de integração, *middleware*, migrações, observabilidade, repositórios e serviços.

Backend/Compensa.Core.Api/CompensaCoreApi/
Controllers/
AcademicYearsController.cs
ClassroomsController.cs
CompensationRequestsController.cs
CoursesController.cs
DashboardController.cs
SchedulesController.cs
SearchController.cs
UserUnitAssignmentsController.cs
Domain/
AcademicYears/
Assignments/
Audit/
Classrooms/
CompensationRequests/
Courses/
Dtos/
Infrastructure/
IntegrationEvents/
Observability/
Repositories/
Services/

A Figura 52 apresenta a estrutura interna da Core API.

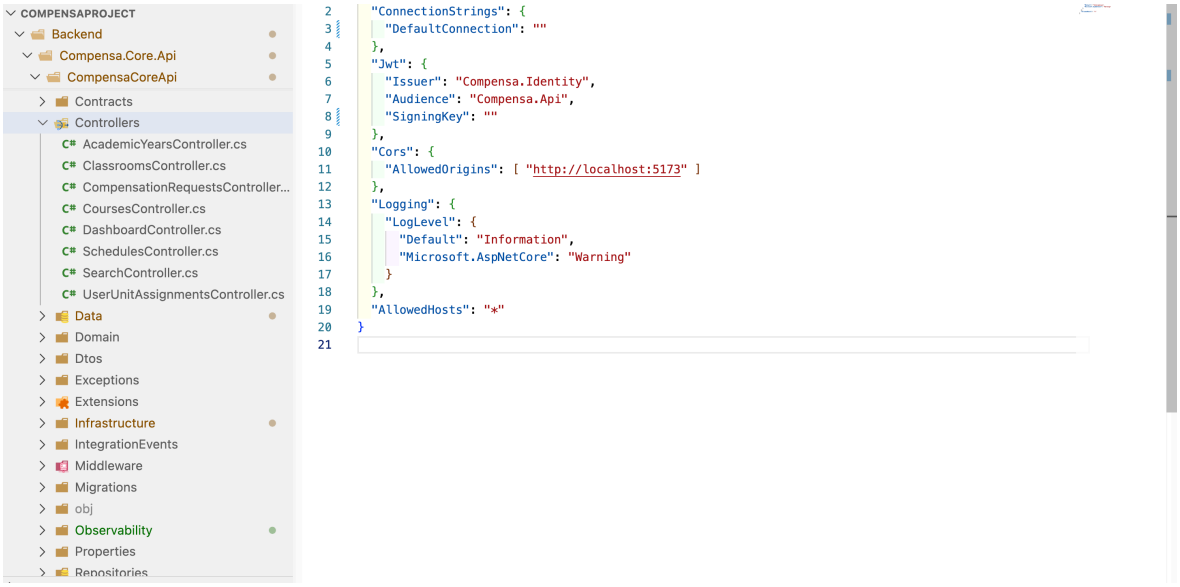


Figura 52 - Estrutura interna da Core API. Fonte: elaboração própria.

Como se observa na Figura 52, a Core API está organizada em *controllers*, domínio, *DTOs*, infraestrutura, eventos de integração, observabilidade, repositórios e serviços. Esta estrutura evidencia a aplicação de uma organização orientada ao domínio, alinhada com as funcionalidades centrais da plataforma.

A *Core API* foi organizada com inspiração em *Domain-Driven Design*, colocando o domínio académico no centro da implementação. Em vez de estruturar o sistema apenas em função das tabelas da base de dados, a lógica foi orientada por conceitos como pedido de compensação, curso, turma, sala, horário e ano letivo.

Entre as principais entidades de domínio destacam-se:

- *CompensationRequest;*
- *CompensationRequestComment;*
- *CompensationRequestDocument;*
- *CompensationRequestStatus;*
- *Course;*
- *CourseOffering;*
- *CurricularUnit;*
- *CurricularUnitOffering;*
- *CurricularUnitComponent;*
- *ClassGroup;*
- *ClassSchedule;*
- *Classroom;*
- *AcademicYear;*
- *AuditLog.*

A Figura 53 apresenta as principais entidades de domínio implementadas na Core API.

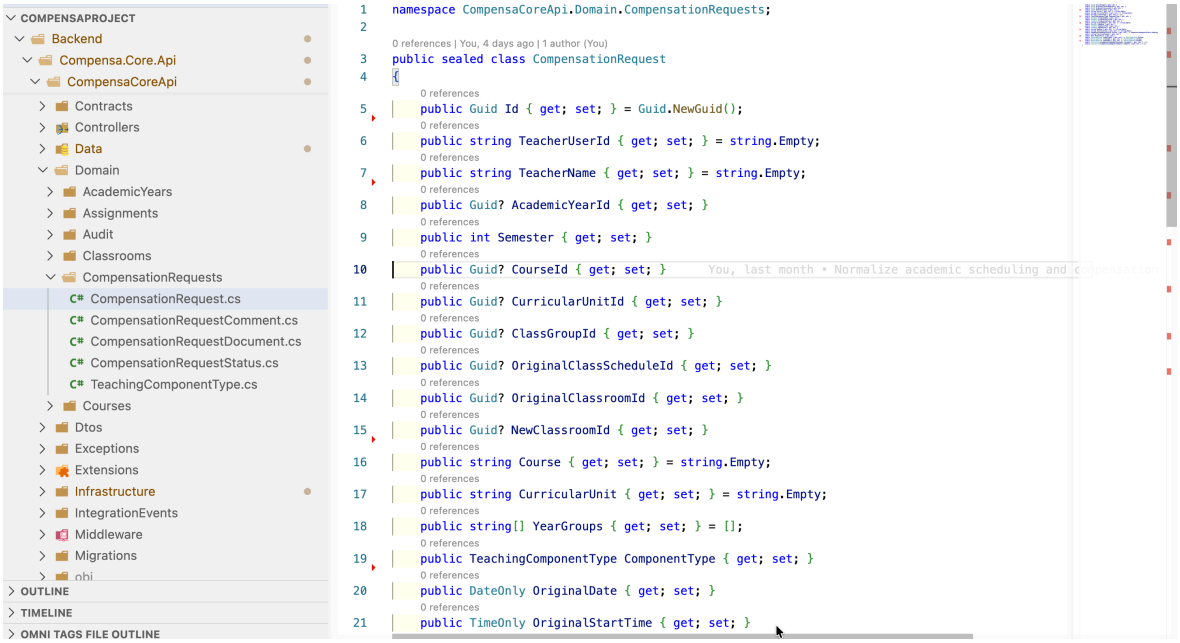


Figura 53 - Entidades de domínio implementadas na Core API. Fonte: elaboração própria.

Como se verifica na Figura 53, a entidade *CompensationRequest* assume um papel central, pois representa o pedido de compensação de aula. Esta entidade relaciona-se com outros conceitos do domínio, como curso, unidade curricular, turma, sala e horário.

Os estados principais de um pedido incluem:

- *Pending*: pedido submetido e ainda não analisado;
- *Approved*: pedido aprovado pelo coordenador;
- *Rejected*: pedido recusado;
- *Cancelled*: pedido cancelado por utilizador autorizado.

A transição entre estados foi tratada de forma controlada. Assim, ações como aprovação ou rejeição são executadas apenas por utilizadores autorizados e respeitando as regras definidas para o processo de compensação.

3.5.4. Serviços da *Core API*

A camada de serviços concentra a lógica aplicacional da *Core API*. Os *controllers* recebem os pedidos *HTTP* e delegam a execução para os serviços, evitando concentrar regras de negócio diretamente nos *endpoints*.

Entre os principais serviços implementados destacam-se:

- *CompensationRequestService*: responsável pela criação, listagem, atualização, aprovação e rejeição de pedidos;
  ScheduleAvailabilityService: responsável por verificar disponibilidade de horários;
- *ScheduleConflictRule*: responsável por validar conflitos entre sala, turma, docente e horários existentes;
- *ScheduleImportService*: responsável pelo processamento dos ficheiros *HTML* de horários;
- *CourseService*: responsável pela gestão de cursos e unidades curriculares;
- *ClassroomService*: responsável pela gestão de salas;
- *DashboardService*: responsável pela agregação de indicadores do painel principal;
- *SearchService*: responsável pela pesquisa global;
- *AuditService*: responsável pelo registo de alterações relevantes.

A Figura 54 apresenta a organização dos serviços aplicacionais da Core API.

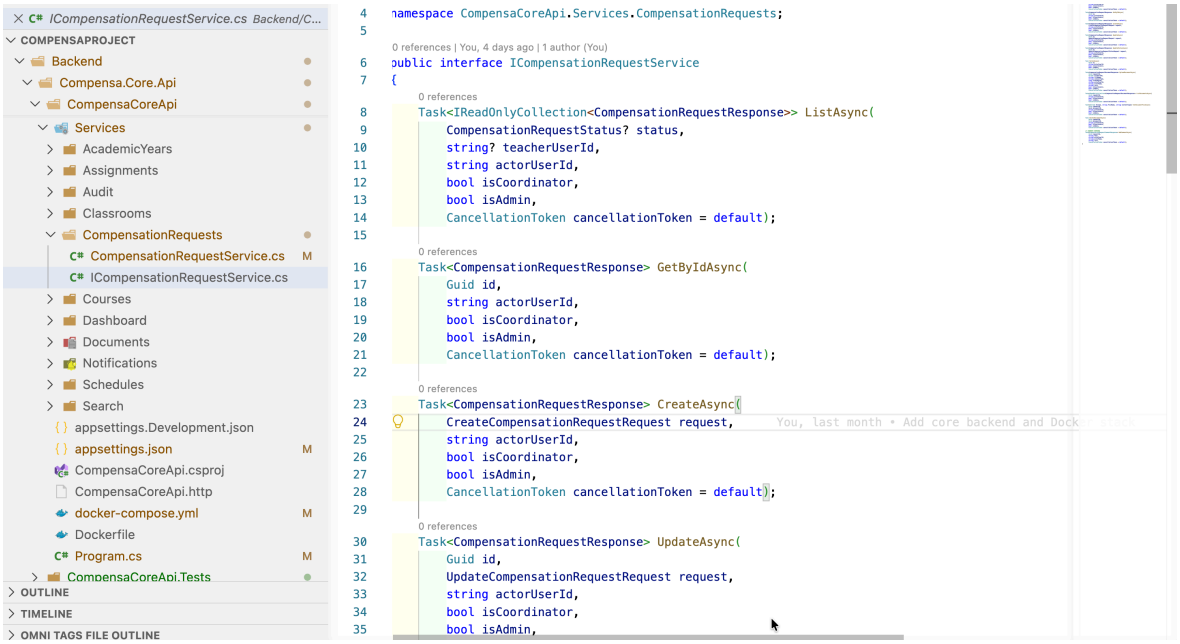


Figura 54 - Organização dos serviços aplicacionais da Core API. Fonte: elaboração própria.

Como apresentado na Figura 54, serviços como *CompensationRequestService, ScheduleAvailabilityService, ScheduleImportService, CourseService, ClassroomService, DashboardService e AuditService* concentram regras e operações essenciais da plataforma. Esta separação evita que a lógica de negócio fique diretamente nos controllers.

Por exemplo, quando um docente submete um pedido de compensação, o *controller* apenas recebe a solicitação e encaminha-a para o *CompensationRequestService*. Este serviço valida os dados, verifica permissões,

consulta a disponibilidade necessária, persiste o pedido e publica um evento para que a *Notifications API* gere a notificação correspondente.

3.5.5. Repositórios e Persistência

A camada de repositórios foi implementada para isolar o acesso à base de dados. Em vez de os serviços comunicarem diretamente com o *Entity Framework Core* ou com *queries* específicas, utilizam repositórios responsáveis por consultar e persistir os dados.

Entre os principais repositórios implementados destacam-se:

- *AcademicYearRepository;*
- *CourseRepository;*
- *CurricularUnitRepository;*
- *ClassGroupRepository;*
- *ClassroomRepository;*
- *ScheduleRepository;*
- *CompensationRequestRepository;*
- *NotificationRepository;*
- *UserUnitAssignmentRepository.*

A Figura 55 apresenta a organização dos repositórios na Core API.

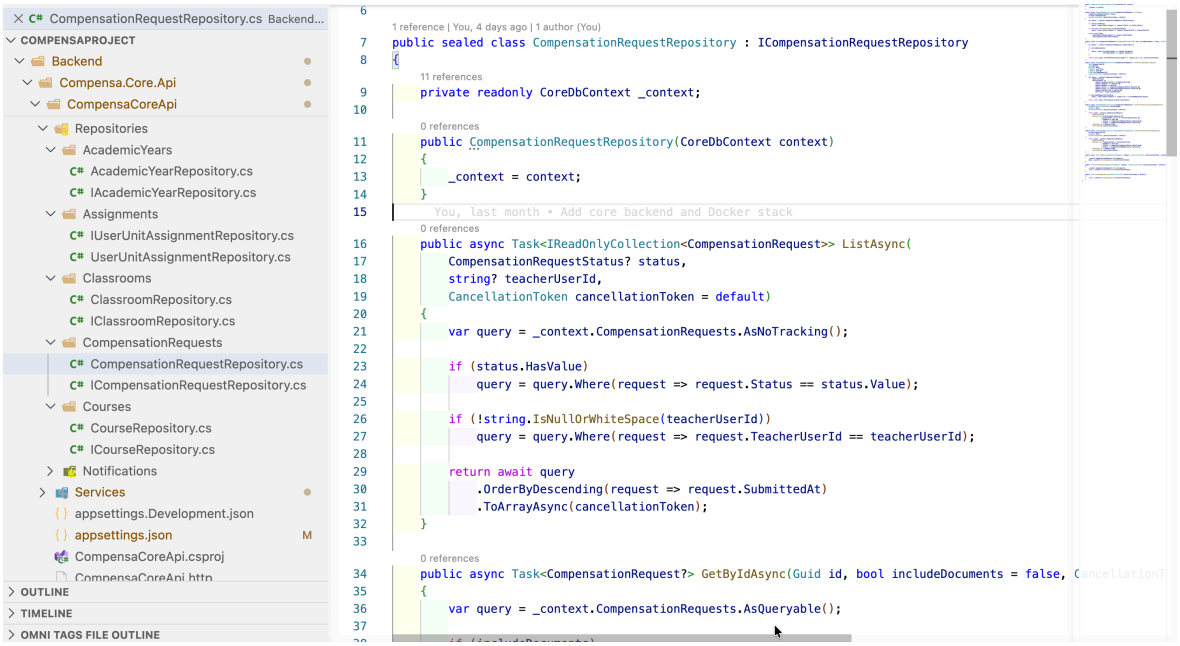


Figura 55 - Organização dos repositórios na Core API. Fonte: elaboração própria.

Como se observa na Figura 55, os repositórios estão organizados de acordo com as entidades do domínio, como cursos, salas, horários, pedidos de compensação e anos letivos. Esta organização contribui para maior manutenibilidade e separação de responsabilidades.

O *CompensationRequestRepository*, por exemplo, permite criar pedidos, consultar pedidos por utilizador, filtrar pedidos por estado, obter detalhes de um pedido e atualizar a decisão tomada pelo coordenador.

3.5.6. Regras de Conflitos e Disponibilidade

Uma parte crítica da *Core API* corresponde à validação de conflitos de horários. Esta validação é essencial para garantir que uma aula de compensação não é marcada em sobreposição com outra aula, sala ocupada, turma indisponível ou docente indisponível.

A lógica de validação foi concentrada em serviços específicos, como *ScheduleAvailabilityService* e *ScheduleConflictRule*. Estes serviços verificam aspetos como:

- disponibilidade da sala;
- disponibilidade da turma;
- disponibilidade do docente;
- sobreposição com horários já existentes;
- validade das datas propostas;
- permissões do utilizador que executa a operação.

A Figura 56 apresenta a lógica ou estrutura responsável pela validação de conflitos de horário na Core API.

```
18   namespace CompensaCoreApi.Services.CompensationRequests;
20   public sealed class CompensationRequestService : ICompensationRequestService
95       public async Task<CompensationRequestResponse> CreateAsync(
101      {
102          ValidateSchedule(request.NewStartTime, request.NewEndTime, "new");
103          var canCreateForOthers = isCoordinator || isAdmin;
104          EnsureCanCreateRequestForUser(request, actorUserId, canCreateForOthers);
105
106          var academicYear = await _academicYearRepository.GetByIdAsync(request.AcademicYearId, cancellationToken)
107              ?? throw new NotFoundException($"Academic year '{request.AcademicYearId}' was not found.");
108
109          var course = await _courseRepository.GetByIdAsync(request.CourseId, cancellationToken)
110              ?? throw new NotFoundException($"Course '{request.CourseId}' was not found.");
111
112          var unit = await _courseRepository.GetUnitByIdAsync(request.CourseId, request.CurricularUnitId, cancellationToken)
113              ?? throw new NotFoundException($"Curricular unit '{request.CurricularUnitId}' was not found in course '{request.CourseId}'.");
114
115          var classGroup = await _courseRepository.GetClassGroupByIdAsync(request.CourseId, request.ClassGroupId, cancellationToken)
116              ?? throw new NotFoundException($"Class group '{request.ClassGroupId}' was not found in course '{request.CourseId}'.");
117
118          var originalSchedule = await _courseRepository.GetClassScheduleByIdAsync(request.OriginalClassScheduleId, cancellationToken)
119              ?? throw new NotFoundException($"Original class schedule '{request.OriginalClassScheduleId}' was not found.");
120
121          var originalRoom = originalSchedule.ClassroomId.HasValue
122              ? await _classroomRepository.GetByIdAsync(originalSchedule.ClassroomId.Value, cancellationToken)      You, 6 days ago • feat: upda
123              : null;
124
125          var newRoom = await _classroomRepository.GetByIdAsync(request.NewClassroomId, cancellationToken)
126              ?? throw new NotFoundException($"New classroom '{request.NewClassroomId}' was not found.");
127
128          await EnsureCanCreateRequestInCourseAsync(request, course, actorUserId, isCoordinator, isAdmin, cancellationToken);
129          ValidateOriginalSchedule(request, academicYear.Id, course.Id, unit.Id, classGroup.Id, originalSchedule);
130          await EnsureTeacherCanUseClassAsync(request, classGroup, canCreateForOthers, cancellationToken);
131          await EnsureNewScheduleHasNoConflictsAsync(request, classGroup, originalSchedule, cancellationToken);
132
133          var now = DateTimeOffset.UtcNow;
134          var compensationRequest = new CompensationRequest
```

Figura 56 - Validação de conflitos de horário na Core API. *Fonte: elaboração própria.*

Como demonstrado na Figura 56, a validação é realizada no *backend*, garantindo que as regras de negócio sejam aplicadas mesmo quando uma requisição é enviada diretamente para a *API*. Esta decisão aumenta a segurança e a consistência da plataforma.

3.5.7. Importação de Turmas e Horários no *Backend*

O módulo de importação de turmas e horários exigiu uma implementação específica na Core API, uma vez que envolve leitura de ficheiros HTML, extração de dados académicos, normalização da informação, mapeamento de entidades e persistência na base de dados. Esta funcionalidade foi desenvolvida para permitir que horários exportados em formato HTML fossem importados para a plataforma Compensa+, reduzindo o trabalho manual associado à criação de cursos, turmas, unidades curriculares, salas e blocos horários.

Do ponto de vista técnico, a importação foi organizada em duas fases principais: pré-visualização e confirmação. Esta separação foi adotada para permitir que os dados extraídos fossem apresentados ao utilizador antes da gravação definitiva, garantindo maior controlo sobre o processo e reduzindo o risco de inserir informação incorreta na base de dados.

Na primeira fase, correspondente à pré-visualização, o *frontend* valida os ficheiros submetidos, aceitando apenas documentos nos formatos *.html* ou *.htm.* A página de importação permite o carregamento dos ficheiros através de seleção manual ou mecanismo de *drag-and-drop*. Após a seleção, o *frontend* constrói um objeto *FormData* com os ficheiros e o identificador do ano letivo selecionado, enviando posteriormente a requisição para o *endpoint* de pré-visualização da *Core API*.

No *backend*, o *endpoint* de importação recebe os ficheiros através de uma coleção *IFormFileCollection*. Cada ficheiro é lido e convertido para um *array* de bytes, sendo depois encaminhado para o serviço responsável pelo processamento dos horários. Este processamento é realizado pelo *TimetableImportService*, que centraliza a lógica de leitura, interpretação, normalização e mapeamento dos dados extraídos dos ficheiros *HTML*.

O processamento dos ficheiros é iniciado no método *ProcessTimetablesAsync*. Este método recebe a lista de ficheiros *HTML* submetidos pelo utilizador, o identificador do ano letivo selecionado e o *token* de cancelamento da operação. Antes de interpretar os horários, o serviço verifica se existe um ano letivo ativo ou selecionado. Caso nenhum ano letivo seja encontrado, é lançada uma exceção de domínio, impedindo que a importação prossiga sem contexto académico válido.

A Figura 57 apresenta o início do método responsável pelo processamento dos ficheiros *HTML* no *backend*.

```
17  namespace CompensaCoreApi.Services.Schedules;
19  public sealed class TimetableImportService : ITimetableImportService
30      public async Task<TimetableImportPreviewResponse> ProcessTimetablesAsync(
31          IEnumerable<(string Filename, byte[] Content)> files,
32          Guid? academicYearId,
33          CancellationToken cancellationToken = default)
34      {
35          // 1. Get Academic Year
36          var activeYear = academicYearId.HasValue
37              ? await _context.AcademicYears.FindAsync([academicYearId.Value], cancellationToken)
38              : await _context.AcademicYears.FirstOrDefaultAsync(y => y.IsActive, cancellationToken);
39
40          if (activeYear == null)
41              throw new InvalidOperationException("No active academic year found.");
42
43          // 2. Load DB entities for matching
44          var dbCourses = await _context.Courses.ToListAsync(cancellationToken);
45          var dbClassGroups = await _context.ClassGroups
46              .Where(cg => cg.AcademicYearId == activeYear.Id)
47              .ToListAsync(cancellationToken);
48          var dbUnits = await _context.CurricularUnits.ToListAsync(cancellationToken);
49          var dbClassrooms = await _context.Classrooms.ToListAsync(cancellationToken);
50
51          var dbClassroomNames = dbClassrooms.Select(c => c.Name.ToUpperInvariant()).ToHashSet();
52
53          // 3. Prepare response
54          var response = new TimetableImportPreviewResponse
55          {
56              AcademicYearId = activeYear.Id,
57              AcademicYearName = activeYear.Name,
58              DetectedSemester = 1
59          };
60
61          // We will group by Course + Class Group name dynamically
62          var coursesMap = new Dictionary<string, CoursePreviewDto>();
63
64          foreach (var file in files)
```

Figura 57 - Processamento de ficheiros HTML no serviço de importação de turmas e horários. Fonte: elaboração própria.

Como se observa na Figura 57, após a identificação do ano letivo, o serviço carrega da base de dados os cursos, turmas, unidades curriculares e salas já existentes. Estes dados são utilizados posteriormente para realizar o *matching* entre a informação extraída dos ficheiros *HTML* e os registos já persistidos na plataforma. Desta forma, o sistema consegue identificar quais entidades já existem e quais precisam de ser criadas ou associadas durante a confirmação da importação.

Ainda nesta fase, o serviço prepara a resposta de pré-visualização, associando o identificador e o nome do ano letivo ao resultado da importação. Esta resposta será depois preenchida com os cursos, turmas, unidades curriculares, salas e blocos horários detetados nos ficheiros enviados. A utilização desta etapa intermédia permite que o utilizador valide os dados antes da gravação definitiva na base de dados.

Após esta preparação inicial, cada ficheiro *HTML* é processado individualmente. O serviço realiza a descodificação do conteúdo, interpreta a estrutura das tabelas do horário e extrai os dados relevantes, como turma, unidade curricular, tipo de aula, docente, sala, dia da semana, hora de início e hora de fim. Estes dados são posteriormente normalizados e organizados para apresentação na fase de pré-visualização.

A leitura dos ficheiros *HTML* contempla o tratamento de diferentes codificações de texto. O serviço tenta reconhecer formatos como *UTF-16LE, UTF-16BE, UTF-8*

com BOM e *UTF-16* sem BOM. Quando nenhuma destas codificações é detetada, é utilizada por defeito a codificação *ISO-8859-1*, uma vez que muitos ficheiros *HTML* exportados por sistemas antigos utilizam codificações próximas de *Windows-1252.* Esta decisão foi importante para evitar problemas com acentos, caracteres especiais e nomes de unidades curriculares ou docentes.

O *parser HTML* foi implementado sem recurso a uma biblioteca *DOM* externa. A extração da estrutura do documento é realizada através de expressões regulares, identificando inicialmente as tabelas *HTML*, depois as linhas e, por fim, as células. Durante este processo, são analisadas *tags* como *table, tr, td e th*, sendo também interpretado o atributo *rowspan*. Além disso, as quebras de linha representadas por br são convertidas para quebras reais, as restantes tags *HTML* são removidas e o conteúdo é normalizado através de *HTML decode*. Os dados extraídos são guardados em estruturas internas simples, como *ParsedTable e ParsedCell*, que facilitam o processamento posterior da grelha horária.

Após a extração das tabelas, o serviço identifica dois tipos principais de informação: a tabela principal do horário e a tabela de legenda. A tabela principal é reconhecida quando o cabeçalho contém dias da semana, como Segunda, Terça, Quarta, Quinta ou Sexta. A tabela de legenda é identificada através de cabeçalhos como Sigla e Disciplina, permitindo associar as siglas presentes na grelha ao nome completo das unidades curriculares.

A identificação da turma é realizada através de várias estratégias, de forma a aumentar a robustez do processo. O sistema tenta primeiro localizar no corpo do ficheiro uma referência textual no formato “Turma: ...”. Caso esta informação não seja encontrada, procura a mesma indicação na *tag title* do *HTML*. Como alternativa, o serviço tenta ainda inferir a turma a partir do nome do ficheiro, utilizando padrões associados aos ficheiros exportados.

A leitura da grelha do horário utiliza um algoritmo próprio para lidar com células que ocupam várias linhas, através do atributo *rowspan*. Como uma aula pode decorrer durante mais do que um bloco horário, o serviço mantém estruturas auxiliares para controlar as células ativas, os blocos de tempo e as linhas abrangidas por cada aula. Assim, quando uma célula ocupa duas ou mais linhas, o sistema associa corretamente a aula ao respetivo dia e ajusta a hora de fim para corresponder ao último bloco horário abrangido.

Cada célula de aula é interpretada segundo uma estrutura esperada. A primeira linha identifica a sigla da unidade curricular; a segunda linha representa o tipo de componente letiva, como T, TP, PL ou P; e as últimas linhas são analisadas para identificar o docente e a sala. A sigla da unidade curricular é posteriormente cruzada com a tabela de legenda, permitindo obter o nome completo da disciplina. A sala é validada através de padrões reconhecidos, como A., B., LAB, ANF ou

SALA. O tipo de componente é convertido para um valor lógico da aplicação, como *Theoretical, Practical, TheoreticalPractical, PracticalLaboratorial ou All.*

O resultado da fase de pré-visualização inclui os cursos, turmas, unidades curriculares, salas e horários encontrados, indicando quais os elementos que já existem na base de dados e quais necessitam de ser criados ou mapeados. Esta informação é devolvida ao *frontend* para que o utilizador possa confirmar os dados extraídos, corrigir associações e decidir se pretende prosseguir com a gravação.

Na segunda fase, correspondente à confirmação, o *frontend* constrói o *payload* final da importação, incluindo os cursos, turmas, unidades curriculares e salas a criar, os mapeamentos ajustados pelo utilizador e os slots finais de horário. Este pedido é enviado novamente para a *Core API*, que executa a gravação definitiva através do método de confirmação do *TimetableImportService*.

Durante a confirmação, o serviço cria as entidades em falta, evita duplicações através da verificação de identificadores, nomes e siglas, e regista os novos horários como *ClassSchedule*. Caso a opção de substituição de horários existentes esteja ativa, o sistema remove previamente os horários antigos das turmas e semestre envolvidos, garantindo que os novos dados substituem a informação anterior de forma controlada. Após a gravação, são invalidadas as caches relacionadas com cursos e *dashboard*, garantindo que os dados apresentados na interface permanecem atualizados.

Desta forma, o módulo de importação de turmas e horários não se limita ao simples carregamento de ficheiros. Trata-se de um processo completo de leitura, interpretação, validação, mapeamento, pré-visualização e persistência dos dados académicos. Esta implementação permite adaptar a plataforma ao funcionamento real da Escola Superior de Tecnologia, reduzindo o trabalho manual de configuração e aumentando a consistência dos horários utilizados nos pedidos de compensação.

3.5.8. Notifications API

A *Notifications API* foi implementada em *Python* com *FastAPI* e é responsável por persistir notificações, gerir preferências e consumir eventos publicados através do *RabbitMQ*.

A estrutura principal do serviço inclui *endpoints*, configuração, base de dados, integração com *RabbitMQ*, modelos, serviços, métricas e testes.
Backend/Compensa.Notifications/
app/
api/

auth.py
endpoints.py
core/
config.py
database.py
rabbitmq.py
models/
notification.py
services/
consumer.py
email.py
main.py
metrics.py

As principais funcionalidades implementadas são:

- listagem de notificações por utilizador;
- marcação de notificações como lidas;
- gestão de preferências de notificação;
- consumo de eventos *RabbitMQ*;
- envio de email;
- autenticação por JWT;
- *health check;*
- métricas de negócio.

Os endpoints principais incluem:

- *GET /health*
- *GET /api/notifications*
- *PATCH /api/notifications/{notification_id}/read*
- *GET /api/preferences*
- *PUT /api/preferences*

A Figura 58 apresenta os *endpoints* principais da *Notifications API.*

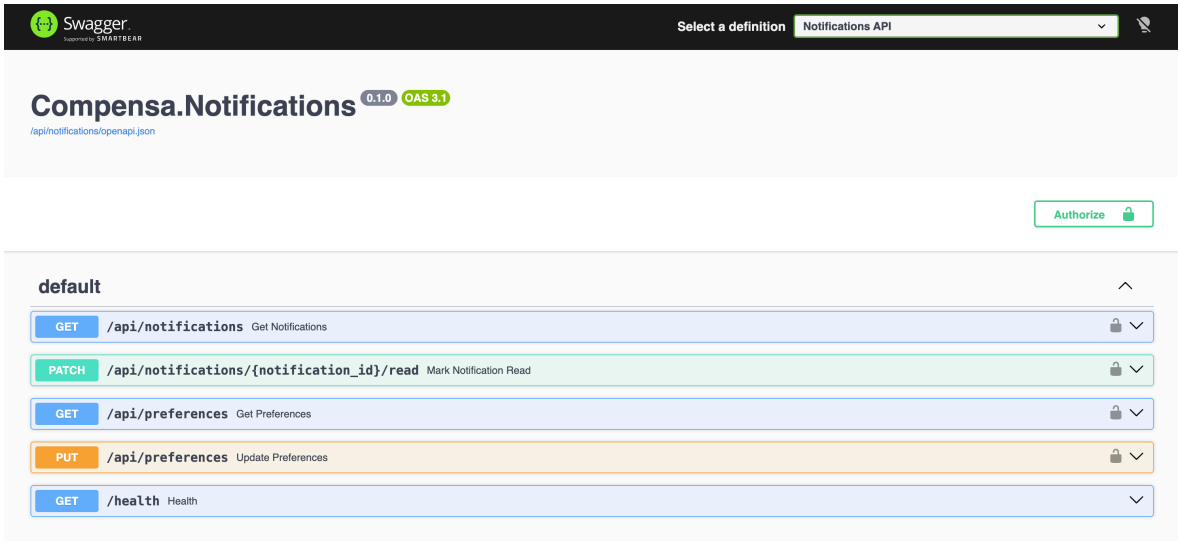


Figura 58 - Endpoints principais da Notifications API. Fonte: elaboração própria.

Como se observa na Figura 58, este microserviço disponibiliza *endpoints* para consulta e atualização das notificações. Este microserviço permite que o envio e registo de notificações sejam tratados de forma desacoplada da *Core API.* Assim, a criação ou atualização de um pedido não fica dependente diretamente do envio de email ou criação da notificação interna.

3.5.9. CompensaAI API

A CompensaAI API foi implementada com *FastAPI* e corresponde ao microserviço responsável pela assistência inteligente da plataforma. Este serviço disponibiliza *endpoints* autenticados para chat, upload de ficheiros e gestão de *threads*.

A estrutura principal do serviço encontra-se organizada da seguinte forma:
Backend/CompensaAI/
app/
api/
auth.py
routes/
chat.py
core/
config.py
services/
ai_coordinator.py
cache_service.py
core_api_service.py
gemini_service.py
openai_service.py
main.py
metrics.py

A Figura 59 apresenta a estrutura interna da CompensaAI API.

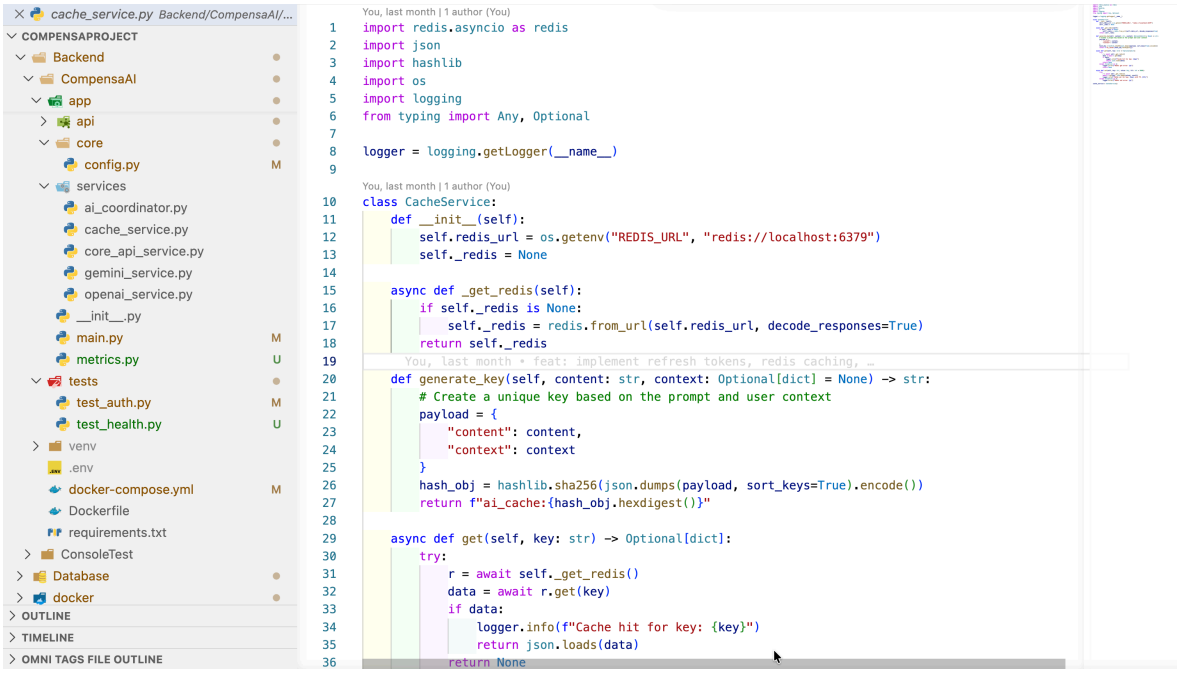


Figura 59 - Estrutura interna da CompensaAI API. Fonte: elaboração própria.

Como se observa na Figura 59, o serviço foi organizado em módulos especializados, como ai_*coordinator*, *openai_service*, *gemini_service*, *core_api_service* e *cache_service*. Esta organização permite alternar entre provedores de IA, consultar dados reais da plataforma e manter as regras de negócio centralizadas na Core API.

A arquitetura interna da IA foi dividida em serviços especializados:

- ai_coordinator.py: decide qual provedor de IA utilizar, gere fallback, threads e cache;
- openai_service.py: integra a plataforma com a API da OpenAI;
- gemini_service.py: integra a plataforma com o Gemini;
- core_api_service.py: encapsula chamadas autenticadas à Core API;
- cache_service.py: guarda respostas ou contexto reutilizável para reduzir chamadas repetidas.

O fluxo simplificado da CompensaAI API é o seguinte:

1. O utilizador envia uma mensagem através do *frontend*;
2. A CompensaAI API valida o JWT;
3. O serviço cria o contexto do utilizador;
4. O *ai_coordinator* verifica cache e escolhe o provedor de IA;
5. O modelo responde diretamente ou chama uma ferramenta;
6. Quando necessário, a ferramenta consulta a *Core API*;

7. A resposta é devolvida ao *frontend*.

A Figura 60 apresenta o fluxo de funcionamento do assistente inteligente.

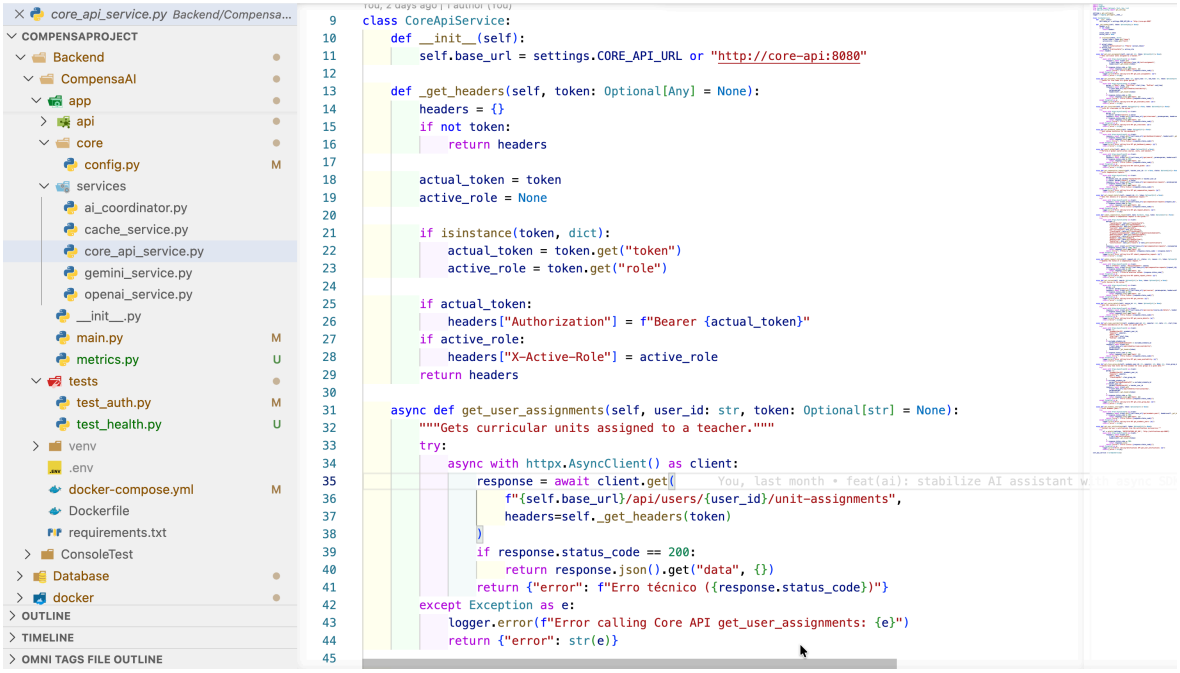


Figura 60 - Fluxo de funcionamento do assistente inteligente. Fonte: elaboração própria.

Como apresentado na Figura 60, quando o assistente necessita de dados reais, utiliza ferramentas que comunicam com a *Core API*. Assim, a IA não acede diretamente à base de dados nem substitui as regras formais da plataforma.

Entre as ferramentas disponibilizadas ao assistente destacam-se:

- consultar notificações do utilizador;
- listar pedidos de compensação;
- consultar detalhes de um pedido;
- criar rascunho de pedido;
- submeter pedido de compensação;
- aprovar ou rejeitar pedido;
- consultar salas disponíveis;
- consultar disponibilidade de turmas;
- consultar cursos;
- consultar anos letivos;
- obter resumo do painel principal;
- executar pesquisa global.

A segurança deste módulo depende de três camadas: JWT obrigatório, contexto do utilizador autenticado e validação final das regras na *Core API*. Assim, mesmo que o assistente tente executar uma ação, a autorização real continua a ser validada pelo *backend*.

3.6. Comunicação Assíncrona, Cache e Observabilidade na Implementação

Embora a arquitetura destes componentes tenha sido apresentada no Capítulo 2, durante a implementação foi necessário integrar concretamente Redis, *RabbitMQ* e *OpenTelemetry* nos serviços.

O *RabbitMQ* foi utilizado para comunicação por eventos. Quando ocorre uma ação relevante, como criação ou alteração de estado de um pedido de compensação, a Core API publica um evento. A *Notifications* API consome esse evento e cria a notificação correspondente. Esta abordagem reduz o acoplamento entre os serviços, pois a Core API não precisa enviar notificações diretamente.

A Figura 61 apresenta o fluxo de publicação e consumo de eventos entre a Core API, *RabbitMQ* e *Notifications* API.

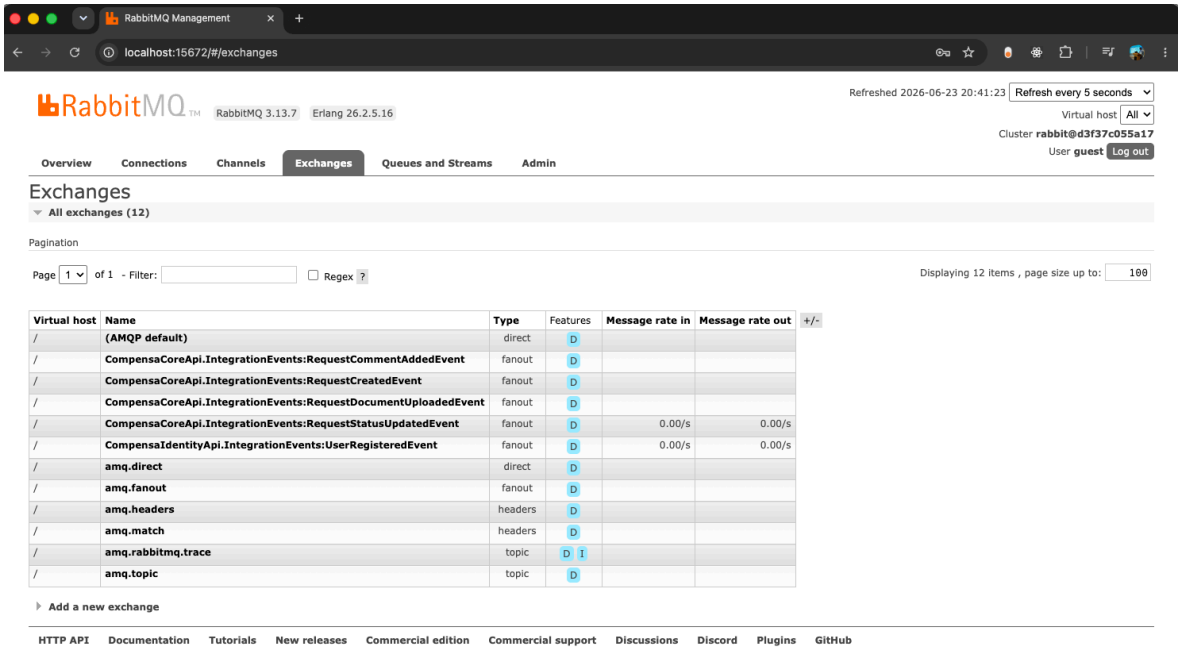


Figura 61 - Publicação e consumo de eventos entre microserviços. Fonte: elaboração própria.

Como ilustrado na Figura 61, a Core API publica eventos como *RequestCreatedEvent* ou *RequestStatusUpdatedEvent*, enquanto a *Notifications* API consome esses eventos e gera as notificações correspondentes. Esta abordagem evita que o envio de notificações bloqueie o fluxo principal da aplicação.

O Redis foi utilizado como mecanismo de cache para operações de leitura frequente. Dados como listas, métricas do painel principal ou resultados de consultas recorrentes podem ser armazenados temporariamente para melhorar o tempo de resposta e reduzir a carga sobre o *PostgreSQL*.

A observabilidade foi implementada através de *OpenTelemetry*, permitindo exportar métricas, logs e traces dos serviços. Cada serviço possui ficheiros próprios de métricas, como *CompensaCoreMetrics*, *CompensaIdentityMetrics* e ficheiros metrics.py nos serviços *Python*. Estes dados são enviados para o *OpenTelemetry Collector* e posteriormente visualizados em ferramentas como *Prometheus*, *Grafana*, *Loki*, Tempo e *Aspire Dashboard*.

3.7. Considerações Finais

A implementação da plataforma Compensa+ permitiu transformar os modelos e protótipos definidos no Projeto I numa aplicação funcional, composta por *frontend web*, microserviços *backend* e serviços de suporte.

No *frontend*, foram implementadas as principais páginas da plataforma, incluindo autenticação, painel principal, pedidos de compensação, detalhes de pedidos, calendário académico, cursos, turmas, salas, utilizadores, notificações, importação de horários e assistente inteligente. Estas páginas comunicam com dados reais através do API Gateway e dos microserviços *backend*.

No *backend*, a implementação foi organizada em microserviços com responsabilidades bem definidas. A Core API concentra as regras principais de negócio e foi estruturada com inspiração em *Domain-Driven Design*, utilizando entidades, serviços, repositórios e regras de validação. A *Identity* API trata a autenticação por link mágico e JWT. A *Notifications* API processa eventos e notificações de forma assíncrona. A *CompensaAI* API acrescenta uma camada de assistência inteligente, mantendo as regras de negócio centralizadas na *Core API.*

O módulo de importação de turmas e horários destacou-se como uma das funcionalidades mais relevantes, por permitir carregar ficheiros *HTML*, extrair dados académicos, criar ou atualizar automaticamente cursos, turmas, unidades curriculares, salas e horários. Esta funcionalidade aproxima a plataforma do funcionamento real da Escola Superior de Tecnologia e reduz significativamente o esforço manual de configuração.

A integração entre *frontend, API Gateway, Identity API, Core API, Notifications API, CompensaAI API, PostgreSQL, Redis, RabbitMQ e Docker* permitiu construir uma solução modular, organizada e preparada para evolução futura. O capítulo seguinte apresenta a fase de testes e validação, onde são descritos os procedimentos utilizados para verificar o correto funcionamento das funcionalidades implementadas.

4. Testes e Validação

O presente capítulo descreve o processo de testes e validação realizado na plataforma Compensa+. Após a implementação das principais funcionalidades, tornou-se necessário verificar se o sistema respondia corretamente aos requisitos definidos durante a fase de análise e modelação, bem como garantir que os diferentes componentes funcionavam de forma integrada.

Considerando que a plataforma foi desenvolvida com base numa arquitetura de microserviços, a fase de testes assumiu especial importância. O correto funcionamento do sistema depende da comunicação *entre frontend, API Gateway, Identity API, Core API, Notifications API, CompensaAI API, PostgreSQL, Redis, RabbitMQ* e serviços de infraestrutura. Assim, a validação não incidiu apenas sobre a interface, mas também sobre os fluxos internos, os *endpoints*, as regras de negócio, a comunicação entre serviços e a infraestrutura de *deploy*.

A validação da plataforma foi realizada através de diferentes tipos de testes, nomeadamente testes unitários, testes de integração, testes manuais às APIs, testes funcionais da interface, testes *end-to-end* e validação da infraestrutura em containers e ambiente *cloud*. Esta abordagem permitiu avaliar tanto o comportamento isolado de determinadas funcionalidades como o funcionamento global da solução.

Os testes realizados tiveram como principais objetivos:

- Verificar o correto funcionamento das funcionalidades implementadas;
- Validar a autenticação por link mágico e o controlo de permissões;
- Confirmar a comunicação entre *frontend, API Gateway* e microserviços;
- Validar a criação, aprovação, rejeição e consulta de pedidos de compensação;
- Testar as regras de disponibilidade e conflito de horários;
- Confirmar o funcionamento da importação de turmas e horários;
- Avaliar o comportamento das notificações e da comunicação assíncrona;
- Validar o funcionamento da CompensaAI API;
- Confirmar a persistência correta dos dados na base de dados;
- Verificar a execução dos serviços em *Docker Compose e Docker Swarm*;
- Validar a estratégia de *deploy* com *Microsoft Azure, Terraform e Ansible*;
- Identificar erros e inconsistências antes da entrega final da solução.

4.1. Estratégia de Validação

A estratégia de validação adotada combinou testes automáticos e testes manuais. Os testes automáticos foram utilizados para validar componentes específicos da

aplicação, regras de negócio e fluxos críticos. Os testes manuais permitiram verificar o comportamento da plataforma do ponto de vista do utilizador, simulando ações reais de docentes, coordenadores e administradores.

A validação foi organizada em vários níveis:

- Testes unitários: utilizados para validar métodos, serviços e regras de negócio de forma isolada;
- Testes de integração: utilizados para verificar a comunicação entre diferentes componentes da aplicação;
- Testes de API: realizados através de ferramentas como *Swagger e Postman*, permitindo testar *endpoints* e respostas dos microserviços;
- Testes funcionais da interface: realizados através da navegação nas principais páginas da plataforma;
- Testes *end-to-end:* utilizados para simular fluxos completos da aplicação através do frontend;
- Validação da infraestrutura: utilizada para confirmar a execução dos serviços, *health checks, containers*, comunicação entre componentes e *deploy* em ambiente *cloud*.

Esta estratégia permitiu validar a plataforma de forma progressiva, começando por componentes isolados e avançando para fluxos completos, como autenticação, criação de pedido, aprovação por coordenador, importação de horários, geração de notificações, assistência inteligente e execução em infraestrutura distribuída.

4.2. Ambiente de Testes

Os testes foram realizados num ambiente composto pelos principais serviços da plataforma, executados com recurso *a Docker Compose e Docker Swarm*. Esta abordagem permitiu simular a execução integrada dos microserviços e dos serviços de suporte.

Os principais componentes considerados no ambiente de testes foram:

- *Frontend ReactJS* servido através de Nginx;
- *API Gateway* implementado com *YARP*;
- *Identity API* para autenticação e autorização;
- *Core API* para regras de negócio e gestão académica;
- *Notifications API* para notificações e consumo de eventos;
- *CompensaAI API* para assistência inteligente;
- *PostgreSQL* para persistência de dados;
- *Redis* para cache;
- *RabbitMQ* para comunicação assíncrona;

- *OpenTelemetry, Prometheus, Grafana, Loki, Tempo, Portainer e Aspire Dashboard* para observabilidade e administração operacional.

A Figura 62 apresenta os serviços da plataforma em execução no ambiente de testes.

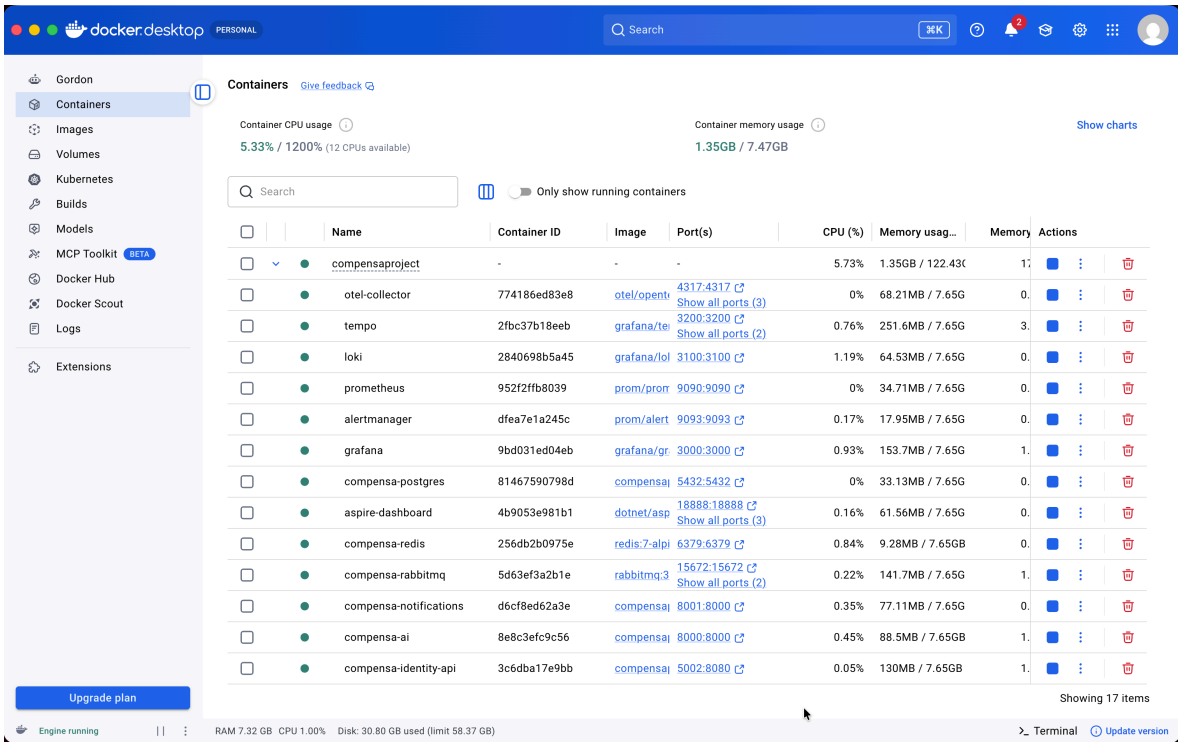


Figura 62 - Serviços da plataforma em execução durante a validação. Fonte: elaboração própria.

Como se observa na Figura 62, os diferentes componentes da plataforma foram executados de forma isolada em containers, permitindo validar a comunicação entre serviços e identificar eventuais falhas de configuração ou integração.

4.3. Validação da Autenticação e Controlo de Acessos

A autenticação foi validada por ser o ponto de entrada da plataforma e por condicionar o acesso às restantes funcionalidades. A *Identity* API implementa autenticação por *link* mágico, geração de *tokens* JWT e controlo de permissões com base no perfil do utilizador.

A validação deste módulo incidiu sobre os seguintes aspetos:

- Submissão do email pelo utilizador;
- Geração do *token* temporário associado ao *link* mágico;
- Validação do *token* recebido;
- Emissão do JWT;
- Acesso às rotas protegidas;

- Diferenciação de permissões entre Administrador, Coordenador e Docente.

A Figura 63 apresenta um exemplo de teste do fluxo de autenticação ou da resposta da API após validação do utilizador.

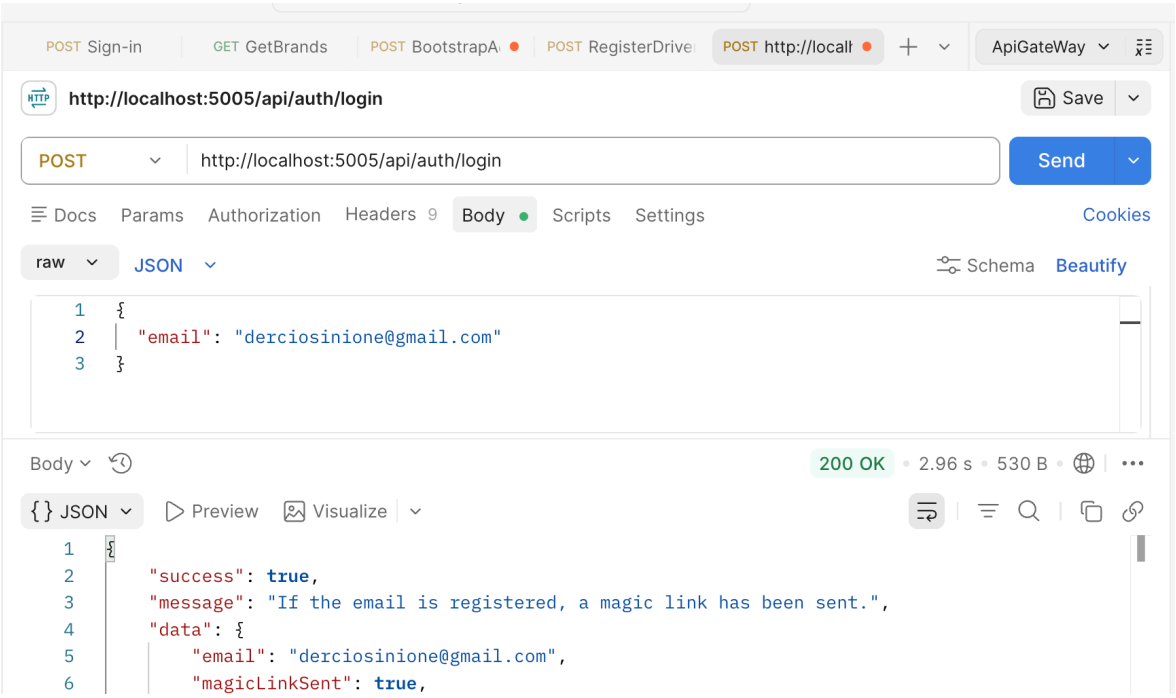


Figura 63 - Validação do fluxo de autenticação da plataforma. Fonte: elaboração própria.

Como apresentado na Figura 63, após a validação do utilizador, o sistema disponibiliza um *token JWT* utilizado nas requisições seguintes. Esta validação permitiu confirmar que apenas utilizadores autenticados conseguem aceder aos recursos protegidos.

Também foi validado o controlo de permissões por perfil. Por exemplo, o docente consegue criar pedidos de compensação, mas não deve ter acesso às ações de aprovação ou rejeição. O coordenador, por sua vez, consegue consultar pedidos pendentes e decidir sobre a sua aprovação ou recusa. Esta validação confirmou que a plataforma aplica restrições de acesso de acordo com o papel do utilizador autenticado.

4.4. Testes do Microserviço Core API

A Core API representa o microserviço central da plataforma, sendo responsável pela maior parte das regras de negócio. Por esse motivo, foi alvo de validação específica, incluindo testes de pedidos de compensação, cursos, turmas, salas, calendário, horários, importação e *dashboard*.

Os testes da Core API tiveram como objetivo validar:

- *Endpoints* principais;
- Serviços aplicacionais;
- Regras de negócio;
- Repositórios e persistência;
- Validação de permissões;
- Regras de conflito de horários;
- Publicação de eventos para notificações.

A Figura 64 apresenta um exemplo de validação dos *endpoints* da *Core API* através do *Swagger ou Postman*.

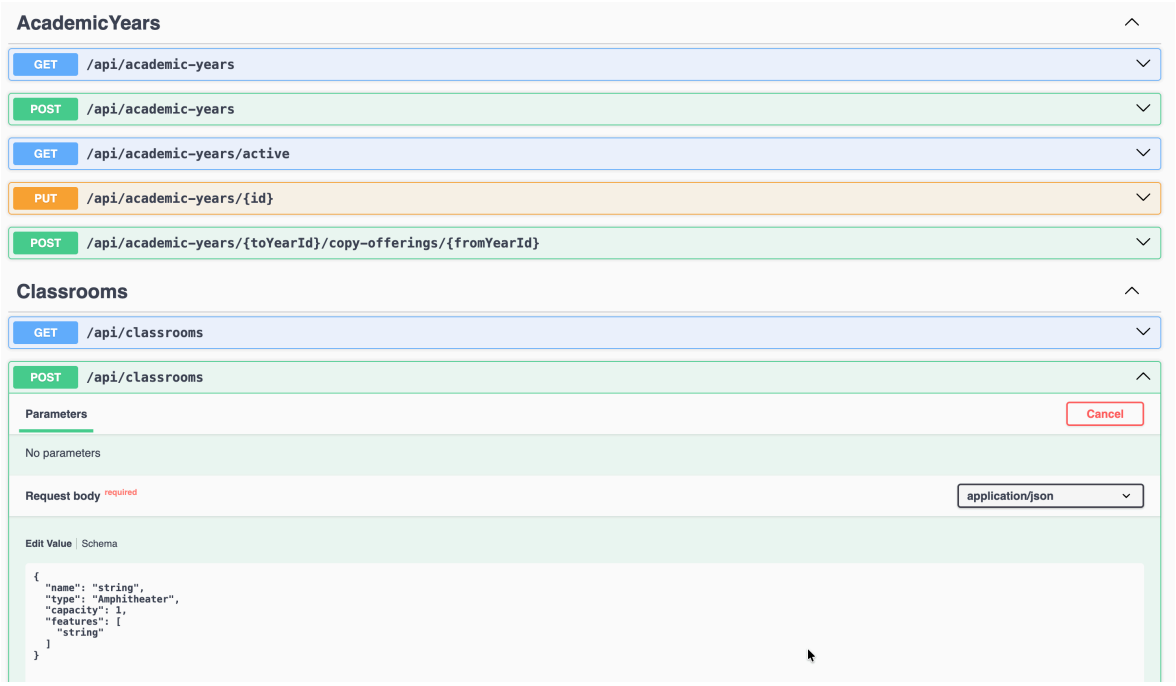


Figura 64 - Validação dos endpoints principais da Core API. Fonte: elaboração própria.

Como se observa na Figura 64, os *endpoints* da *Core API* permitem testar operações relacionadas com pedidos de compensação, cursos, salas, horários, calendário e *dashboard*. Esta validação permitiu confirmar que as respostas devolvidas pela API estavam de acordo com o comportamento esperado.

4.4.1. Testes de Pedidos de Compensação

Os pedidos de compensação constituem a funcionalidade principal da plataforma. Assim, foram testados os principais fluxos associados à criação, consulta, aprovação, rejeição e acompanhamento dos pedidos.

Foram considerados os seguintes cenários:

- Criação de pedido por docente;
- Listagem de pedidos por estado;

- Consulta de detalhes de um pedido;
- Aprovação de pedido por coordenador;
- Rejeição de pedido por coordenador;
- Tentativa de alteração por utilizador sem permissão;
- Atualização do estado do pedido;
- Registo da decisão tomada.

A Figura 65 apresenta um exemplo de criação de pedido de compensação através da API ou da interface.

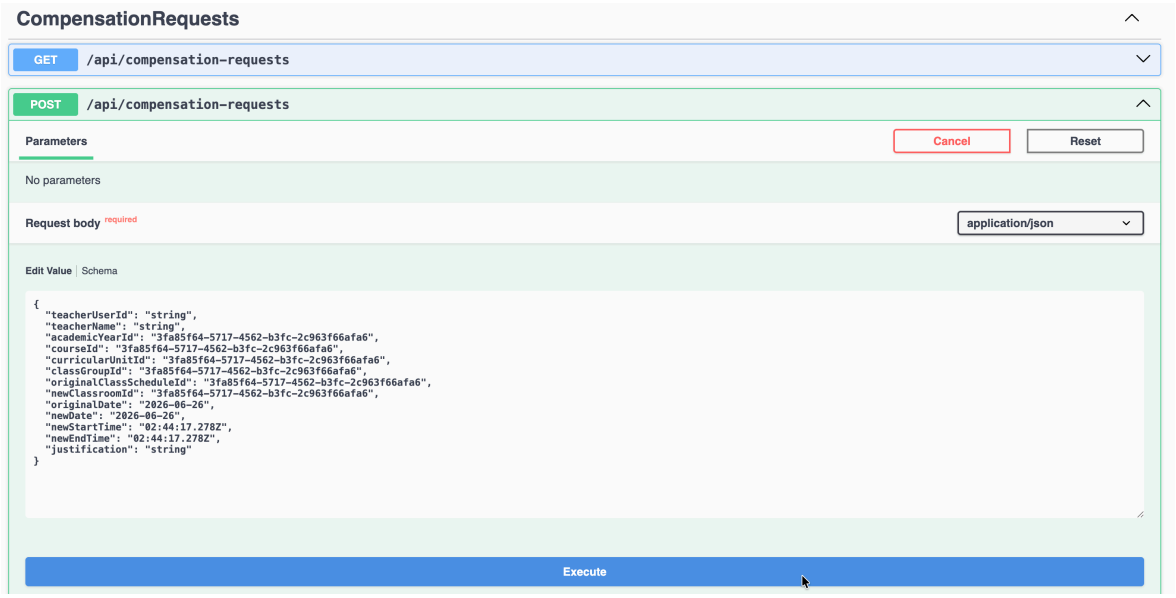


Figura 65 - Validação da criação de pedido de compensação. Fonte: elaboração própria.

Como demonstrado na Figura 65, o pedido é criado com estado inicial pendente, ficando disponível para análise pelo coordenador responsável. Este teste confirmou que os dados submetidos pelo docente são corretamente enviados ao *backend* e persistidos na base de dados.

4.4.2. Testes das Regras de Conflito de Horários

Uma parte crítica da plataforma corresponde à validação de conflitos de horários. Esta validação garante que uma aula de compensação não é marcada em sobreposição com outra aula, sala ocupada, turma indisponível ou docente indisponível.

Foram testadas regras relacionadas com:

- Disponibilidade da sala;
- Disponibilidade da turma;
- Disponibilidade do docente;
- Sobreposição com horários já existentes;

- Validade da data proposta;
- Períodos bloqueados no calendário académico;
- Permissões do utilizador que executa a operação.

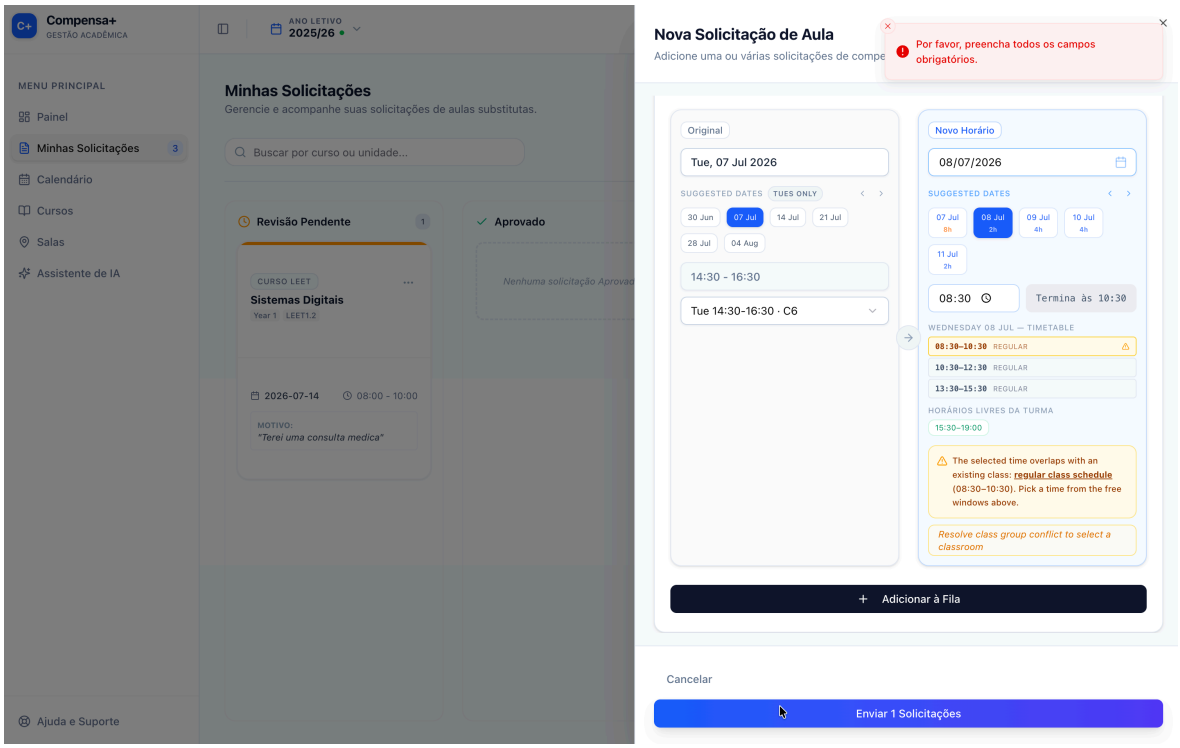


Figura 66 - Validação das regras de conflito de horário. Fonte: elaboração própria.

Como apresentado na Figura 66, quando é detetado um conflito, o sistema impede a marcação inválida e devolve uma resposta adequada. Esta validação confirmou que as regras de negócio são aplicadas no *backend* e não dependem apenas da interface do utilizador.

4.5. Validação da CompensaAI API

A CompensaAI API foi validada por representar o módulo de assistência inteligente da plataforma. Este serviço permite que o utilizador interaja com um assistente capaz de apoiar dúvidas relacionadas com pedidos, regras, horários, disponibilidade de salas e dados académicos.

A validação incidiu sobre os seguintes aspetos:

- Acesso autenticado ao *endpoint* de chat;
- Validação do JWT;
- Envio de mensagem pelo utilizador;
- Resposta do assistente inteligente;
- Integração com provedores de IA;
- Utilização de ferramentas para consulta de dados reais;

- Comunicação com a *Core API* quando necessário;
- Respeito pelas permissões do utilizador autenticado.

A Figura 67 presenta um exemplo de teste do *endpoint* de chat ou da interface do assistente inteligente.

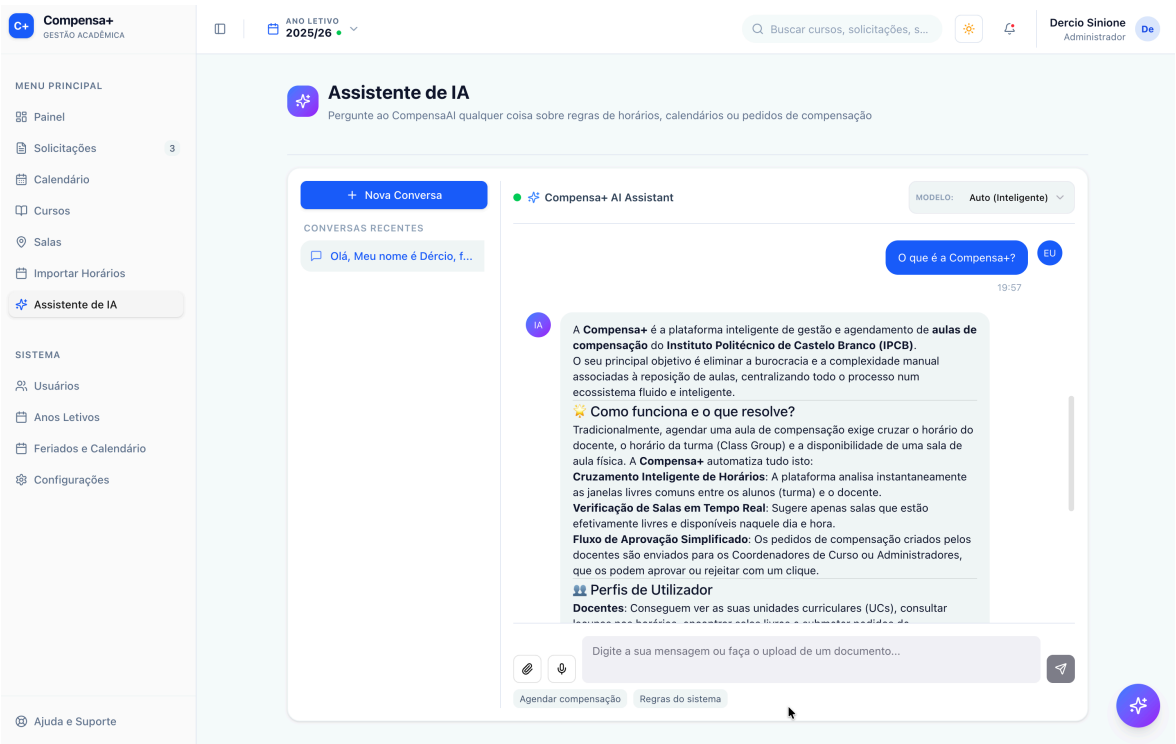


Figura 67 - Validação do assistente inteligente da plataforma Compensa+. Fonte: elaboração própria.

Como se observa na Figura 67, o assistente responde às questões do utilizador no contexto da plataforma. Sempre que necessita de consultar dados reais, a *CompensaAI* API comunica com a *Core API*, mantendo as regras de negócio centralizadas no *backend*.

Esta validação permitiu confirmar que a IA atua como camada de apoio e não substitui os mecanismos formais de autorização, validação e decisão da plataforma.

4.6. Testes *Unitários e Testes de Integração*

Para reforçar a qualidade, fiabilidade e manutenibilidade da plataforma Compensa+, foram implementados testes unitários e testes de integração em diferentes microserviços do sistema, nomeadamente na *Core API, Identity API, Notifications API* e no serviço Compensa AI.

A estratégia de testes foi adaptada à tecnologia utilizada em cada microserviço. Nos microserviços desenvolvidos em .NET, foi utilizado o xUnit, instalado através de pacotes NuGet, para a criação e execução dos testes. Para simulação de

dependências, como repositórios, serviços externos e componentes auxiliares, foi utilizada a biblioteca Moq.

No caso do microserviço Compensa AI, desenvolvido em *Python*, foram utilizados *pytest* e *unittest* para a execução dos testes. Para criação de *mocks* e simulação de dependências, recorreu-se ao unittest.*mock*, biblioteca incluída na standard *library* do *Python*.

4.6.1. Testes Unitários

Os testes unitários foram utilizados para validar regras de negócio isoladas, sem dependência direta de base de dados, Docker, filas de mensagens ou serviços externos. Estes testes permitem verificar se cada componente funciona corretamente de forma independente.

Na Core API, os testes unitários incidiram sobretudo sobre regras de conflito de horários, validação de intervalos, permissões de utilizadores e regras associadas aos pedidos de compensação. Foram testados cenários como a deteção de sobreposição entre horários, aceitação de horários consecutivos sem conflito e rejeição de intervalos inválidos, por exemplo quando a hora de início é igual ou posterior à hora de fim.

A Figura 68 apresenta a execução dos testes automáticos do microserviço *Core API* no ambiente de desenvolvimento. Os testes foram executados através do explorador de testes da IDE, sendo possível observar que todos os casos de teste aparecem com o estado *Success*, indicando que foram concluídos sem falhas.

Os testes validam diferentes componentes do Core, incluindo regras de negócio, serviços internos e cenários associados à gestão de pedidos de compensação, disponibilidade de horários, conflitos de agenda e validações aplicacionais. A presença dos vistos verdes confirma que os comportamentos esperados foram verificados corretamente, contribuindo para aumentar a confiança na estabilidade e fiabilidade do microserviço.

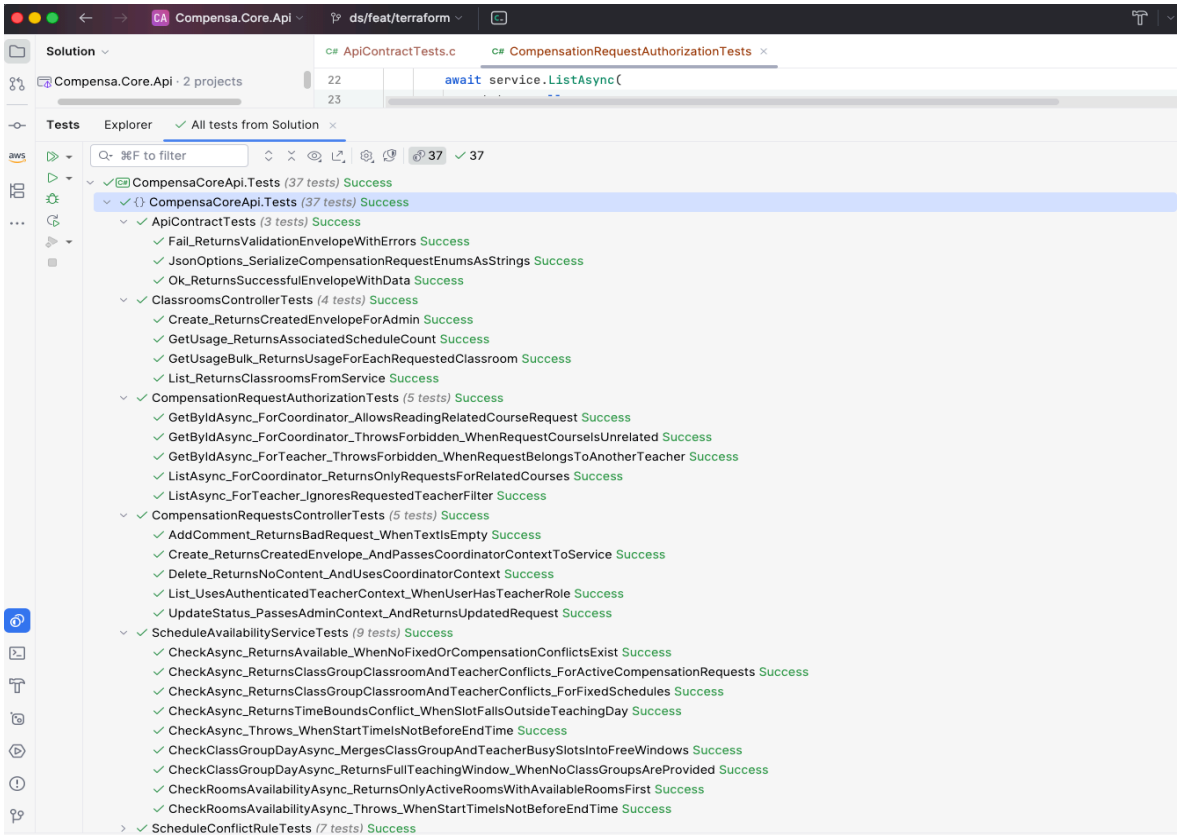


Figura 68 - Execução com sucesso dos testes automatizados do microserviço Core API. Fonte: elaboração própria

Na *Identity API,* os testes unitários foram aplicados a regras de autenticação e segurança, incluindo geração e validação de *tokens JWT, refresh tokens*, extração da identidade do utilizador a partir do *token* e validação de permissões.

Na *Notifications API,* os testes permitiram validar a lógica de criação, envio e tratamento de notificações, garantindo que os eventos recebidos são corretamente convertidos em notificações para os utilizadores.

No serviço Compensa AI, os testes unitários validaram funções internas relacionadas com processamento de pedidos, seleção de fornecedor de IA, mecanismos de *fallback*, cache e tratamento de erros de resposta.

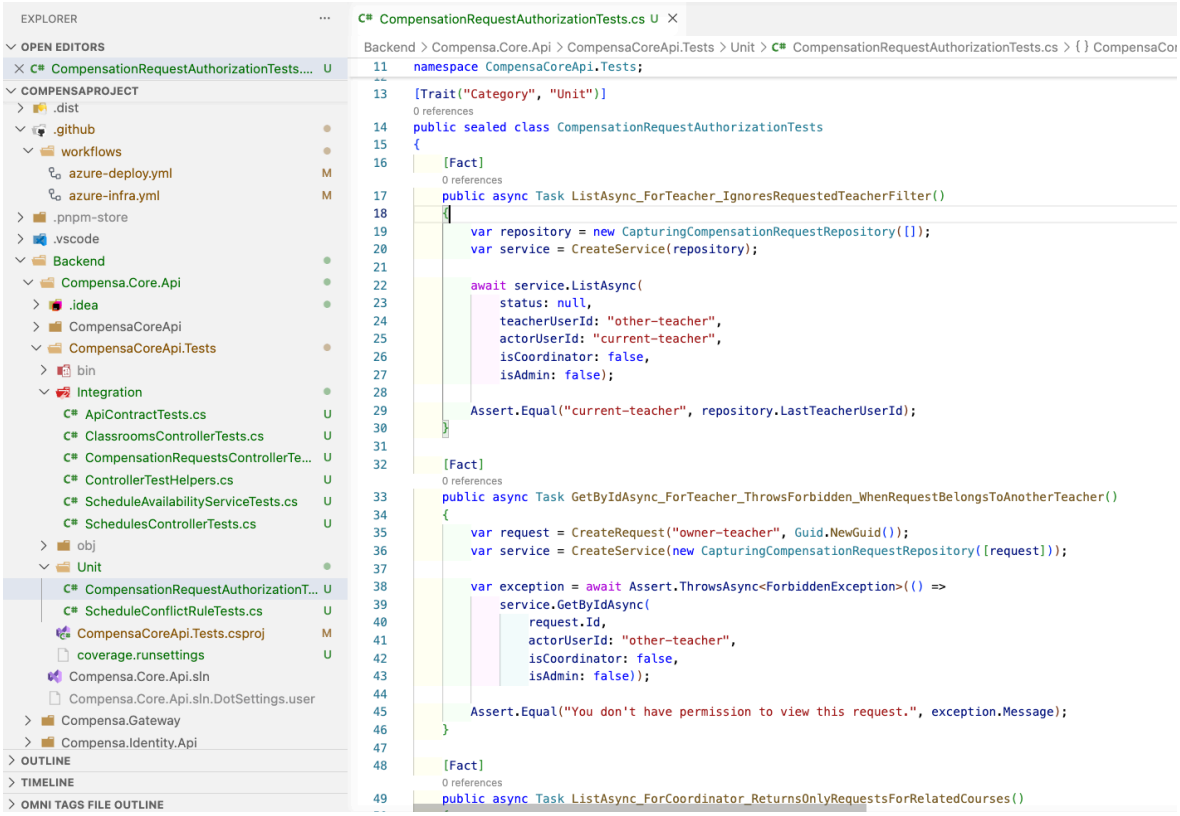


Figura 69 - Exemplo de código de teste unitário em .NET com xUnit. Fonte: elaboração própria.

4.6.2. Testes de Integrações

Os testes de integração foram utilizados para validar o comportamento de vários componentes a trabalhar em conjunto. Ao contrário dos testes unitários, estes testes verificam fluxos mais próximos do funcionamento real da aplicação, envolvendo serviços aplicacionais, repositórios simulados, regras de negócio e dependências controladas.

Na Core API, foram realizados testes de integração para o serviço de disponibilidade de horários, cobrindo cenários como conflitos de turma, sala, docente e pedidos de compensação já existentes. Também foram testadas regras de disponibilidade de salas, ordenação entre salas livres e ocupadas, limites do horário letivo entre as 08:00 e as 19:00, bem como o cálculo de janelas livres ao longo do dia.

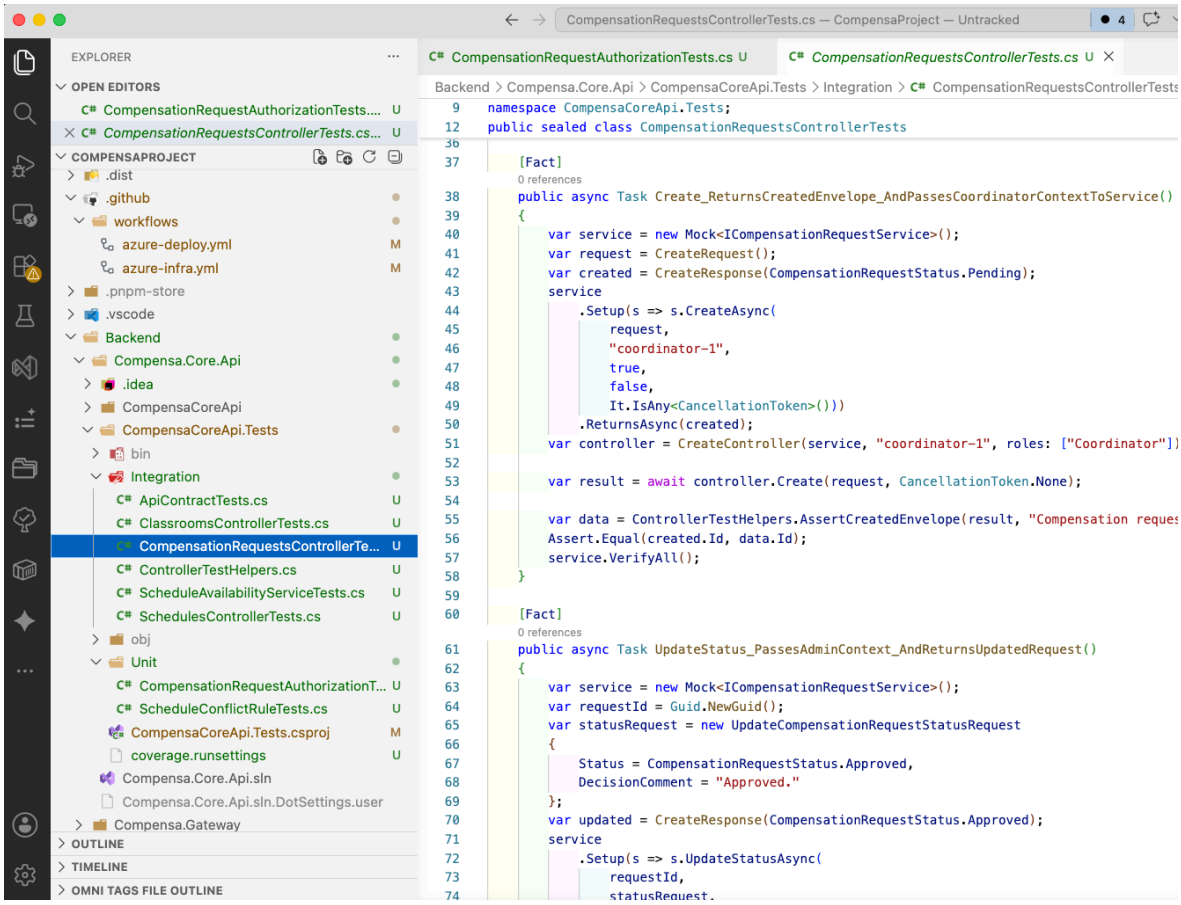


Figura 70 - Exemplo de código de teste de integração em .NET com xUnit e Moq. Fonte: elaboração própria.

Na *Identity API*, os testes de integração permitiram validar fluxos relacionados com autenticação, autorização e renovação de *tokens*, verificando a interação entre os serviços responsáveis pela identidade do utilizador e os mecanismos de segurança.

Na *Notifications API,* os testes de integração validaram o comportamento do serviço perante eventos recebidos, garantindo que as notificações são geradas corretamente a partir de ações realizadas noutros microserviços, como criação, aprovação, rejeição ou cancelamento de pedidos de compensação.

No Compensa AI, os testes de integração verificaram o funcionamento conjunto entre o serviço de IA, camada de cache, tratamento de *fallback e mocks* dos fornecedores externos. Esta abordagem permitiu validar o comportamento esperado mesmo em cenários de erro ou indisponibilidade de um fornecedor principal.

4.6.3. Cobertura de Testes

Para acompanhar a qualidade dos testes implementados, foi configurada a análise de cobertura nos principais microserviços da plataforma. No caso da Core API, foi

definido um objetivo mínimo de 70% de cobertura para os módulos associados às regras de negócio e horários.

Após a inclusão dos novos testes unitários e de integração, a cobertura das regras e serviços relacionados com horários atingiu 100%, garantindo que os principais fluxos de conflitos, disponibilidade e validação de intervalos são exercitados automaticamente.

Na Figura 71 é possível observar a execução do script test-core-coverage.sh, responsável por executar os testes automatizados da Core API e gerar o relatório de cobertura. O resultado demonstra que foram executados 37 testes, todos concluídos com sucesso, sem falhas e sem testes ignorados. A cobertura final apresentada foi de 100%, superando o valor mínimo definido de 70%. Apesar da existência de alguns avisos relacionados com dependências NuGet, estes não comprometeram a compilação nem a execução dos testes.

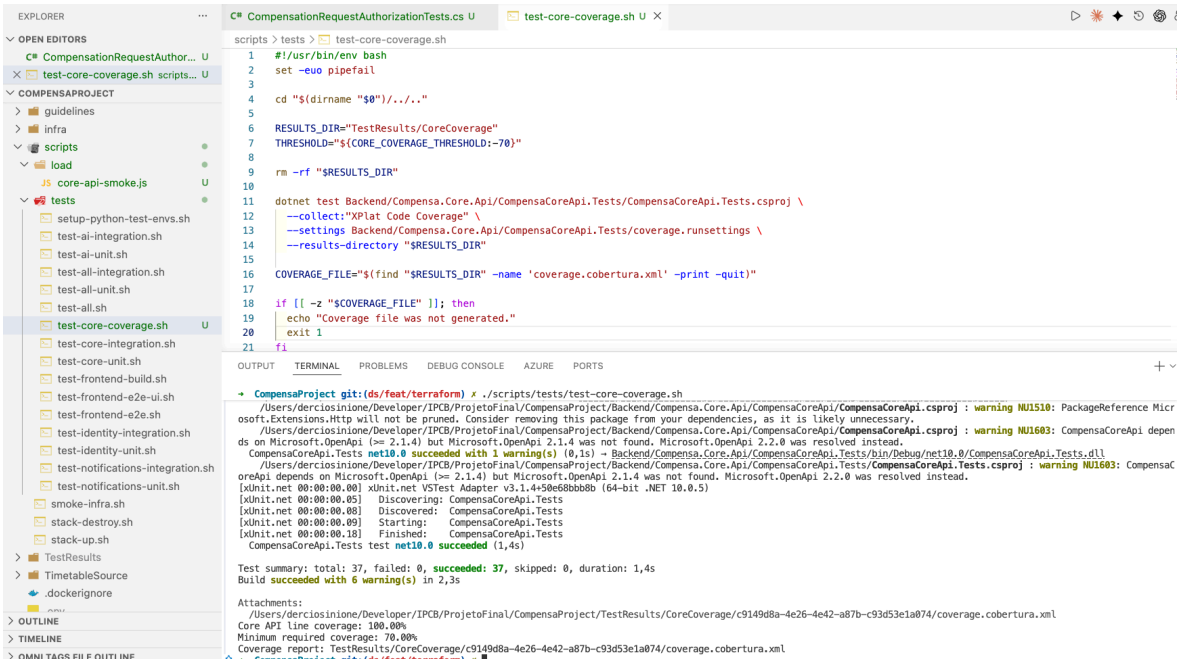


Figura 71 - Execução dos testes automatizados e validação da cobertura de código do microserviço Core API. Fonte: elaboração própria.

4.6.4. Testes de Carga do Microserviço Core API

Para além dos testes unitários e de integração, foram realizados testes de carga sobre a Core API, com o objetivo de avaliar o comportamento do microserviço perante múltiplos pedidos simultâneos. Para este teste foi utilizada a ferramenta k6, através do script scripts/load/core-api-smoke.js.

O teste foi executado em ambiente local, apontando para a Core API disponível em http://localhost:5001, com 5 utilizadores virtuais durante 60 segundos, através do seguinte comando:

```
CORE_API_BASE_URL=http://localhost:5001 K6_VUS=5 K6_DURATION=60s k6 run scripts/load/core-api-smoke.js
```

Como se observa na Figura 72, o teste executou um cenário com 5 VUs durante aproximadamente 1 minuto, realizando pedidos repetidos ao *endpoint* de verificação de estado da *Core API*. No total, foram efetuadas 300 verificações, das quais 100% foram concluídas com sucesso, sem qualquer falha registada.

Os resultados demonstram que a API respondeu corretamente durante todo o período de execução, com 0% de pedidos falhados e resposta de estado 200 OK. Ao nível do desempenho, verificou-se uma duração média dos pedidos de aproximadamente 4,93 ms, com percentil 95 de 10,01 ms, ficando dentro dos limites definidos no teste.

Estes resultados indicam que, no cenário testado, a *Core API* manteve um comportamento estável perante pedidos concorrentes, respondendo de forma consistente e sem degradação visível do serviço.



Figura 72 - Resultado do teste de carga da Core API com k6, utilizando 5 utilizadores virtuais durante 60 segundos. Fonte: elaboração própria.

4.7. Testes *End-to-End* com *Playwright*

Para complementar os testes manuais e de integração, foram utilizados testes *end-to-end* com recurso à ferramenta *Playwright*. Este tipo de teste permite simular interações reais do utilizador na interface web, validando fluxos completos da aplicação.

No contexto da Compensa+, os testes *end-to-end* foram aplicados ou preparados para cenários como:

- *Login* do utilizador;
- Acesso a rotas protegidas;
- Validação de permissões por perfil;
- Renderização do painel principal;
- Criação de pedido de compensação;
- Consulta da lista de pedidos;

- Aprovação de pedido pelo coordenador;
- Consulta de notificações;
- Importação de turmas e horários;
- Navegação entre páginas principais.

A Figura 73 apresenta a execução dos testes *end-to-end* com *Playwright*.

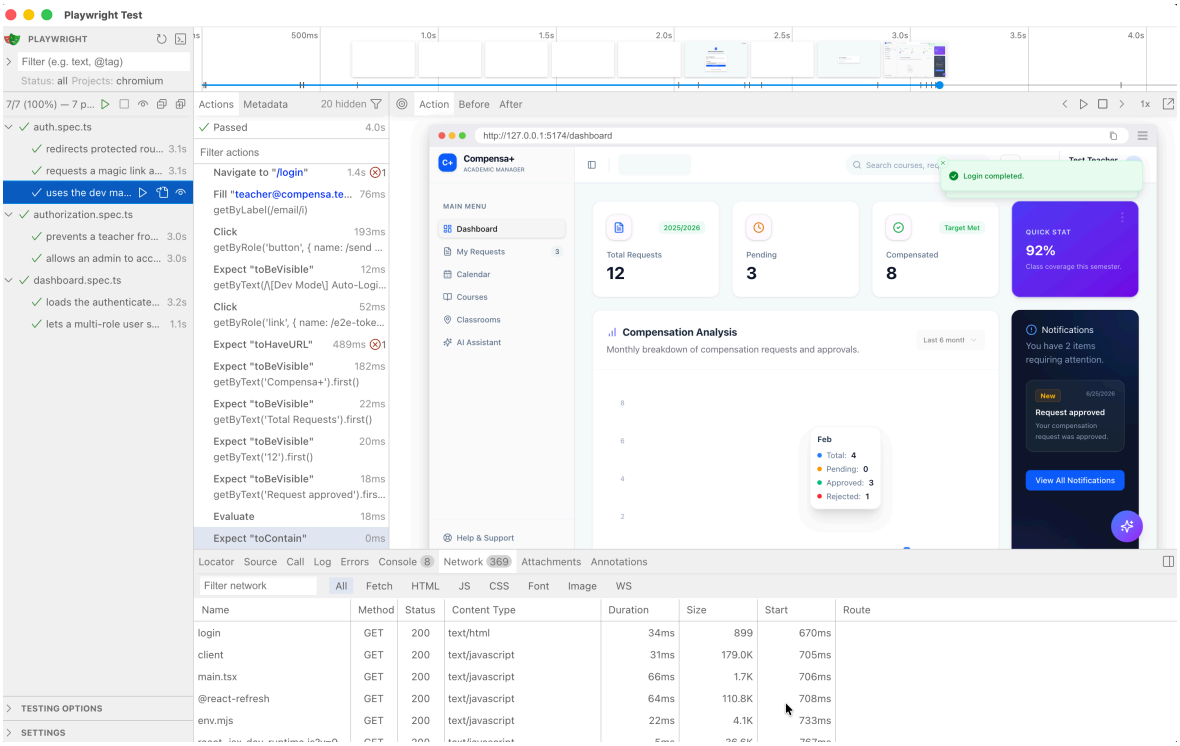


Figura 73 - Execução de testes end-to-end com Playwright. Fonte: elaboração própria.

Como se observa na Figura 73, os testes *end-to-end* permitem validar a plataforma de forma próxima à utilização real. Esta abordagem aumenta a confiança no funcionamento da aplicação, pois testa a interação entre interface, rotas protegidas, serviços *frontend* e *backend*.

4.8. Validação da Infraestrutura *Cloud* e *Deploy*

Para além das funcionalidades da aplicação, foi realizada a validação da infraestrutura de execução e da estratégia de *deploy* da plataforma. Esta etapa permitiu confirmar que a solução estava preparada para funcionar num ambiente mais próximo de produção, recorrendo a Microsoft *Azure, Terraform, Ansible, Docker Swarm, Nginx, Portainer* e ferramentas de observabilidade.

A estratégia de *deploy* definida para a plataforma Compensa+ assenta em três componentes principais:

- Microsoft *Azure*: infraestrutura *cloud* utilizada para alojamento das máquinas virtuais e recursos de rede;

- *Terraform*: ferramenta utilizada para provisionar a infraestrutura como código;
- *Ansible*: ferramenta utilizada para automatizar a configuração das máquinas virtuais e preparar o ambiente de execução.

A validação da infraestrutura *cloud* incidiu sobre os seguintes aspetos:

- Criação do *Resource Group* dedicado à plataforma;
- Definição da *Virtual Network* e respetivas sub-redes;
- Configuração dos Network *Security Groups*;
- Criação e organização das máquinas virtuais;
- Associação das máquinas virtuais à rede privada;
- Configuração do *Docker Engine*;
- Inicialização do *Docker Swarm*;
- *Deploy* dos serviços através do ficheiro *docker-stack.yml*;
- Exposição controlada do *frontend* e do *API Gateway*;
- Execução dos serviços de dados, cache, mensageria e monitorização;
- Acompanhamento dos serviços através do *Portainer e Grafana*.

A Figura 74 apresenta a topologia da infraestrutura *cloud* definida para a plataforma Compensa+ na *Microsoft Azure*.

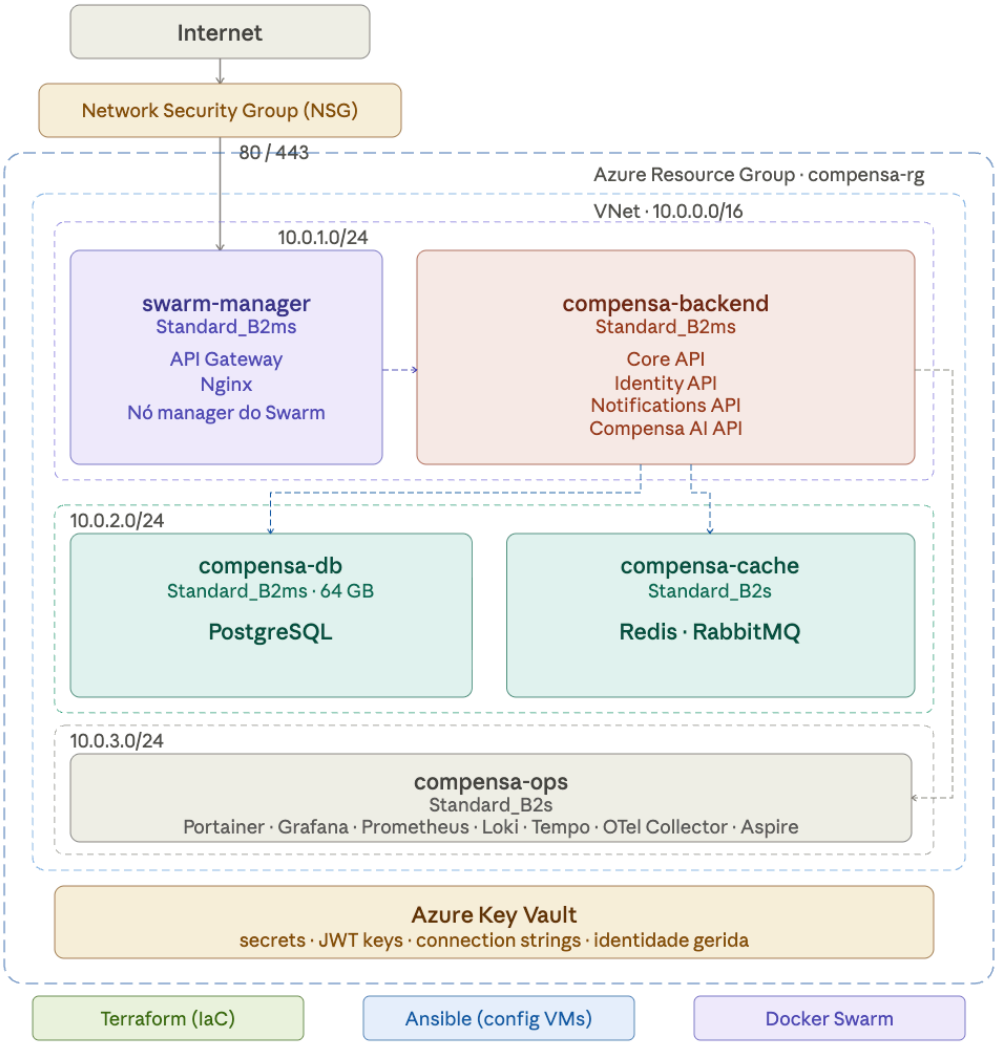


Figura 74 - Topologia da infraestrutura de produção da plataforma Compensa+ na Microsoft Azure. Fonte: elaboração própria.

Como apresentado na Figura 74, a infraestrutura foi organizada em máquinas virtuais com responsabilidades distintas, incluindo serviços de aplicação, base de dados, cache, mensageria e observabilidade. Esta divisão permite maior organização operacional e facilita a manutenção dos componentes da plataforma.

A validação do *Terraform* consistiu em verificar se os recursos definidos nos ficheiros de configuração eram corretamente planeados e provisionados. Esta etapa permitiu garantir que a infraestrutura poderia ser recriada de forma controlada e rastreável, evitando configurações manuais inconsistentes.

A Figura 75 apresenta a validação ou aplicação da infraestrutura com *Terraform*.

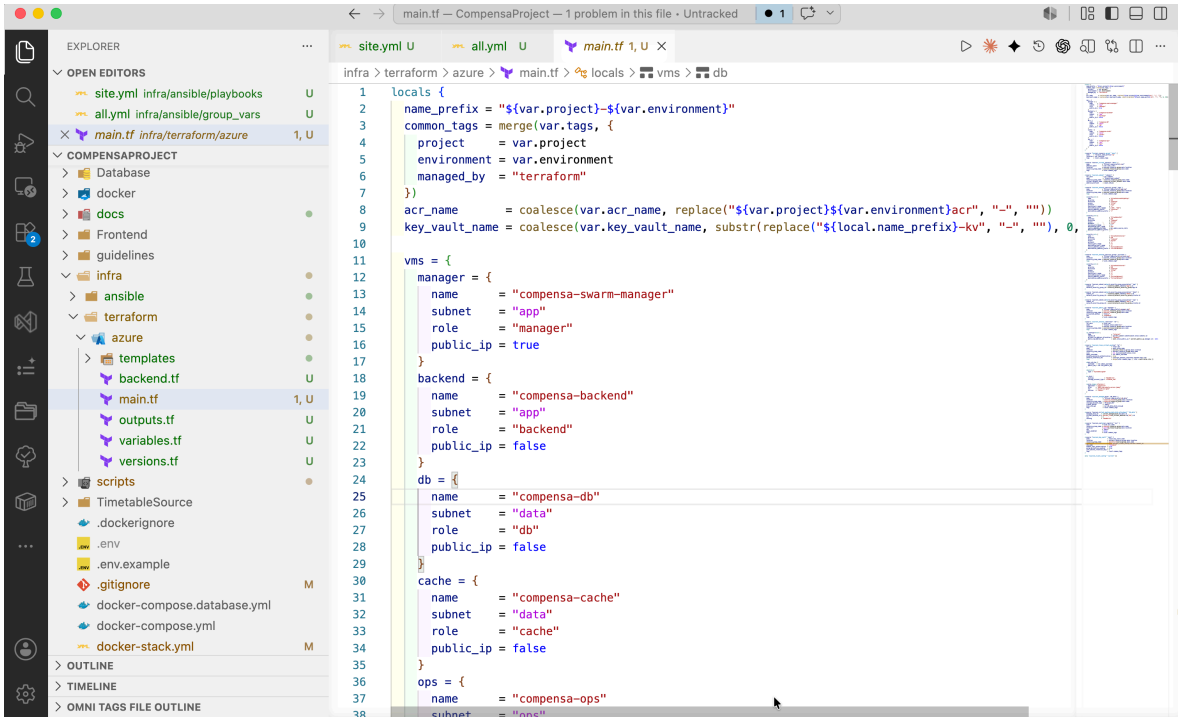


Figura 75 - Validação do provisionamento da infraestrutura com Terraform. Fonte: elaboração própria.

Como se observa na Figura 76, o *Terraform* permite confirmar os recursos a criar, alterar ou manter, antes da aplicação definitiva da infraestrutura. Esta abordagem contribui para maior controlo e rastreabilidade no processo de *deploy*.

Após o provisionamento da infraestrutura, foi validada a configuração das máquinas virtuais com *Ansible*. Esta etapa incluiu a instalação do Docker, preparação do ambiente, configuração do *Docker Swarm*, aplicação de dependências e disponibilização dos ficheiros necessários para execução da *stack*.

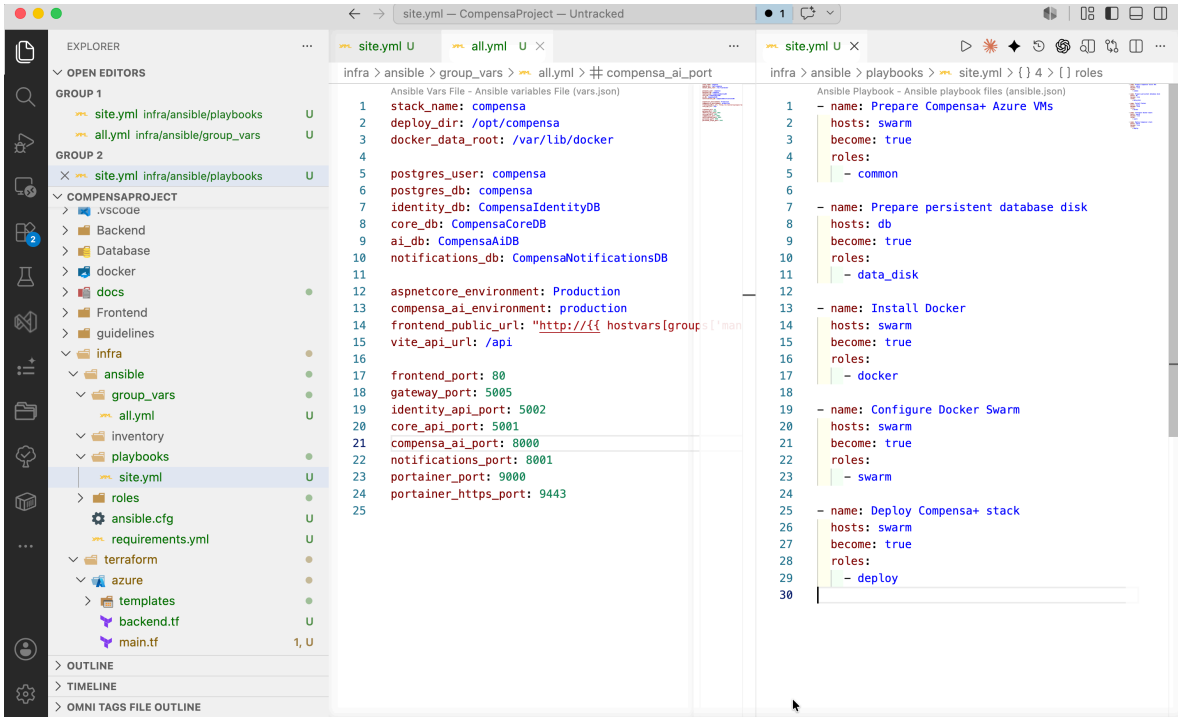


Figura 76 - Configuração automatizada das máquinas virtuais com Ansible. Fonte: elaboração própria.

Como demonstrado na Figura 76, o *Ansible* permite automatizar a configuração das máquinas, reduzindo intervenção manual e diminuindo a probabilidade de erros operacionais.

4.9. Resultados da Validação

Após a realização dos testes, foi possível verificar que as principais funcionalidades da plataforma se encontram alinhadas com os requisitos definidos. Os fluxos de autenticação, gestão de pedidos, validação por coordenadores, notificações, gestão de dados académicos, importação de turmas e horários, assistente inteligente, comunicação entre serviços e infraestrutura de *deploy* apresentaram comportamento adequado.

A Tabela 8 apresenta um resumo dos principais cenários validados.

Tabela 8 - Resumo dos cenários de validação. Fonte: elaboração própria.

Cenário testado	Resultado esperado	Resultado obtido
Autenticação por *link* mágico	Utilizador autenticado e *token* JWT emitido	Conforme esperado
Controlo de permissões	Funcionalidades apresentadas conforme o perfil	Conforme esperado

Criação de pedido	Pedido criado com estado pendente	Conforme esperado
Aprovação de pedido	Estado do pedido atualizado para aprovado	Conforme esperado
Rejeição de pedido	Estado do pedido atualizado para recusado	Conforme esperado
Validação de conflitos	Sistema impede marcações incompatíveis	Conforme esperado
Consulta de calendário	Eventos e horários apresentados corretamente	Conforme esperado
Consulta de cursos, turmas e salas	Dados académicos apresentados corretamente	Conforme esperado
Importação de turmas e horários	Dados extraídos e apresentados para validação	Conforme esperado
Criação automática de dados académicos	Elementos inexistentes criados automaticamente	Conforme esperado
Publicação de eventos	Evento publicado no *RabbitMQ*	Conforme esperado
Processamento de notificações	Notificação criada e apresentada ao utilizador	Conforme esperado
CompensaAI API	Assistente responde e consulta dados quando necessário	Conforme esperado
Frontend	Páginas e fluxos principais funcionam corretamente	Conforme esperado
Playwright	Fluxos simulados executados na interface	Conforme esperado
Docker Compose	Serviços executados localmente	Conforme esperado
Docker Swarm	Serviços executados com réplicas e gestão distribuída	Conforme esperado
Terraform	Infraestrutura definida e provisionada como código	Conforme esperado
Ansible	Máquinas configuradas de forma automatizada	Conforme esperado
Azure	Topologia cloud preparada para execução da plataforma	Conforme esperado
Observabilidade	Métricas, *logs* e traces recolhidos e visualizados	Conforme esperado

Os testes permitiram ainda identificar pequenos ajustes na interface, nos fluxos de navegação, na configuração dos serviços e na integração entre componentes. Estes ajustes foram corrigidos durante o processo de desenvolvimento,

contribuindo para melhorar a estabilidade, usabilidade e fiabilidade da solução implementada.

4.10. Considerações Finais

A fase de testes e validação permitiu verificar o correto funcionamento das principais funcionalidades da plataforma Compensa+. A combinação de testes unitários, testes de integração, testes de API, testes funcionais da interface, testes *end-to-end*, validação da infraestrutura e validação da observabilidade possibilitou uma análise mais completa do sistema.

Os resultados obtidos demonstram que a plataforma cumpre os principais objetivos definidos, permitindo autenticação de utilizadores, submissão e validação de pedidos de compensação, gestão de dados académicos, consulta de calendário, envio de notificações, importação automática de turmas e horários, utilização do assistente inteligente e execução em ambiente distribuído.

A validação da *Core API* confirmou o correto funcionamento das regras principais de negócio, especialmente no que diz respeito à criação de pedidos, aprovação, rejeição e validação de conflitos de horários. A validação da *Notifications API* confirmou que os eventos publicados pela *Core API* são processados de forma assíncrona. A validação da *CompensaAI API* demonstrou que o assistente inteligente pode apoiar os utilizadores sem substituir as regras formais de negócio.

O módulo de importação de turmas e horários destacou-se como uma das funcionalidades mais relevantes, uma vez que permite reduzir o trabalho manual de configuração e aproximar a plataforma do contexto real da Escola Superior de Tecnologia.

A validação da infraestrutura *cloud* permitiu confirmar que a plataforma está preparada para execução num cenário mais próximo de produção, recorrendo a *Azure, Terraform, Ansible, Docker Swarm, Portainer* e ferramentas de observabilidade. Esta validação reforça a viabilidade técnica da solução e demonstra que a plataforma foi pensada não apenas como aplicação funcional, mas também como sistema preparado para operação e evolução futura.

Apesar dos resultados positivos, a validação poderá ser aprofundada no futuro através da realização de testes com utilizadores reais, recolha de feedback junto de docentes e coordenadores, testes de desempenho, testes de segurança mais detalhados, testes de carga e validação contínua em ambiente *cloud*. Desta forma, a fase de testes confirmou a viabilidade da solução implementada e reforçou a qualidade da plataforma antes da apresentação final do projeto.

5. Conclusão e Trabalho Futuro

O presente trabalho teve como objetivo desenvolver e implementar a plataforma Compensa+, uma solução web destinada à gestão centralizada, automatizada e rastreável de aulas de compensação no contexto do ensino superior. A proposta surgiu da necessidade de melhorar um processo que, em muitos casos, continua a depender de mecanismos manuais ou informais, como troca de emails, folhas de cálculo e validações descentralizadas.

Durante o Projeto I foi realizada a análise e modelação da solução, recorrendo à metodologia *ICONIX*. Esta fase permitiu identificar os principais requisitos funcionais e não funcionais, definir os perfis de utilizador, modelar os casos de uso, estruturar o domínio do sistema e construir um protótipo não funcional. O Projeto II deu continuidade a esse trabalho, tendo como foco a implementação da solução modelada, transformando o protótipo e os modelos definidos numa aplicação funcional.

A plataforma desenvolvida permite responder aos principais objetivos inicialmente definidos, disponibilizando funcionalidades como autenticação de utilizadores, gestão de perfis, submissão de pedidos de compensação, aprovação e rejeição de pedidos, consulta de histórico, gestão de cursos, turmas, unidades curriculares, salas, calendário académico, notificações, assistente inteligente e importação automática de turmas e horários.

Do ponto de vista técnico, a solução foi implementada com uma arquitetura baseada em microserviços, composta *por frontend web, API Gateway, Identity API, Core API, Notifications API e CompensaAI API*. Esta abordagem permitiu separar responsabilidades, melhorar a organização do código e preparar a plataforma para futuras evoluções. A utilização de tecnologias como *ReactJS, .NET, FastAPI, PostgreSQL, Redis, RabbitMQ, Docker, Docker Swarm, Terraform, Ansible, Microsoft Azure* e ferramentas de observabilidade contribuiu para a construção de uma solução modular, escalável e alinhada com práticas atuais de desenvolvimento de *software*.

A *Core API* assumiu um papel central na implementação, concentrando as principais regras de negócio da plataforma. Este microserviço foi organizado com inspiração em *Domain-Driven Design*, recorrendo a entidades de domínio, serviços aplicacionais, repositórios e regras de validação. Esta estrutura permitiu representar conceitos essenciais do domínio académico, como pedido de compensação, curso, turma, sala, horário, unidade curricular e ano letivo.

Uma das funcionalidades de maior relevância implementadas foi o módulo de importação de turmas e horários. Este módulo permite carregar ficheiros HTML exportados do sistema utilizado pela instituição, extrair automaticamente

informações académicas e criar ou atualizar cursos, turmas, unidades curriculares, salas e horários. Esta funcionalidade aproxima a plataforma do funcionamento real da Escola Superior de Tecnologia e reduz significativamente o esforço manual associado à configuração inicial dos dados académicos.

A plataforma integrou ainda uma componente de assistência inteligente através da *CompensaAI API*. Este microserviço permite disponibilizar apoio contextual aos utilizadores, ajudando na interpretação de regras, consulta de dados e acompanhamento de pedidos. Apesar disso, as regras de negócio e permissões permanecem centralizadas na *Core API*, garantindo que a inteligência artificial atua apenas como camada auxiliar e não como substituto dos mecanismos formais de validação.

Ao nível da infraestrutura, foi definida uma estratégia de *deploy* baseada em *Microsoft Azure, Terraform, Ansible e Docker Swarm.* Esta abordagem permitiu preparar a plataforma para um cenário mais próximo de produção, com máquinas virtuais organizadas por função, provisionamento da infraestrutura como código, configuração automatizada dos servidores, gestão distribuída dos containers e monitorização operacional através de ferramentas como *Portainer, Grafana, Prometheus, Loki,* Tempo *e OpenTelemetry.*

A fase de implementação permitiu consolidar a ligação entre os modelos definidos no Projeto I e a solução final. Os casos de uso relacionados com autenticação, pedidos de compensação, validação por coordenador, notificações, gestão académica, importação de horários e apoio ao utilizador foram materializados em interfaces, *endpoints*, serviços, entidades de domínio e repositórios. Desta forma, o sistema passou de uma fase conceptual para uma solução funcional, organizada e preparada para utilização em contexto académico.

A validação da plataforma foi realizada através de testes funcionais, testes unitários, testes de integração, testes de API, testes *end-to-end*, validação da infraestrutura e validação da observabilidade. Estes testes permitiram verificar o comportamento das principais funcionalidades, confirmar a comunicação entre os microserviços, validar a persistência dos dados, testar a estratégia de *deploy* e identificar ajustes necessários na interface e nos fluxos de utilização.

Apesar dos resultados alcançados, o desenvolvimento da plataforma também apresentou alguns desafios. Entre eles destacam-se a integração entre os diferentes microserviços, a gestão da comunicação assíncrona através do *RabbitMQ*, a organização dos dados académicos, a implementação do módulo de importação de horários, a validação de conflitos, a configuração da infraestrutura *cloud* e a necessidade de garantir coerência entre *frontend, backend*, base de dados e serviços de suporte. Estes desafios foram ultrapassados através de uma organização progressiva do desenvolvimento, testes contínuos e utilização de

ferramentas de apoio, como *Docker, Jira, Postman, Playwright, Terraform, Ansible* e ferramentas de monitorização.

De forma geral, considera-se que os objetivos do Projeto II foram alcançados. A plataforma Compensa+ apresenta-se como uma solução capaz de centralizar e automatizar a gestão das aulas de compensação, contribuindo para reduzir erros, melhorar a comunicação entre docentes e coordenadores, aumentar a transparência do processo e apoiar a tomada de decisão. A solução desenvolvida representa uma base sólida para futura utilização, validação institucional e evolução funcional.

5.1. Trabalho Futuro

Embora a plataforma Compensa+ já apresente um conjunto significativo de funcionalidades implementadas, existem várias melhorias e evoluções que poderão ser realizadas em trabalhos futuros. Estas melhorias podem ser organizadas por prioridade, considerando o impacto esperado na plataforma e a viabilidade de implementação.

A primeira prioridade consiste na consolidação do *deploy* da plataforma nos servidores do IPCB. Embora a solução já esteja preparada para execução em containers e orquestração com *Docker Swarm*, será importante adaptar a infraestrutura ao ambiente real disponibilizado pela instituição. Esta etapa deverá incluir a configuração definitiva dos servidores, regras de rede, domínios, certificados, políticas de segurança, volumes persistentes, backups e mecanismos de recuperação em caso de falha. Esta evolução terá elevado impacto, uma vez que permitirá aproximar a plataforma de um cenário real de utilização institucional.

A segunda prioridade consiste no reforço do pipeline de integração e entrega contínua. No futuro, poderá ser implementado um processo completo de CI/CD, permitindo automatizar a execução de testes, construção de imagens Docker, publicação em registo privado e atualização dos serviços no ambiente de execução. Esta melhoria reduziria a intervenção manual no processo de *deploy* e aumentaria a rastreabilidade das versões publicadas.

A terceira prioridade passa pelo aprofundamento dos testes com utilizadores reais, nomeadamente docentes, coordenadores e administradores. Esta validação permitiria recolher feedback sobre a clareza dos fluxos, facilidade de submissão dos pedidos, compreensão das notificações, organização das páginas e adequação das permissões aos diferentes perfis. Os resultados destes testes poderiam orientar melhorias de usabilidade e acessibilidade na interface.

A quarta prioridade corresponde ao reforço da segurança da plataforma. Sendo uma solução destinada à gestão de dados académicos, será importante realizar testes específicos de autenticação, autorização, proteção de *endpoints*, validação de permissões, gestão de *tokens*, exposição de dados sensíveis e segurança das comunicações. Também poderá ser estudada a integração com mecanismos institucionais de autenticação, caso venham a estar disponíveis.

A quinta prioridade incide sobre o módulo de importação de turmas e horários. Apesar de a funcionalidade já permitir processar ficheiros HTML, poderão ser adicionadas melhorias como suporte a outros formatos de ficheiro, deteção automática de inconsistências, relatórios de erros de importação, validação mais robusta de conflitos e mecanismos de correção assistida dos dados importados. Estas melhorias tornariam o processo mais adaptável a diferentes estruturas de horários.

A sexta prioridade está relacionada com a evolução da *CompensaAI API*. No futuro, o assistente inteligente poderá ser melhorado através de maior integração com os dados internos da plataforma, mecanismos de explicabilidade das respostas, limitação de custos por utilização, métricas de latência, controlo de *tokens* e avaliação da qualidade das respostas. Também poderá ser aprofundado o uso de ferramentas controladas, permitindo que o assistente consulte dados reais da plataforma sem comprometer as regras de segurança e autorização.

A sétima prioridade consiste na criação de relatórios estatísticos e indicadores de gestão. Poderão ser disponibilizados *dashboards* com métricas como número de pedidos por curso, taxa de aprovação, principais motivos de rejeição, tempo médio de resposta, volume de notificações enviadas, conflitos detetados, utilização das salas e taxa de sucesso das importações. Estes indicadores poderiam apoiar coordenadores e administradores na tomada de decisão.

A oitava prioridade corresponde à integração com sistemas académicos institucionais. Esta evolução permitiria sincronizar automaticamente dados como docentes, unidades curriculares, turmas, salas, horários e calendário académico. A integração com sistemas existentes reduziria ainda mais o trabalho manual e permitiria manter a plataforma atualizada com os dados oficiais da instituição.

Por fim, como evolução de menor prioridade inicial, poderá ser considerada a criação de uma aplicação móvel ou de uma versão progressiva da aplicação web. Esta melhoria permitiria que docentes e coordenadores acompanhassem pedidos, recebessem notificações e consultassem horários de forma mais rápida através de dispositivos móveis.

Em síntese, o trabalho futuro deverá concentrar-se, numa primeira fase, na consolidação do ambiente de execução nos servidores do IPCB, na automação do

deploy, na validação com utilizadores reais e no reforço da segurança. Numa fase posterior, poderão ser desenvolvidas melhorias funcionais e analíticas, como relatórios estatísticos, evolução do assistente inteligente, integração com sistemas institucionais e suporte móvel.

Referências Bibliográficas

Axios. (2025). *Axios Documentation*. Retrieved 06 15, 2026, from https://axios-http.com/docs/intro

Burns, B. (2018). *Designing Distributed Systems.* Sebastopol: O'Reilly Media.

Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison-Wesley: Boston.

FastAPI. (2026). *FastAPI*. Retrieved 05 30, 2026, from https://fastapi.tiangolo.com/

FenixEdu. (2026). *FenixEdu*. Retrieved from https://fenixedu.org/

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-Based Software Architectures.* Irvine: University of California.

Fowler, M. (2002). *Patterns of Enterprise Application Architecture.* Boston: Addison-Wesley.

Google. (2024). *Gemini API Documentation*. Retrieved from https://ai.google.dev/docs

Grafana Labs. (2025). *Grafana Documentation*. Retrieved 06 05, 2026, from https://grafana.com/docs

HashiCorp. (2025). *Terraform documentation*. Retrieved from https://developer.hashicorp.com/terraform/docs

Hohpe, G. W. (2003). *Enterprise Integration Patterns.* Boston: Addison-Wesley.

Laudon, K. C., & Laudon, J. P. (2022). *Management Information Systems: Managing the Digital Firm.*

Merkel, D. (2014). Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux Journal*(239).

Meta Platforms, Inc. (n.d.). *React Reference Overview*. Retrieved 05 30, 2026, from https://react.dev/reference/react

Microsoft. (2025). *.NET Aspire Documentation*. (Microsoft Learn) Retrieved 05 30, 2026, from https://learn.microsoft.com/dotnet/aspire

Microsoft. (2025). *ASP.NET Core Documentation*. Retrieved 06 05, 2026, from https://learn.microsoft.com/aspnet/core

Microsoft. (2025). *TypeScript Documentation*. Retrieved 05 30, 2026, from https://www.typescriptlang.org/docs

Microsoft. (2026). *Playwright Documentation*. Retrieved 05 30, 2026, from https://playwright.dev/docs/intro

Morris, K. (2020). *Infrastructure as Code 2nd ed. .* O'Reilly Media.

Newman, S. (2021). *Building Microservices.* O'Reilly Media: Sebastopol.

Nginx Inc. (2025). *NGINX documentation*. Retrieved from https://nginx.org/en/docs/

OpenAI. (2024). *GPT-4.2-mini model documentation*. Retrieved from https://platform.openai.com/docs

OpenTelemetry Authors. (2025). *OpenTelemetry Documentation*. Retrieved 06 05, 2026, from https://opentelemetry.io/docs

Portainer.io. (2025). *Portainer documentation*. Retrieved from https://docs.portainer.io

PostgreSQL Global Development Group. (2025). *PostgreSQL Documentation*. Retrieved 06 10, 2026, from https://www.postgresql.org/docs

Pressman, R., & Maxim, B. (2020). *Software Engineering: A Practitioner's Approach.*

Prometheus Authors. (2025). *Prometheus Documentation*. Retrieved 06 05, 2026, from https://prometheus.io/docs

Red Hat. (2025). *Ansible documentation*. Retrieved from https://docs.ansible.com

Redis. (2025). *Redis Documentation*. Retrieved 06 15, 2026, from https://redis.io/docs

Richardson, C. (2018). *Microservices Patterns.* Shelter Island: Manning Publications.

Rosenberg, D., & Stephens, M. (2007). *Use Case Driven Object Modeling with UML: Theory and Practice.*

Selwyn, N. (2016). *Education and Technology: Key Issues and Debates.*

Sommerville, I. (2016). *Software Engineering.*

Tailwind Labs. (2026). *Installing Tailwind CSS with Vite*. (Tailwind CSS Documentation) Retrieved 05 30, 2026, from https://tailwindcss.com/docs/installation/using-vite

TanStack. (2026). *TanStack Query*. Retrieved 05 30, 2026, from https://tanstack.com/query/latest

UNESCO. (2023). *Guidance for Generative AI in Education and Research.*

Vite. (2026). *Getting Started, Vite Documentation*. Retrieved 05 30, 2026, from https://vite.dev/guide/

VMware. (2025). *RabbitMQ Documentation*. Retrieved 06 05, 2026, from https://www.rabbitmq.com/documentation.html