



INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA (ISEL)

DEPARTAMENTO DE ENGENHARIA ELETRÓNICA E DE
TELECOMUNICAÇÕES E COMPUTADORES (DEETC)

LEIM

LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA

UNIDADE CURRICULAR DE PROJETO

MMDv2 - Manage My Domain Version 2

Tomás Matos (48286)

Orientadores

Doutor Paulo Trigo

Mestre Alexandre Brito

Junho, 2025

Resumo

O presente projeto incide sobre o desenvolvimento de uma solução digital para apoiar a gestão de recursos humanos e técnicos, a sua afetação a projetos e o acompanhamento de oportunidades em contexto organizacional.

A ferramenta concebida, **Manage My de Domain** (MMD), foi desenvolvida no contexto da **Cloud Platform Solution Factory** (CSF) da empresa Accenture, com o objetivo de apoiar a equipa de liderança na visualização e organização da informação relativa aos recursos disponíveis, projetos em curso e novas oportunidades de negócio.

Este projeto corresponde à evolução da aplicação pré-existente, denominada **Manage My de Domain Version 1** (MMDv1), que, apesar de funcional, apresentava limitações significativas. Entre os principais problemas identificados destacavam-se: a utilização de uma base de dados não relacional (**MongoDB** [MongoDB8.0.4, 2009]), pouco adequada à complexidade das relações entre entidades como recursos, oportunidades e projetos.

Em resposta a estas limitações, foi desenvolvido o **Manage My de Domain Version 2** (MMDv2), com base numa arquitetura modular monolítica, seguindo o padrão **Model-View-Controller** (MVC) [MVC, 2023], e recorrendo a tecnologias amplamente adotadas no mercado: **Next.js** [Vercel, 2016] para o *frontend*, **Java** [JavaSE17, 1995] + **Spring Boot** [SpringBoot3.5.0, 2014] no *backend*, e **PostgreSQL** [PostgreSQL17.5, 1996] como base de dados relacional. Esta escolha tecnológica permitiu garantir uma maior manutenibilidade e uma integração mais estruturada entre entidades.

A contribuição principal deste trabalho centrou-se na implementação de funcionalidades no *frontend*, com foco na criação e melhoria de formulários, gestão de posições e atribuições, controlo de acessos, e reorganização de componentes para promover a reutilização de código. Foram também efetuadas tarefas no backend e realizadas validações rigorosas às funcionalidades desenvolvidas, recorrendo a testes manuais e a testes automatizados com o apoio da ferramenta **Selenium** [Selenium4.33.0, 2004], que permitiu simular interações reais com a aplicação e garantir o correto funcionamento do sistema em diferentes cenários de utilização.

Através da execução destas tarefas, foi possível consolidar práticas modernas de desenvolvimento de software, como a separação de responsabilidades, o uso de componentes reutilizáveis, e a organização modular do código. A avaliação das funcionalidades implementadas demonstrou melhorias na experiência de utilização e na estabilidade do sistema, contribuindo para uma base sólida para o desenvolvimento futuro da plataforma.

Abstract

This project focuses on the development of a digital solution to support the management of human and technical resources, their allocation to projects, and the tracking of business opportunities within an organizational context.

The designed tool, MMD, was developed within the scope of Accenture's CSF unit, aiming to support the leadership team in visualizing and organizing information related to available resources, ongoing projects, and new business opportunities.

The work corresponds to the evolution of a pre-existing application, MMDv1, which, despite being functional, presented significant limitations. Among the main issues identified was the use of a non-relational database (**MongoDB** [MongoDB8.0.4, 2009]), which proved inadequate for managing the complex relationships between entities such as resources, opportunities, and projects.

To address these limitations, a new version—MMDv2—was developed using a modular monolithic architecture based on the MVC pattern [MVC, 2023], and supported by widely adopted technologies: **Next.js** [Vercel, 2016] for the *frontend*, **Java** [JavaSE17, 1995] and **Spring Boot** [SpringBoot3.5.0, 2014] for the *backend*, and **PostgreSQL** [PostgreSQL17.5, 1996] as the relational database. This technological stack enables better maintainability and a more structured integration between system entities.

The main contribution of this work involved the implementation of features on the *frontend*, including the creation and improvement of forms, management of roles and assignments, access control, and the reorganization of components to promote code reuse. Additionally, contributions were made on the *backend*, and all features were validated through both manual and automated testing (using **Selenium** [Selenium4.33.0, 2004]), that allowed to simulate user interaction with the application ensuring the reliability and consistency of the implemented changes.

Through the execution of these tasks, it was possible to consolidate modern software development practices, such as separation of concerns, use of reusable components, and modular code organization. The evaluation of the implemented features showed improvements in user experience and system

stability, contributing to a solid foundation for the platform's future evolution.

Índice

Resumo	i
Abstract	iii
Índice	v
Lista de Tabelas	vii
Lista de Figuras	ix
Lista de Acrónimos	xi
1 Introdução	1
2 Trabalho Relacionado	5
3 Modelo Proposto	9
3.1 Requisitos	10
3.2 Fundamentos	11
3.2.1 Arquitetura da Aplicação	11
3.2.2 Organização do Código e Práticas Adotadas	17
3.2.3 Experiência do Utilizador e Eficiência	19
3.3 Abordagem	19
3.3.1 Tarefas Desenvolvidas	20
3.3.2 Criação de Projetos e Recursos	22
3.3.3 Edição de Projetos e Recursos	23
3.3.4 Controlo de Acessos	23
3.3.5 Erros de filtragem e pesquisa	25

4	Implementação do Modelo	27
4.1	Tecnologias e Dependências	27
4.1.1	Organização e Estrutura de Código	28
4.2	Estrutura da Aplicação	28
4.3	Implementação das Funcionalidades	29
4.3.1	Gestão de Posições, Atribuições e Previsões	29
4.3.2	Formulários Condicionais	30
4.3.3	Refatorações e Componentes Auxiliares	31
4.3.4	Otimizações de Desempenho e Gestão de Estado	31
4.3.5	Controlo de Acessos	34
4.4	Atividades de Garantia de Qualidade (QA)	34
5	Validação e Testes	37
5.1	Processo de Validação por Pull Requests	37
5.2	Testes Automatizados com Selenium	38
5.3	Validação Funcional e Colaboração	38
5.4	Testes Manuais	38
5.5	Resumo das Estratégias de Validação	40
6	Conclusões e Trabalho Futuro	41
A	Gestão de versões	43
A.1	Interligação de Repositórios e Estrutura	44
A.2	Resumo das Convenções Adotadas	45
	Bibliografia	47

Lista de Tabelas

2.1	Comparação entre OpenProject, Taiga e MMDv2	7
4.1	Principais tecnologias utilizadas	28
5.1	Resumo das estratégias de validação aplicadas	40
A.1	Convenções de nomenclatura e fluxo de trabalho	45

Lista de Figuras

3.1	Arquitetura global de alto nível do sistema MMDv2	13
3.2	Estrutura do módulo de dados (<i>Data</i>)	13
3.3	Estrutura do módulo de migração de dados (<i>Data Migration</i>) .	14
3.4	Estrutura do módulo <i>Backend</i> (Java + Spring Boot)	15
3.5	Estrutura do módulo <i>Frontend</i> (Next.js)	15
3.6	Arquitetura detalhada do padrão MVC no MMDv2	16
3.7	Campo position desativado enquanto não houver cliente	21
4.1	Campo WBS desativado quando o valor do FTE é zero ou vazio	30
4.2	Campo WBS ativado quando o valor do FTE é superior a zero	31
5.1	Estado inicial da modal	39
5.2	Todos os campos renderizados	39

Lista de Acrónimos

MMD	Manage My de Domain
CSF	Cloud Platform Solution Factory
MMDv1	Manage My de Domain Version 1
MMDv2	Manage My de Domain Version 2
MVC	Model-View-Controller
QA	Quality Assurance
FTE	Full Time Equivalent
WBS	Work Breakdown Structure
DTO	Data Transfer Object
API	Application Programming Interface
UI	User Interface
MUI	Material User Interface (UI)
PR	Pull Request
SSR	Server-Side Rendering
SSG	Static Site Generation
ISR	Incremental Static Regeneration
CI/CD	Continuous Improvent and Continous Delivery

HTTP	H yper T ext T ransfer P rotocol
REST	R epresentational S tate T ransfer
URL	U niform R esource L ocator
JSON	J ava S cript O bject N otation
ORM	O bject- R elational M apping

Capítulo 1

Introdução

A gestão eficiente de recursos humanos e técnicos, a sua afetação a projetos e o acompanhamento de oportunidades são elementos essenciais para o bom funcionamento de grandes organizações. A crescente complexidade associada a estas atividades exige soluções informáticas que promovam visibilidade, organização e apoio à tomada de decisão.

O presente projeto insere-se neste contexto, com o objetivo de desenvolver uma ferramenta de apoio à liderança da unidade CSF da empresa Accenture. A ferramenta, denominada MMD, permite não apenas visualizar, mas também gerir informação relativa a recursos, projetos e oportunidades. Entre as funcionalidades disponibilizadas destacam-se a criação, edição e remoção destas entidades, facilitando o controlo centralizado e a atualização contínua dos dados em tempo real.

A versão atualmente em uso, MMDv1, apresenta limitações significativas, sobretudo ao nível da estrutura de dados utilizada. O sistema foi originalmente construído sobre uma base de dados não relacional (**MongoDB**), que, embora tenha oferecido flexibilidade nas fases iniciais de desenvolvimento, revelou-se inadequada para um projeto com esta escala e complexidade. Este tipo de base de dados não impõe esquemas rígidos nem assegura integridade referencial entre entidades, o que dificulta a representação de relações complexas entre projetos, recursos e oportunidades. Adicionalmente, a ausência de mecanismos como *joins* ou *constraints* compromete significativamente a consistência dos dados e a integridade das relações entre entidades. Num sistema com elevada interdependência entre dados, como é o caso de posições associadas a projetos, atribuições a recursos, ou previsões de atividade, a

impossibilidade de estabelecer relações formais entre tabelas dificulta a validação automática de integridade referencial. Sem *constraints*, como *foreign keys*, o sistema não consegue garantir, por exemplo, que uma posição pertença efetivamente a um projeto existente ou que uma atribuição não aponte para um recurso já removido. Do mesmo modo, a ausência de suporte a *joins* obriga a lógica da aplicação a realizar manualmente o cruzamento de dados em múltiplas etapas, aumentando a complexidade, a propensão a erros e o esforço de manutenção à medida que o sistema cresce.

Para superar estas limitações, foi iniciada a construção de uma nova versão da aplicação, MMDv2, que introduz melhorias significativas tanto a nível funcional como arquitetural.

A nível funcional, esta nova versão introduz:

- Reorganização da estrutura de dados, permitindo uma melhor representação das relações entre entidades;
- Interface mais intuitiva para a criação e edição de projetos;
- Sistema de controlo de acessos baseado em perfis de utilizador.
- Utilização da framework **Next.js** no *frontend*, que permite funcionalidades avançadas como *routing* automático, **Server-Side Rendering** (SSR), **Static Site Generation** (SSG) e **Incremental Static Regeneration** (ISR), contribuindo para uma aplicação mais rápida, escalável e otimizada.

A nível tecnológico, a aplicação passa a adotar:

- Arquitetura modular e relacional, adequada à complexidade do domínio;
- **Next.js** (**React**) para o *frontend*;
- **Java** com **Spring Boot** para o *backend*;
- Base de dados relacional **PostgreSQL**.

Esta evolução tecnológica proporciona ganhos concretos ao nível da escalabilidade, da consistência dos dados e da facilidade de integração contínua.

Este relatório está organizado em seis capítulos. O Capítulo 2 apresenta o trabalho relacionado, enquadrando o projeto no estado da arte e identificando

os principais pressupostos teóricos e tecnológicos. O Capítulo 3 descreve o modelo conceptual adotado, incluindo os requisitos, fundamentos e a abordagem técnica seguida. O Capítulo 4 detalha a implementação prática do modelo, com foco nas decisões tecnológicas e nos componentes efetivamente desenvolvidos. O Capítulo 5 apresenta os testes realizados e os resultados obtidos, avaliando a eficácia e robustez da solução implementada. Por fim, o Capítulo 6 reúne as conclusões do trabalho, uma reflexão sobre os objetivos alcançados e propostas para desenvolvimento futuro.

As convenções de grafia adotadas ao longo do relatório visam promover clareza e consistência: o *itálico* é utilizado para destacar expressões em língua estrangeira; o **negrito** é reservado para conceitos importantes ou elementos estruturais, como nomes de capítulos e tecnologias-chave; e o **texto monoespaçado** é utilizado para nomes de campos, botões da interface e termos técnicos ou de programação.

Capítulo 2

Trabalho Relacionado

A aplicação MMDv2 insere-se na categoria de ferramentas de apoio à gestão de projetos e recursos humanos, com foco em organizações de média a grande dimensão. Esta categoria é amplamente explorada por diversas soluções, tanto comerciais como de código aberto, que procuram facilitar a alocação eficiente de recursos, o acompanhamento de oportunidades de negócio e a gestão integrada de equipas.

Uma das plataformas *open-source* mais relevantes neste domínio é o **OpenProject** [OpenProject16.0.0, 2012]. Trata-se de uma ferramenta colaborativa de gestão de projetos, desenvolvida em **Ruby on Rails**, que oferece funcionalidades como planeamento de projetos, acompanhamento de tarefas, gestão de recursos, controlo de tempo, e visualização de progresso. Tal como o MMDv2, o **OpenProject** suporta perfis de utilizador com permissões diferentes e permite a ligação entre diferentes entidades, como tarefas, equipas e prazos.

No entanto, a principal distinção do MMDv2 reside na sua orientação específica para contextos empresariais internos, como o da CSF da Accenture. Ao contrário de soluções genéricas como o **OpenProject**, o MMDv2 integra diretamente a gestão de oportunidades comerciais, a atribuição de recursos a projetos em curso e a criação de previsões de disponibilidade, proporcionando uma abordagem adaptada às necessidades reais da organização.

Outra ferramenta relevante no panorama de gestão de projetos *open-source* é o *Taiga*. Taiga é uma plataforma desenvolvida em **Python** e **Angular**, concebida para equipas ágeis que seguem metodologias como *Scrum* e *Kanban*. Oferece funcionalidades como gestão de tarefas, criação de *sprints*,

planeamento de *backlog*, e integração com ferramentas de comunicação.

Embora seja uma solução poderosa para equipas de desenvolvimento de software, o foco do Taiga é claramente orientado para o acompanhamento de tarefas e fluxos de trabalho colaborativos, não oferecendo funcionalidades específicas para a gestão de recursos empresariais ou a ligação direta com oportunidades comerciais, como acontece no MMDv2.

Importa ainda referir que, apesar da maturidade e estabilidade que plataformas como o **OpenProject** e o **Taiga** oferecem, essa mesma longevidade traduz-se frequentemente numa base tecnológica menos atual. Estas soluções foram concebidas numa altura em que práticas como *Server-Side Rendering (SSR)*, *Static Site Generation (SSG)* ou integração com pipelines de **Continuous Improvement** and **Continuous Delivery (CI/CD)** modernos ainda não estavam amplamente disseminadas. Como tal, a adoção de frameworks recentes, como o **Next.js**, no desenvolvimento do MMDv2, representa uma vantagem significativa em termos de desempenho, escalabilidade e facilidade de manutenção, ao permitir tirar partido das mais recentes inovações do ecossistema web.

Apesar de ambos os projetos apresentarem semelhanças, não oferecem as necessidades específicas para um projeto interno de uma empresa. Ainda assim, sendo um tipo de aplicação que já foi desenvolvido e estudado amplamente, o MMDv2 retira inspiração de certas funcionalidades existentes e constrói sobre essa base uma solução mais ajustada ao seu contexto de aplicação.

A Tabela 2.1 apresenta uma comparação resumida entre estas ferramentas e o MMDv2.

Tabela 2.1: Comparação entre OpenProject, Taiga e MMDv2

Critério	OpenProject	Taiga	MMDv2
Objetivo principal	Gestão colaborativa de projetos e tarefas	Gestão de projetos ágeis (Scrum, Kanban)	Gestão integrada de recursos, projetos e oportunidades
Tecnologias	Ruby on Rails, PostgreSQL	Python, Angular, PostgreSQL	Next.js, Spring Boot, PostgreSQL
Gestão de recursos	Parcial	Não suportado	Suportado com lógica avançada de atribuições
Arquitetura modular	Limitada	Parcial	Altamente modular e atualizada
Foco empresarial	Gestão de tarefas e equipas de projeto	Equipas de desenvolvimento ágil	Gestão organizacional interna (CSF)
Atribuição de recursos a projetos	Parcial	Não suportado	Total, com previsões e lógica condicional

Capítulo 3

Modelo Proposto

O modelo proposto descreve a reformulação funcional e estrutural da aplicação MMDv2, com o objetivo de melhorar a gestão de recursos, projetos e oportunidades. A versão anterior, MMDv1, baseava-se numa arquitetura de micro-serviços, com múltiplos *schemas* na base de dados, o que introduzia complexidade adicional na manutenção e interoperabilidade do sistema. Em contraste, a nova versão adota uma arquitetura modular monolítica baseada no padrão MVC, promovendo uma separação clara de responsabilidades entre camadas e facilitando a manutenção, a escalabilidade e a integração entre componentes. Esta reformulação inclui também uma experiência de utilização mais intuitiva, com foco na organização visual dos formulários, navegação mais fluida e feedback contextual ao utilizador.

O desenvolvimento da nova versão da aplicação teve como ponto de partida um conjunto de objetivos definidos pela equipa de projeto, a partir dos quais foi possível identificar, planear e executar um conjunto de tarefas técnicas que contribuíram diretamente para a concretização desses objetivos.

No início do projeto, já existiam alguns componentes estruturais e visuais da aplicação, bem como uma parte da comunicação com o *backend*. O trabalho aqui descrito inclui, entre outros contributos, intervenções que visaram a melhoria da experiência de utilização, da qualidade do código e da integração entre componentes da aplicação. Entre essas intervenções destacam-se:

- Desenvolvimento de novos módulos de interface;
- Uniformização da apresentação visual da aplicação;

- Reformulação de formulários multi-seção, com separação lógica dos campos;
- Melhoria da navegação e da acessibilidade da interface;
- Introdução de mecanismos de validação de dados no cliente;
- Reorganização e reutilização de componentes para maior eficiência e manutenibilidade;
- Propostas técnicas para a melhoria da estrutura dos dados trocados entre o *frontend* e o *backend*, facilitando o seu consumo e integração.

A estrutura do capítulo está dividida em três seções principais: em primeiro lugar, são apresentados os **requisitos** funcionais e não funcionais extraídos a partir dos objetivos definidos; de seguida, a seção de **fundamentos** detalha os princípios e práticas seguidos no desenho e desenvolvimento da aplicação; por fim, a **abordagem** descreve as decisões técnicas e exemplos ilustrativos da aplicação prática do modelo proposto.

3.1 Requisitos

Os requisitos funcionais centram-se na capacidade do sistema de permitir a criação, edição, visualização e remoção das principais entidades de negócio.

Ao longo do desenvolvimento, foram também identificadas oportunidades de melhoria não previstas inicialmente. Estas resultaram em diversas decisões técnicas, entre as quais se destacam:

- Reorganização de componentes para facilitar a sua reutilização em diferentes contextos da aplicação;
- Ajustes ao modelo de dados, com o objetivo de melhorar a interoperabilidade entre o *frontend* e o *backend*;
- Pequenas alterações na estrutura das páginas, promovendo maior clareza visual e melhor usabilidade.
- Criação de separadores para preencher informação relevante para os recursos, projetos e oportunidades.

- Atualização de pedidos **HyperText Transfer Protocol** (HTTP) para acumular alterações de *endpoints*.
- Desenvolvimento de alguns testes em **Selenium** para simular interação de utilizadores.

Estas decisões são descritas em maior detalhe na Secção 3.3.

Nota: A priorização e avaliação global dos requisitos do sistema foram conduzidas internamente pela equipa da Accenture, não estando incluídas no âmbito deste trabalho. Esta secção apresenta os requisitos diretamente relacionados com os objetivos a que este projeto contribuiu.

3.2 Fundamentos

Esta secção descreve os fundamentos que sustentam a estrutura da aplicação, com foco nas decisões relativas à arquitetura, organização do código e experiência de utilização, adotadas ao longo do desenvolvimento.

3.2.1 Arquitetura da Aplicação

O sistema MMDv2 adota uma **arquitetura modular em camadas**, composta por serviços independentes para o *frontend* e o *backend*, que comunicam entre si através de interfaces bem definidas (**Application Programming Interface** (API) **RE**presentational **S**tate **T**ransfer (REST)). Ambos os serviços partilham uma base de dados relacional única, o que garante a coerência e persistência dos dados ao longo de toda a aplicação.

Esta abordagem representa um compromisso entre simplicidade e separação de responsabilidades. Comparativamente a uma arquitetura monolítica clássica, onde todos os componentes estão integrados num único processo e não podem ser executados de forma autónoma, o MMDv2 permite maior flexibilidade: o *frontend* e o *backend* são desenvolvidos, testados e implantados separadamente, o que facilita a manutenção e a escalabilidade.

Por outro lado, o sistema não segue uma arquitetura de microserviços, onde cada componente de negócio (e.g.: recursos, iniciativas, posições, atribuições, contratos e finanças) seria isolado num serviço próprio, com base de dados e ciclo de vida independentes, como era no MMDv1. Em vez disso,

o MMDv2 mantém uma estrutura centralizada de dados, reduzindo a complexidade de comunicação entre serviços e a necessidade de orquestração, ao mesmo tempo que beneficia da separação tecnológica e lógica entre as camadas da aplicação.

Este modelo híbrido proporciona uma boa base de evolução técnica, permitindo que futuramente se explorem abordagens mais distribuídas caso a complexidade ou o volume de utilização assim o justifique.

Apesar desta separação de serviços, a camada de *backend* foi concebida segundo o padrão MVC, permitindo organizar logicamente o código em três componentes principais:

A **Model** representa as entidades de domínio e assegura a persistência dos dados na base de dados;

A **Controller** contém a lógica de negócio e é responsável por orquestrar as operações entre os dados e a apresentação;

A **View** corresponde à interface com o utilizador, ou seja, a camada de apresentação, desenvolvida com recurso à framework **Next.js**.

Esta separação de responsabilidades facilita a manutenção, escalabilidade e evolução da aplicação, permitindo que cada camada seja desenvolvida, testada e ajustada de forma independente.

Visão Global da Arquitetura

A Figura 3.1 apresenta uma visão de alto nível da arquitetura do sistema, onde são identificados os quatro módulos principais: **Data**, **Data Migration**, **Backend** e **Frontend**.

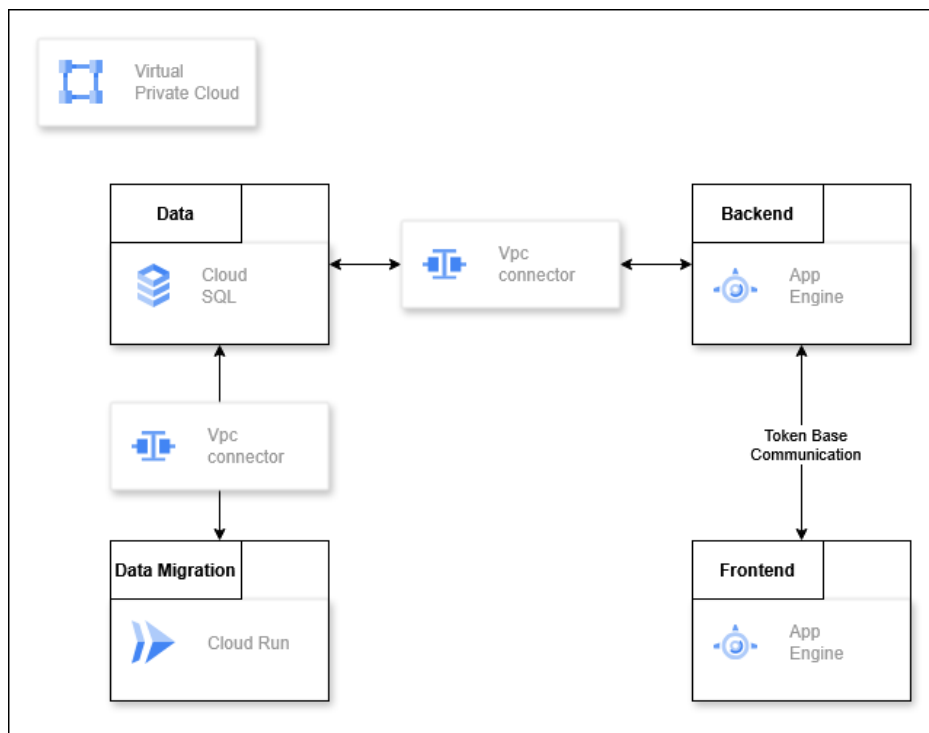


Figura 3.1: Arquitetura global de alto nível do sistema MMDv2

Para melhor ilustrar o papel e estrutura de cada módulo, foram incluídas as seguintes figuras complementares:

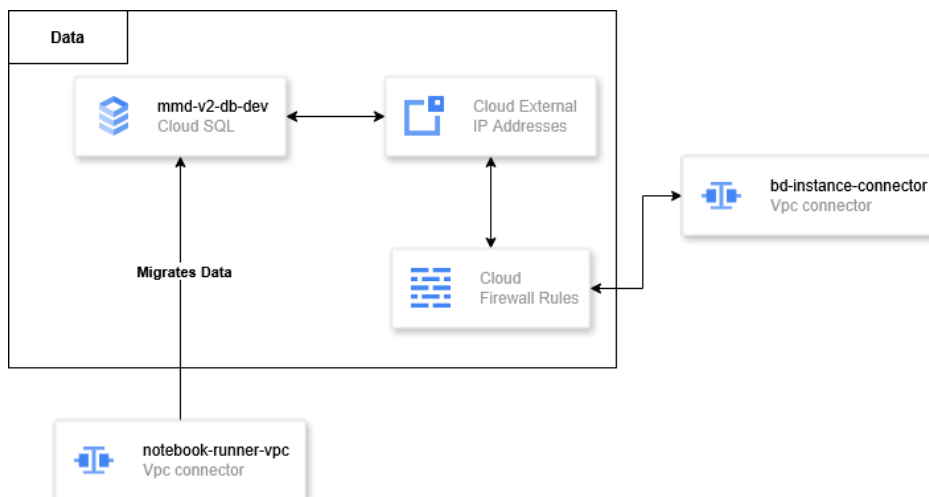


Figura 3.2: Estrutura do módulo de dados (*Data*)

A Figura 3.2 mostra a organização do módulo **Data**, responsável pela definição das entidades de negócio, das relações entre elas e pela configuração dos *repositories* que asseguram a persistência na base de dados. Este módulo define o modelo relacional da aplicação.

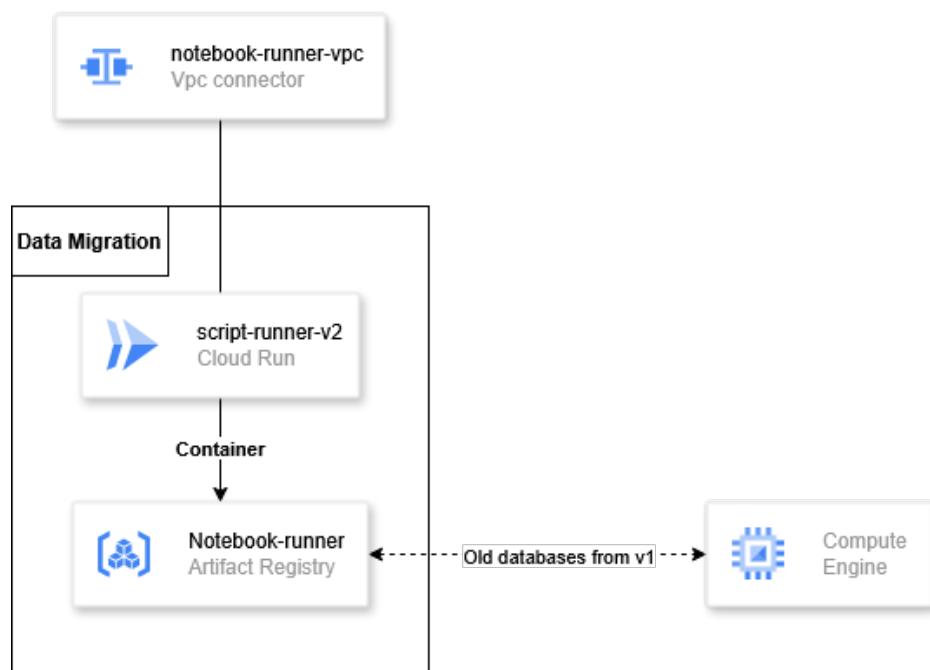


Figura 3.3: Estrutura do módulo de migração de dados (*Data Migration*)

A Figura 3.3 representa o módulo **Data Migration**, utilizado para importar dados da versão anterior da aplicação. Este módulo trata da transformação e validação dos dados a migrar, garantindo a integridade da informação.

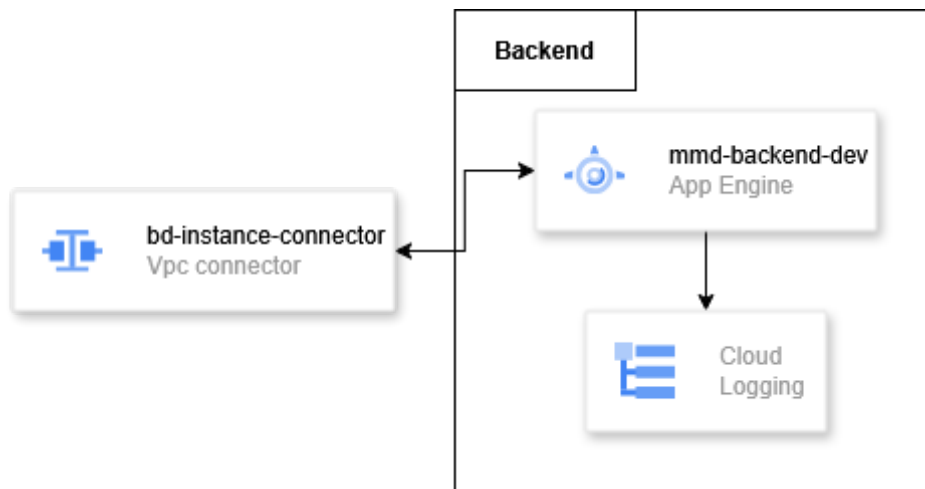


Figura 3.4: Estrutura do módulo *Backend* (Java + Spring Boot)

A Figura 3.4 ilustra o módulo **Backend**, responsável pela implementação da lógica de negócio, exposição de *endpoints* REST e gestão de autenticação, autorização e validação de dados. Este módulo serve de ponte entre o *frontend* e o sistema de base de dados. A comunicação com o *frontend* é realizada através de *tokens*.

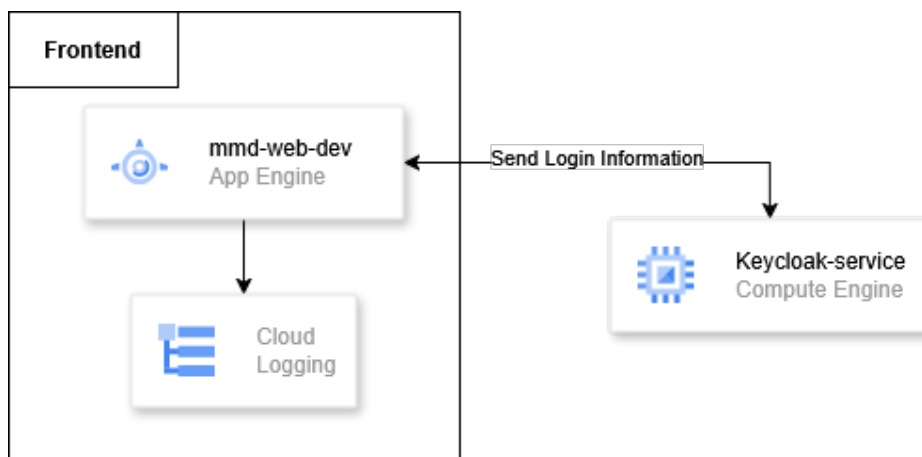


Figura 3.5: Estrutura do módulo *Frontend* (Next.js)

A Figura 3.5 apresenta o módulo **Frontend**, desenvolvido em **Next.js**, responsável pela camada de apresentação. Este módulo realiza comunicação com a API do backend e envia a informação de *login* para o serviço de

Keycloak [Keycloak26.2.5, 2014] que retorna um *token* utilizado na comunicação com o *backend*.

Padrão MVC e Fluxo de Comunicação

Para complementar a explicação anterior, a Figura 3.6 apresenta a implementação concreta do padrão MVC na camada de *backend*, destacando a interação entre os principais componentes: **Entity**, **Mapper**, **Repository**, **DTO**, **Controller**, **View** e **Database**.

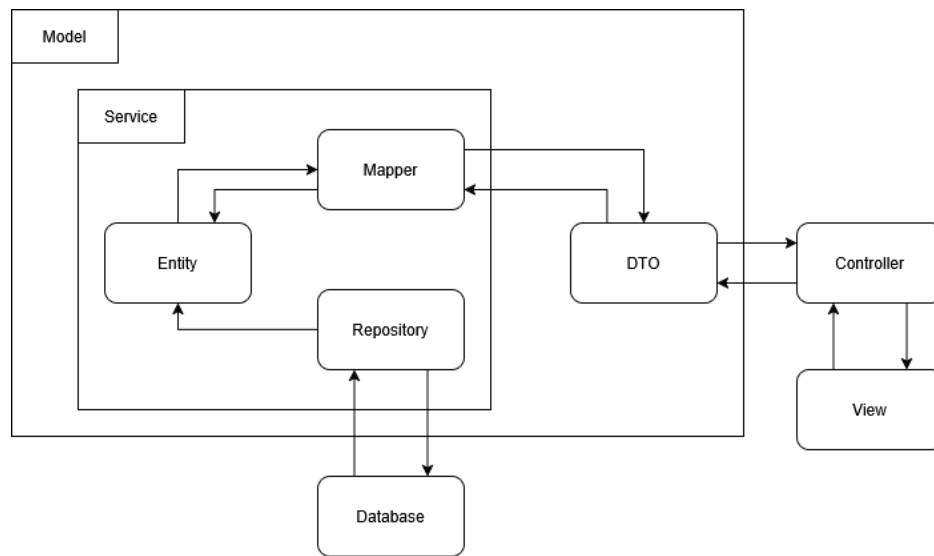


Figura 3.6: Arquitetura detalhada do padrão MVC no MMDv2

A comunicação entre os componentes segue o fluxo:

O utilizador interage com a **View** (*frontend*), que comunica com os **Controllers** através da API;

O **Controller** trata o pedido, aplica a lógica de negócio e utiliza os **Data Transfer Object (DTO)** para transportar dados;

Os **DTO** são convertidos em **Entities** e vice-versa com recurso a um **Mapper**, que garante a separação entre domínios;

As **Entities** são persistidas ou consultadas através dos **Repositories**, que abstraem o acesso direto à **Database**;

A resposta segue o fluxo inverso até ao *frontend*, onde os dados são apresentados ao utilizador.

A resposta segue o fluxo inverso até à **View**, onde os dados são apresentados ao utilizador de forma estruturada.

Este fluxo modular assegura uma separação clara de responsabilidades, promovendo a clareza, a testabilidade e a reutilização do código. Cada componente tem um papel bem definido, o que facilita a manutenção, a introdução de novas funcionalidades e a escalabilidade do sistema.

Para evitar o acoplamento entre as entidades de domínio e a estrutura de dados exposta ao exterior, a comunicação entre o *Controller* e o restante sistema é feita através de DTO, cuja conversão é assegurada por um *Mapper*. Esta camada intermédia permite adaptar os dados às necessidades específicas de cada operação, incluindo apenas os atributos relevantes, omitindo informações sensíveis ou desnecessárias e evitando a criação de campos nulos.

As *Entities* representam os modelos persistentes do domínio e são manipuladas pela camada *Repository*, que fornece métodos para acesso e modificação da base de dados. Esta organização promove uma arquitetura modular e escalável, com responsabilidades bem delimitadas ao longo de todo o ciclo de dados.

3.2.2 Organização do Código e Práticas Adotadas

No desenvolvimento do *frontend*, foi adotada uma abordagem baseada em componentes reutilizáveis, organizados em pastas por domínio funcional. Esta organização permitiu reduzir duplicação de código e facilitou a escalabilidade da interface. Foram também aplicadas convenções de nomenclatura e estruturas compatíveis com o padrão seguido pelas equipas anteriores, promovendo a consistência e reduzindo a curva de aprendizagem para novos elementos.

Ao longo do projeto, foram identificados casos de código redundante ou pouco modular. Nessas situações, procedeu-se à sua reorganização em componentes mais genéricos, com propriedades configuráveis, permitindo a sua reutilização em múltiplos contextos.

Um caso específico, foi detetada uma repetição constante de operações sobre um tipo específico de entidade, sobretudo no que diz respeito à sua representação em listas de seleção. Para resolver este problema, foi criado um novo ficheiro na pasta de utilitários, onde foram centralizadas essas operações

recorrentes. Entre elas, destaca-se a formatação dos rótulos apresentados nas listas suspensas, bem como a conversão dos objetos originais para uma estrutura adequada à sua utilização nesses componentes. Esta abordagem permitiu reduzir a duplicação de lógica, melhorar a legibilidade do código e facilitar a sua reutilização em vários pontos da aplicação.

Um caso específico de melhoria no código surgiu com a entidade *posição*, cuja importância aumentou significativamente nesta nova versão da aplicação devido à multiplicação de relações com outras entidades. Verificou-se uma repetição constante de operações associadas à manipulação desta entidade, tanto na preparação dos dados para componentes como listas suspensas, como na forma de aceder à posição dependendo da origem dos dados.

Para lidar com estas situações, foi criada uma função utilitária que centraliza a conversão de uma posição num objeto compatível com os componentes de seleção da interface. Esta função trata os diferentes formatos possíveis em que a posição pode ser recebida — seja diretamente como um objeto do tipo `UnionOptionType`, ou como um objeto proveniente de uma resposta da base de dados, e formata a `label` de acordo com o conteúdo disponível.

```
1 export const handlePositionAsOptionType = (  
2   position: StaffingPlanRow | IStaffingPlanList | null,  
3   index: number  
4 ): UnionOptionType | null => {  
5   if (!position) {  
6     return null;  
7   }  
8  
9   if (isUnionOptionType(position)) {  
10    if ('extraAttrs' in position) {  
11      return {  
12        id: index,  
13        label: getPositionExtraAttrsLabel(position.extraAttrs!, index)  
14      };  
15    }  
16    return position;  
17  }  
18  
19  return {  
20    id: index,  
21    label: getPositionLabel(position, index)  
22  };  
23  };
```

Código 1: Função para converter a posição num objeto compatível com listas

suspensas

A separação entre lógica de apresentação e lógica de interação com a API foi mantida através da criação de serviços dedicados a chamadas HTTP, com tratamento uniforme de erros e integração com os estados da aplicação (*loading*, *success*, *error*).

3.2.3 Experiência do Utilizador e Eficiência

Na camada de apresentação, foi dada especial atenção à experiência de utilização, com o objetivo de tornar a interface mais clara, responsiva e intuitiva. Foram implementadas melhorias visuais nos formulários multi-seção, assegurando *feedback* imediato através de validações em tempo real, mensagens de confirmação e indicadores de carregamento. Estas alterações visaram reduzir a frustração do utilizador e tornar a interação com o sistema mais previsível e eficiente.

A eficácia destas melhorias foi validada com o envolvimento de utilizadores chave da organização que, não tendo participado no desenvolvimento, forneceram uma perspetiva imparcial sobre a usabilidade da aplicação.

No que respeita ao desempenho, foram implementadas estratégias de otimização como a eliminação de chamadas redundantes à API, evitando pedidos desnecessários e melhorando a fluidez da navegação. Esta abordagem permitiu reduzir significativamente a carga sobre o servidor e acelerar os tempos de resposta da aplicação, contribuindo para uma experiência global mais satisfatória.

3.3 Abordagem

Esta secção descreve a abordagem prática seguida na concretização dos objetivos definidos para o projeto MMDv2. Com base nos fundamentos técnicos apresentados na Secção 3.2, as decisões tomadas ao longo do desenvolvimento procuraram garantir coerência com a arquitetura adotada, a qualidade do código produzido, e a evolução sustentável da aplicação.

Complementarmente, foram também realizadas tarefas ao nível do *backend*, nomeadamente no ajuste de endpoints e na adaptação de dados recebidos pelo cliente, bem como atividades de **Quality Assurance (QA)**, através

da verificação de funcionalidades implementadas, identificação de anomalias e colaboração na validação de correções, os vários testes foram realizados através do **Selenium** [Selenium4.33.0, 2004] que permite simular interações reais com a aplicação.

Sempre que possível, foram introduzidas melhorias técnicas em áreas como a reutilização de código, a validação de dados no cliente, a clareza das mensagens de erro, e a simplificação do fluxo de utilização.

3.3.1 Tarefas Desenvolvidas

Ao longo do projeto, foram desenvolvidas diversas tarefas, que incluíram tanto a implementação de novas funcionalidades como a resolução de problemas e refatoração de componentes existentes. Estas tarefas estão agrupadas por natureza da intervenção:

Funcionalidades Novas

- **Separadores e janelas modais para criação de posições em projetos e oportunidades** — Cada separador apresenta uma tabela com as posições já existentes, filtros relevantes e um botão para criação de novas entradas. A janela modal inclui os vários campos que podem ou devem ser preenchidos, sendo esses campos compostos por texto livre, listas suspensas ou seletores de datas.
- **Separadores e janelas modais para previsões (*forecasts*) num recurso** — Esta funcionalidade segue a mesma estrutura e lógica da tarefa anterior, com a criação de um separador dedicado à gestão de previsões associadas a um recurso. O separador apresenta uma tabela com previsões existentes, filtros aplicáveis e um botão para adicionar novas previsões. A respetiva janela modal mantém o mesmo aspeto visual e funcionalidade, com campos de entrada que incluem texto livre, listas suspensas e seletores de datas.
- **Separador para introdução de informações complementares numa oportunidade** — Este separador apresenta os vários campos que podem ser preenchidos com as várias informações de uma oportunidade.

- **Lógica de criação automática de posições a partir de atribuições e previsões** — Garante a coerência entre entidades interdependentes, evitando omissões. Aqui foi necessário adicionar ao formulário que será submetido ao *backend*, também um objeto representante de uma posição quando se adiciona uma atribuição ou uma previsão.

Refatorações e Melhorias

- **Reformulação de modais dependentes de contexto** — Em alguns modais foi realizada uma alteração na lógica de visualização para apenas renderizar os campos todos quando um primeiro campo fosse preenchido. Também existem alguns campos que ficam desativados dependendo do valor de outros.
e.g.: enquanto não for escolhido um cliente, não é possível escolher uma posição



Figura 3.7: Campo position desativado enquanto não houver cliente

- **Ligação entre campos relacionados (e.g.: roleId de atribuição e de custo)** — Evita preenchimento redundante, visto que na maioria das vezes a atribuição e o custo da atribuição vão apresentar o mesmo valor para esse campo.
- **Centralização de operações em ficheiros utilitários** — Criação de funções reutilizáveis para transformar objetos em estruturas adequadas para listas suspensas e para formatar rótulos apresentados. e.g.: Uma função que permite extrair os dados importantes tanto dos objetos quando estes são criados no momento ou quando estes são obtidos através de pedidos ao backend.
- **Adaptação da transformação dos dados recebidos da API** — O formato de alguns dados foi alterado no backend e foi necessário adaptar como eram utilizados no frontend. e.g.: As atribuições deixaram de ter uma lista de perfis e passaram a ter apenas um perfil.

Correções de Problemas

- **Libertação automática de posições quando uma atribuição é encerrada.**
- **Correção na remoção de atribuições** — As posições associadas voltam a ficar disponíveis.
- **Revisão da barra de pesquisa** — Corrige filtros obsoletos.
- **Correção na eliminação de entidades** — Resolução de erros devido a referências a nomes desatualizados.

3.3.2 Criação de Projetos e Recursos

Uma das funcionalidades desenvolvidas foi a interface para a criação de novos projetos e novos recursos. Esta funcionalidade envolveu:

- A construção de um formulário dinâmico, dividido por separadores, com o objetivo de agrupar campos relacionados, como os referentes a informações gerais ou a posições;
- A realização de pedidos à API para obter os dados necessários ao preenchimento de *dropdowns*;
- A apresentação de mensagens de erro contextuais e validações específicas em cada secção do formulário.

A implementação mais complexa neste formulário foi a lógica de interação entre as entidades posições, atribuições e previsões. Estas entidades estão intimamente relacionadas: tanto as atribuições como as previsões exigem a associação a uma posição, e as previsões podem ser posteriormente convertidas em atribuições. Esta proximidade semântica traduz-se numa estrutura de dados muito semelhante entre as atribuições e as previsões, com distinções em alguns campos e estados disponíveis.

Estas entidades são utilizadas tanto no contexto de projetos como no de recursos. No entanto, um recurso pode agregar previsões e atribuições associadas a múltiplos projetos, enquanto um projeto apenas apresenta previsões e atribuições no seu próprio âmbito.

Para garantir a estabilidade do preenchimento do formulário e evitar a criação de entidades com valores implícitos em falta, determinadas secções ou campos eram desativados até que os campos necessários em outros separadores fossem preenchidos. Por exemplo, a criação de qualquer entidade dentro de um projeto depende previamente da seleção de um cliente no separador de informação.

Foram reutilizados componentes já existentes sempre que possível, seguindo-se as convenções de estilo e estrutura estabelecidas no projeto, de modo a manter a coerência visual e funcional. A implementação foi apoiada por uma comunicação constante com a equipa técnica, assegurando a compatibilidade dos dados trocados entre o cliente e o servidor.

3.3.3 Edição de Projetos e Recursos

A edição de recursos reutiliza as mesmas páginas e janelas modais utilizadas na criação, mas preenche automaticamente os campos com os valores obtidos da base de dados. Adicionalmente, restringe a alteração de certos campos que apenas podem ser definidos no momento da criação.

Apesar de partilharem os mesmos componentes, as operações de criação e edição são acedidas através de **Uniform Resource Locator** (URL) distintas — por exemplo, `projects/create` para criação e `projects/[id]/edit` para edição. Para distinguir entre os dois modos, foi desenvolvido um *custom hook* com recurso à biblioteca **Zustand** [Zustand5.0.5, 2019], que mantém o estado da interface. Este *hook* define se a modal deve estar visível, qual o tipo de ação em curso, o índice da entidade a ser editada e disponibiliza os métodos necessários para atualizar esses valores, seguindo o padrão de *getters* e *setters*.

Determinados campos nas modais são desativados no modo de edição, sendo apenas editáveis no momento da criação. e.g.: a posição associada a uma atribuição só pode ser selecionada no processo de criação da mesma.

3.3.4 Controlo de Acessos

Foi também implementada uma lógica de controlo de acessos baseada em perfis de utilizador, garantindo que cada perfil apenas tem acesso às funcionalidades que lhe são autorizadas. A implementação consistiu em determinar

o perfil do utilizador autenticado e, com base nas regras definidas num objeto **JavaScript Object Notation** (JSON) denominado **AUTH_RULES**, controlar os privilégios de visualização e edição em diferentes partes da aplicação.

O objeto **AUTH_RULES** está organizado em cinco categorias principais:

- **acesso** — define os níveis de permissão necessários para aceder a cada página da aplicação;
- **ações** — especifica os requisitos de acesso para executar diferentes operações, como criar, editar ou remover entidades;
- **recursos** — regula o acesso aos campos individuais no modo de edição de recursos;
- **projetos** — define as permissões sobre os campos editáveis dos projetos;
- **oportunidades** — controla o acesso a campos específicos dentro das oportunidades.

Inicialmente foi considerado aplicar, no *frontend*, a lógica de filtragem dos projetos, recursos e oportunidades que o utilizador poderia visualizar, bem como as atribuições, previsões e posições que poderia editar. No entanto, para simplificar a implementação e aumentar a segurança, foi decidido deslocar essa lógica para o *backend*. Assim, o *frontend* apenas recebe os dados que o utilizador tem efetivamente permissão para visualizar, e cada entidade, como atribuições, previsões ou posições, inclui um campo que indica explicitamente se pode ou não ser editada.

Esta abordagem reforça a segurança do sistema, garantindo que os utilizadores não têm acesso a entidades para as quais não estão autorizados, e contribui para uma maior eficiência ao reduzir o número de chamadas à API necessárias para realizar verificações de permissões no lado do cliente.

Adicionalmente, existe também lógica de controlo de acessos implementada no *backend*, assegurando que os dados expostos ao cliente respeitam os privilégios definidos e que não era possível aceder a conteúdos através do URL. Contudo, essa vertente não foi desenvolvida no âmbito deste projeto e não é aqui abordada em detalhe.

3.3.5 Erros de filtragem e pesquisa

Devido às várias alterações realizadas no modelo de dados ao longo do projeto, foi necessário corrigir diversos erros associados aos mecanismos de filtragem e pesquisa nas listas de projetos e oportunidades.

A filtragem é executada no *backend*, através de uma função que utiliza os parâmetros do URL para aplicar pesquisas sobre todos os campos ou filtrações específicas por campo (e.g.: `projects?search=john` para pesquisa livre ou `projects?name=john` para filtragem específica).

Para isso, é utilizado o método `like` do objeto `CriteriaBuilder`, fornecido pela API **Jakarta Persistence** [JakartaPersistence3.2, 2024]. Esta API permite realizar operações de persistência de dados relacionais através de mapeamento objeto-relacional (**Object-Relational Mapping (ORM)**), e o `CriteriaBuilder` disponibiliza ainda métodos como `or` e `and` para construir expressões compostas e cláusulas de consulta flexíveis.

As alterações no modelo de dados afetaram nomes de campos e a estrutura interna das entidades. Por exemplo, o campo `profile`, que anteriormente era uma lista de perfis, passou a ser representado por uma única *string*. Essas mudanças tornaram obsoletas algumas das expressões de filtragem e pesquisa, que continuavam a tentar aceder a estruturas antigas ou comparar valores de forma incorreta.

Como consequência, foi necessário rever os campos utilizados nos filtros e nas pesquisas, identificar quais sofreram alterações estruturais ou semânticas, e adaptar a lógica de construção das consultas para refletir corretamente a nova estrutura dos dados.

Capítulo 4

Implementação do Modelo

Este capítulo descreve a concretização prática das propostas apresentadas no Capítulo 3, com foco nos aspectos técnicos da implementação da solução MMDv2. São aqui detalhadas as tecnologias utilizadas, a estrutura da aplicação, a organização do código e a implementação efetiva das principais funcionalidades desenvolvidas no âmbito do projeto.

4.1 Tecnologias e Dependências

A aplicação MMDv2 foi desenvolvida utilizando um conjunto de tecnologias amplamente adotadas no mercado e alinhadas com os objetivos de manutenibilidade, escalabilidade e integração contínua. A Tabela 4.1 apresenta um resumo das principais tecnologias utilizadas, bem como a respectiva justificação.

Tabela 4.1: Principais tecnologias utilizadas

Tecnologia	Justificação
Next.js (React)	Framework moderna baseada em React, com suporte nativo a renderização no servidor (SSR) e ótima integração com TypeScript. Permite construção de interfaces modulares e performantes.
Java + Spring Boot	Solução robusta para o backend, com forte suporte a APIs REST, gestão de dependências e modularidade.
PostgreSQL	Base de dados relacional, ideal para manter integridade de dados e representar relações complexas entre entidades.
Selenium	Ferramenta de automação para testes funcionais, utilizada na verificação de cenários de utilização críticos.
Git	Sistema de controlo de versões utilizado em colaboração com a equipa.

4.1.1 Organização e Estrutura de Código

Durante o desenvolvimento, foi mantida uma preocupação constante com a organização e reutilização do código. Foram seguidas convenções de nomeação e estrutura de pastas já existentes, promovendo a integração com os restantes módulos da aplicação.

Foram também criados componentes reutilizáveis, funções auxiliares e melhorias estruturais, como a refatoração de formulários e a centralização de transformações de dados. Estas práticas facilitaram a manutenção, aumentaram a legibilidade e reduziram redundâncias, contribuindo para um sistema mais estável e sustentável.

4.2 Estrutura da Aplicação

A aplicação segue uma arquitetura de tipo monolítica, organizada segundo o padrão MVC, conforme descrito na Secção 3.2 e na figura 3.6. A estrutura está dividida em três grandes áreas:

- **Frontend:** implementado em Next.js, com uma divisão por domínios

funcionais. Cada módulo contém os seus próprios componentes, páginas, estilos e lógica de apresentação. De forma a acelerar o desenvolvimento e garantir a uniformidade visual da aplicação, foi adotada a biblioteca de componentes **Material UI (MUI)**. Sempre que necessário, foram criados componentes personalizados baseados em MUI, permitindo entender o seu comportamento e adaptá-los às necessidades específicas do sistema.

- **Backend:** desenvolvido em Java com Spring Boot, disponibilizando endpoints REST organizados por recursos.
- **Base de dados:** estruturada em PostgreSQL, com tabelas normalizadas e ligações definidas por chaves estrangeiras.

As comunicações entre frontend e backend são feitas por meio de chamadas HTTP autenticadas através, com o envio de dados em formato JSON. A autenticação baseia-se em tokens armazenados no cliente após o login inicial.

4.3 Implementação das Funcionalidades

As principais funcionalidades descritas no Capítulo 3 foram concretizadas com base em componentes reutilizáveis, serviços de comunicação com a API, e formulários dinâmicos. Exemplos concretos incluem:

4.3.1 Gestão de Posições, Atribuições e Previsões

Foram desenvolvidos módulos específicos para permitir a criação e visualização de posições, atribuições e previsões associadas a projetos, recursos e oportunidades.

Adicionalmente, foi implementada a lógica que permite, ao submeter uma nova atribuição ou previsão, criar automaticamente a posição correspondente. Inversamente, ao criar uma nova posição com determinados campos preenchidos, como o recurso associado, o tipo e o estado da posição, pode também ser automaticamente criada uma atribuição ou uma previsão, consoante os valores introduzidos. Esta funcionalidade assegura a coerência do modelo de dados, evitando estados inconsistentes e simplificando o fluxo de criação de entidades relacionadas.

As atribuições, previsões e posições são entidades interligadas no modelo de dados, uma vez que as atribuições e previsões partilham a maior parte dos seus atributos e ambas têm sempre uma posição associada. Como a página de criação e edição de projetos, tem cinco separadores distintos, é possível realizar múltiplas criações e alterações dos seus dados. De forma a reduzir o número de comunicações com a base de dados, estas alterações só são submetidas quando o utilizador aciona explicitamente a operação de guardar. Assim, a lógica de criação cruzada entre posições, atribuições e previsões tem de ser tratada diretamente no *frontend*. Para tal, sempre que uma das entidades é criada com os campos necessários preenchidos, a lista das entidades correspondentes é automaticamente atualizada com a nova informação. Por exemplo, ao selecionar um recurso numa posição e definir o respetivo tipo e estado, pode ser criada automaticamente uma atribuição ou previsão correspondente.

4.3.2 Formulários Condicionais

Alguns formulários foram reestruturados para renderizar campos dinamicamente com base na informação previamente preenchida. Esta lógica condicional permite melhorar a experiência de utilização, evitando interações inválidas ou desnecessárias.

Um exemplo concreto ocorre no formulário de criação de atribuições (*assignments*), onde a seleção de um valor para o campo **Full Time Equivalent** (FTE) determina o estado de outros campos dependentes, como o campo **Work Breakdown Structure** (WBS). Caso o valor do FTE seja zero, o campo WBS é automaticamente desativado, impedindo que o utilizador preencha uma informação que não será considerada válida.

As Figuras 4.1 e 4.2 ilustram este comportamento condicional. Na primeira imagem, o campo WBS aparece desativado após a seleção de FTE com valor nulo. Na segunda, o campo é ativado quando o FTE assume um valor positivo.



Figura 4.1: Campo WBS desativado quando o valor do FTE é zero ou vazio

Seat Charge FTE (%)
10

Seat Charge WBS

Figura 4.2: Campo WBS ativado quando o valor do FTE é superior a zero

4.3.3 Refatorações e Componentes Auxiliares

Com o objetivo de melhorar a legibilidade e promover a reutilização de código, foram extraídas diversas funções auxiliares dedicadas à transformação de dados e à construção de *dropdowns*, posteriormente centralizadas em ficheiros de utilitários. Esta abordagem permitiu reduzir a duplicação de lógica e facilitar a manutenção futura da aplicação.

Um dos exemplos mais significativos desta melhoria foi a criação de um componente reutilizável para a criação de atribuições. Dado que as atribuições podiam ser criadas tanto a partir do seu separador específico como no momento em que uma previsão era atribuída, este componente evitou a necessidade de duplicar a mesma lógica de interface em diferentes locais da aplicação.

4.3.4 Otimizações de Desempenho e Gestão de Estado

No desenvolvimento do *frontend*, foi dada especial atenção ao desempenho e à estabilidade da aplicação durante interações frequentes com o utilizador. Uma das estratégias adotadas passou pela reutilização de dados obtidos em chamadas à API, evitando pedidos redundantes sempre que possível. Esta abordagem revelou-se particularmente importante em formulários com separadores dinâmicos, onde múltiplas *dropdowns* são frequentemente atualizadas.

Contudo, durante a implementação dessa lógica de reutilização, foi identificado um problema de concorrência na gestão de estado. Este surgia quando o utilizador alternava rapidamente entre separadores antes da conclusão da primeira chamada de obtenção de dados.

A obtenção dos dados para as *dropdowns* era realizada através da função `fetchOptions`, incluída na estrutura de estado gerada pela função `createFetchState`. Após a obtenção dos dados, era atualizado um booleano no estado para indicar que os dados já tinham sido carregados. No entanto, como esta função é assíncrona, em casos de alternância rápida entre separadores que partilham

os mesmos dados, poderia ocorrer uma chamada à função `fetchOptions` quando esta já não se encontrava disponível no objeto de estado. Isto devia-se ao facto de, após a primeira atualização, o objeto de estado passar a conter apenas os atributos `options` e `loading`, omitindo a própria referência à função `fetchOptions`, o que originava erros em tempo de execução.

Abaixo apresenta-se a versão original da função, onde se verifica que a função `fetchOptions` apenas era definida na criação inicial do estado:

```

1 export const createFetchState =
2   (key: string, endpoint: string, getEnterpriseId: boolean = false) =>
3   (set: (state: Partial<DropdownCreateResourceStore>) => void): FetchState
4     => ({
5       options: [],
6       loading: false,
7       fetchOptions: async () => {
8         set({ [key]: { loading: true, options: [] } });
9         try {
10           const response = await get(endpoint, null);
11           set({
12             [key]: {
13               options: transformDropdownContent(response?.content,
14               getEnterpriseId),
15               loading: false
16             }
17           });
18         } catch (error) {
19           console.error(error);
20           set({ [key]: { loading: false, options: [] } });
21         }
22       }
23     });

```

Código 2: Versão original da função `createFetchState` sem preservação de `fetchOptions`

Para resolver este problema, a função foi reestruturada de forma a preservar a referência à função `fetchOptions` em todas as atualizações de estado. A nova versão, apresentada na Listagem 3, garante que o método está sempre presente no objeto, mesmo após a conclusão de chamadas assíncronas:

```

1 export const createFetchState =
2   (key: string, endpoint: string, getEnterpriseId: boolean = false) =>
3   (set: (state: Partial<DropdownCreateResourceStore>) => void): FetchState
4     => {
5       const fetchOptions = async (accessToken) => {
6         set({

```

```
7         loading: true,
8         options: [],
9         fetchOptions
10    }
11  });
12  try {
13    const response = await get(endpoint, null, accessToken);
14    set({
15      [key]: {
16        loading: false,
17        options: transformDropdownContent(response?.content,
18        getEnterpriseId),
19        fetchOptions
20      }
21    });
22  } catch (error) {
23    console.error(error);
24    set({
25      [key]: {
26        loading: false,
27        options: [],
28        fetchOptions
29      }
30    });
31  };
32
33  return {
34    options: [],
35    loading: false,
36    fetchOptions
37  };
38  };
```

Código 3: Função `createFetchState` com preservação de `fetchOptions`

Esta alteração assegura que a função `fetchOptions` é mantida em todas as transições de estado, evitando o seu desaparecimento em cenários de navegação rápida entre separadores. Uma alternativa viável para mitigar este problema seria a inclusão de um indicador de chamada em curso, evitando execuções redundantes da função enquanto uma chamada ainda não foi concluída.

4.3.5 Controlo de Acessos

Como detalhado na Subsecção 3.3.4, o controlo de acessos foi implementado com base em perfis de utilizador, através de um objeto `AUTH_RULES` que define, no *frontend*, as permissões de visualização e edição para diferentes entidades e ações da aplicação. Esta lógica garante que cada utilizador interage apenas com os elementos para os quais possui autorização. Para reforçar a segurança e simplificar a implementação, a filtragem de dados visíveis foi delegada ao *backend*, que expõe unicamente os dados permitidos. Embora o controlo de acessos no servidor tenha sido aplicado nos controladores, a sua implementação concreta não integrou o âmbito deste projeto.

4.4 Atividades de Garantia de Qualidade (QA)

A validação técnica das funcionalidades desenvolvidas foi suportada por atividades de garantia de qualidade, nomeadamente através de testes manuais e testes automatizados realizados com a ferramenta **Selenium**. Foram testados cenários de inserção, edição e eliminação de dados, bem como fluxos completos com múltiplas interações entre componentes. As anomalias identificadas durante os testes foram reportadas, corrigidas e subsequentemente verificadas.

Os *scripts* de teste seguem uma lógica estruturada, recorrendo a vários métodos e classes disponibilizados pelo **Selenium**, como:

- `WebDriverWait`, para definir tempos máximos de espera;

- `ExpectedConditions`, que fornece métodos para verificar condições como a visibilidade ou a interatividade de elementos;

- `until`, que aplica essas condições dentro do tempo definido.

No excerto de código 4 pode ser observado como localizar um botão, clicar nele e preencher um campo:

```
1 driver.get(BASE_URL + "resources");
2 wait = new WebDriverWait(driver, Duration.ofSeconds(15));
3 var editButton = wait.until(
4 ExpectedConditions.visibilityOfElementLocated(By.id("resources-edit-button")
5 );
6 editButton.click();
7
8 var eIdField = wait.until(
```



```
9 ExpectedConditions.visibilityOfElementLocated(By.id("resource-info-eid"))
10 );
11 eIdField.click();
12 var eIdValue = env.getProperty("EID_INPUT");
13 eIdField.sendKeys(eIdValue);
```

Código 4: Excerto de código para testes com Selenium

O objeto **driver** corresponde a uma instância da classe **WebDriver**, que permite controlar diferentes *user agents* (*browsers*) . Este objeto é utilizado para navegar pelas páginas, aguardar condições específicas, interagir com elementos da interface e validar resultados.

A localização dos elementos pode ser feita através de identificadores únicos (**id**) ou expressões **XPath**, especialmente quando os elementos têm identificadores dinâmicos ou gerados automaticamente (e.g.: os elementos para selecionar datas (**DatePicker** [DatePicker8.5.2,]) tinham **id** gerados automaticamente). Os testes são alimentados por conjuntos de dados de entrada (*inputs*) e validações baseadas em dados esperados (*outputs*), permitindo verificar se o comportamento da aplicação é consistente com os requisitos definidos.

Capítulo 5

Validação e Testes

A validação e os testes desempenharam um papel fundamental ao longo do desenvolvimento do MMDv2, assegurando a robustez, correção funcional e estabilidade da aplicação. Dado que o projeto decorreu em colaboração com uma empresa (Accenture), todos os contributos de código seguiram um processo formal de revisão e validação, alinhado com as práticas profissionais de engenharia de software.

5.1 Processo de Validação por Pull Requests

Todas as alterações implementadas, quer se tratassem de novas funcionalidades, refatorações ou correções de erros, foram integradas no repositório central através de **Pull Request** (PR). Cada PR era sujeito a uma revisão manual obrigatória por parte de elementos da equipa técnica da Accenture, que avaliavam o código proposto com base em critérios de qualidade, alinhamento com a arquitetura do sistema e impacto funcional.

Além disso, a submissão de um PR exigia a apresentação de evidência visual do funcionamento correto das alterações que necessitava da realização de testes manuais, que vão ser explicados em mais detalhe na secção 5.4. Esta evidência incluía capturas de ecrã que ilustrassem o comportamento esperado da funcionalidade ou a resolução do problema identificado. Esta prática permitiu acelerar o processo de validação e reduzir a ocorrência de regressões.

5.2 Testes Automatizados com Selenium

Complementarmente à revisão manual, foram também utilizados testes funcionais automatizados com a ferramenta **Selenium**. Esta abordagem permitiu simular interações reais do utilizador com a aplicação e validar fluxos completos de utilização.

Os testes criados com Selenium focaram-se principalmente nas áreas mais sensíveis da aplicação, como a criação de entidades, navegação entre separadores, submissão de formulários e verificação de mensagens de erro. A automatização destes testes contribuiu para uma deteção precoce de anomalias, especialmente após atualizações ao código partilhado por múltiplas funcionalidades.

5.3 Validação Funcional e Colaboração

Durante todo o projeto, a validação funcional foi realizada em estreita colaboração com a equipa técnica da empresa. Cada funcionalidade entregue era testada em ambiente de desenvolvimento partilhado, sendo os resultados discutidos em sessões de demonstração e feedback. Estas sessões garantiram o alinhamento com os requisitos previamente definidos (ver Capítulo 3) e permitiram a identificação de melhorias incrementais, algumas das quais foram implementadas no próprio ciclo de desenvolvimento.

5.4 Testes Manuais

Para além dos testes automatizados com Selenium e da validação por pull requests, foi realizada uma componente significativa de testes manuais durante o desenvolvimento e integração das funcionalidades. Estes testes tinham como principal objetivo verificar o correto funcionamento das interfaces e fluxos críticos da aplicação em cenários reais de utilização, garantindo uma experiência consistente e livre de erros para o utilizador final.

Os testes manuais foram especialmente importantes em funcionalidades com elevada variabilidade de comportamento consoante o perfil do utilizador ou o tipo de entidade em causa, como por exemplo:

- Criação e edição de projetos, recursos, oportunidades e atribuições;

- Visualização e filtragem de listas em separadores dinâmicos;
- Navegação entre formulários multi-secção com validações condicionais;
- Comportamento de campos dependentes (ex: ativação/desativação automática);
- Respostas visuais a falhas de comunicação ou dados inválidos.

Os testes manuais foram realizados em ambiente de desenvolvimento, utilizando dados simulados e perfis de utilizador com diferentes permissões. Sempre que uma anomalia era identificada, esta era registada num sistema interno de rastreio de problemas (*Azure DevOps*).



Figura 5.1: Estado inicial da modal

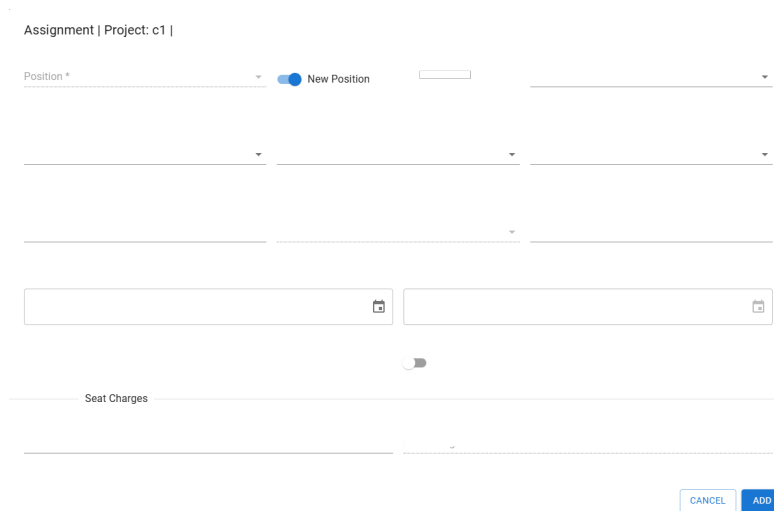


Figura 5.2: Todos os campos renderizados

A Figura 5.1 e a Figura 5.2 mostra um exemplo de verificação visual durante testes manuais, onde se avalia o comportamento dinâmico de campos

numa modal, pode ser observado que quando se preenche o campo *Position* ou se ativa o botão *New Position* os campos todos são renderizados.

Os testes manuais complementaram os restantes mecanismos de validação, permitindo testar situações complexas, com dependência de múltiplos fatores que, por vezes, não estavam cobertos pelos testes automatizados. Foram também essenciais para a validação da componente visual e da experiência de utilização.

5.5 Resumo das Estratégias de Validação

Tabela 5.1: Resumo das estratégias de validação aplicadas

Método	Descrição
Revisão por PR	Validação manual obrigatória por parte da equipa da Accenture.
Evidência visual	Capturas de ecrã enviados com o PR.
Testes com Selenium	Testes funcionais automatizados para simulação de interações reais.
Testes Manuais	Testes realizados ao longo do desenvolvimento das funcionalidades implementadas.
Sessões de feedback	Validação funcional colaborativa com demonstrações práticas.

Capítulo 6

Conclusões e Trabalho Futuro

O desenvolvimento do projeto MMDv2, integrado no contexto de um projeto em cooperação com a Accenture, constituiu uma oportunidade valiosa para aplicar e aprofundar conhecimentos técnicos em desenvolvimento de software numa situação real, com objetivos concretos e impacto direto numa ferramenta em uso pela organização.

Ao longo deste trabalho, foi possível contribuir ativamente para a evolução de uma aplicação de apoio à gestão de recursos, projetos e oportunidades, com particular foco na camada de apresentação (*frontend*) e, complementarmente, em tarefas de *backend* e validação funcional. A intervenção abrangeu tanto a implementação de novas funcionalidades como a identificação e resolução de problemas, reestruturação de componentes existentes, e introdução de melhorias técnicas visando a reutilização, clareza e estabilidade do sistema.

Foram seguidos princípios de engenharia de software modernos, como a separação de responsabilidades, modularização, reaproveitamento de código e validação contínua através de revisões de *pull requests* e testes automatizados com Selenium. Estas práticas não só asseguraram a qualidade das entregas realizadas como também reforçaram a escalabilidade e manutenção futura do sistema.

O projeto destacou-se também por incluir funcionalidades pouco comuns em plataformas genéricas de gestão de projetos, como a interação entre entidades empresariais (projetos, recursos e oportunidades). Esta abordagem resolve os problemas específicos, quando comparada com soluções open-source existentes como o OpenProject ou o Taiga permite uma maior customização

e adaptação às necessidades, mas baseada em princípios e tecnologias amplamente adotados no setor.

Trabalho Futuro

Embora o estágio curricular tenha permitido uma participação significativa na evolução da plataforma, o projeto encontra-se em desenvolvimento contínuo. Durante o próximo mês, irei manter a minha colaboração com a equipa responsável, com o objetivo de concluir funcionalidades em aberto, consolidar melhorias implementadas e apoiar na integração de novas componentes da aplicação.

Além disso, será benéfico, num futuro próximo, considerar a introdução de testes automatizados mais abrangentes no frontend, incluindo testes de regressão para componentes reutilizáveis, e a eventual refatoração de fluxos complexos com base no feedback dos utilizadores.

Este projeto demonstrou o valor de aplicar práticas sólidas de engenharia como a reutilização e clareza do código escrito num contexto real e deixou uma base estruturada que poderá servir como referência ou ponto de partida para futuros projetos de natureza semelhante, sejam eles internos a empresas ou em ambiente académico.

Apêndice A

Gestão de versões

Durante o desenvolvimento do projeto MMDv2, a gestão de versões foi realizada através da plataforma **Azure DevOps**, utilizando um sistema de controlo de versões baseado em Git. O principal objetivo desta abordagem foi garantir uma organização clara do código, permitir o desenvolvimento paralelo de funcionalidades e minimizar conflitos entre alterações realizadas por diferentes membros da equipa.

A estratégia adotada baseou-se na criação de *branches* específicos por tarefa. Sempre que se iniciava o desenvolvimento de uma nova funcionalidade, era criado um ramo com o prefixo **feature/**, seguido do identificador numérico da tarefa e de uma breve descrição. Para a resolução de problemas ou correções de código existente, era utilizado o prefixo **bug/**. Esta convenção permitia identificar rapidamente o propósito de cada ramo e a tarefa correspondente.

- **Exemplo de ramo para funcionalidade nova:**
`feature/3199-projects-role-access-information-tab`
- **Exemplo de ramo para correção de erro:**
`bug/2477-projects-create-delete-assignments`

Paralelamente, foi também adotada uma convenção nas mensagens de *commit*, que incluíam o número da tarefa e uma descrição sucinta da alteração efetuada. Esta prática facilitou a rastreabilidade no histórico do projeto, tanto no editor de código como na interface da plataforma de controlo de versões.

O fluxo de trabalho seguia um modelo padronizado: cada tarefa era desenvolvida num ramo isolado, e a sua conclusão originava a criação de um *pull request* para integração no ramo principal do repositório. Antes da fusão, o código era sujeito a uma revisão obrigatória por outro membro da equipa e à verificação do correto funcionamento das alterações introduzidas, com base em evidência funcional.

A.1 Interligação de Repositórios e Estrutura

O projeto MMDv2 foi desenvolvido numa arquitetura de repositórios múltiplos, todos alojados no Azure DevOps. Cada área funcional do sistema, *frontend*, *backend* e *QA*, corresponde a um repositório independente, o que facilita a especialização das equipas, a separação de responsabilidades e a gestão de permissões de acesso.

Apesar da separação física entre repositórios, foi mantida uma estrutura coerente em termos de organização interna e convenções de desenvolvimento. Cada repositório incluía pipelines próprios de integração e entrega contínuas (CI/CD), bem como scripts de suporte e documentação interna adaptada às necessidades da respetiva área.

Esta abordagem modular permitiu escalar o desenvolvimento da aplicação, garantir maior manutenibilidade e minimizar o impacto de alterações localizadas, mantendo ao mesmo tempo a integração global entre os componentes da solução.

A.2 Resumo das Convenções Adotadas

Tabela A.1: Convenções de nomenclatura e fluxo de trabalho

Elemento	Descrição
Prefixo <code>feature/</code>	Funcionalidades novas (ex: <code>feature/3199-projects-role-access-information</code>)
Prefixo <code>bug/</code>	Correção de problemas em funcionalidades existentes (ex: <code>bug/2477-projects-create-delete-assignments</code>)
Commits	Incluem número da tarefa e descrição concisa (ex: <code>[feature/MMD-3199] FE Projects Role Access - Information Tab</code>)
Pull Requests	Sujeitos a revisão por outro membro da equipa e acompanhados por evidência funcional (imagens)
Repositórios	Estrutura distribuída: repositórios distintos para <i>frontend</i> , <i>backend</i> e <i>QA</i> , todos geridos no <i>Azure DevOps</i>

Bibliografia

[MVC, 2023] (2023). Mvc, model-view-controller. <https://pt.wikipedia.org/wiki/MVC>.

[DatePicker8.5.2,] DatePicker8.5.2.

[JakartaPersistence3.2, 2024] JakartaPersistence3.2 (2024). Jakarta persistence layer. <https://jakarta.ee/specifications/persistence/2.2/apidocs/>.

[JavaSE17, 1995] JavaSE17 (1995). Java platform, standard edition. <https://docs.oracle.com/en/java/javase/17/>.

[Keycloak26.2.5, 2014] Keycloak26.2.5 (2014). Keycloak. <https://www.keycloak.org/documentation>.

[MongoDB8.0.4, 2009] MongoDB8.0.4 (2009). Mongodb. <https://www.mongodb.com/docs/manual/>.

[OpenProject16.0.0, 2012] OpenProject16.0.0 (2012). Open project. <https://www.openproject.org>.

[PostgreSQL17.5, 1996] PostgreSQL17.5 (1996). Postgresql. <https://www.postgresql.org/docs/>.

[Selenium4.33.0, 2004] Selenium4.33.0 (2004). Selenium. <https://www.selenium.dev/documentation/>.

[SpringBoot3.5.0, 2014] SpringBoot3.5.0 (2014). Spring boot. <https://docs.spring.io/spring-boot/index.html>.

[Vercel, 2016] Vercel (2016). Next.js. <https://nextjs.org/docs>.

[Zustand5.0.5, 2019] Zustand5.0.5 (2019). Zustand state management library. <https://zustand.docs.pmnd.rs/getting-started/introduction>.