INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Plataforma de Gestão Documental

André Miguel Pedro Pires

Relatório de estágio apresentado à Escola Superior de Tecnologia e Gestão para obtenção do grau de mestre em Informática

Orientadores:

Prof. Doutor José Eduardo Moreira Fernandes

Dr. Pedro Parreiras

Bragança

Outubro 2024

Agradecimentos

Gostaria de começar por agradecer a todos os colegas e professores com quem tive o privilégio de conviver ao longo do mestrado em Informática de 2022/2023 e 2023/2024.

De forma especial, gostaria de expressar a minha gratidão ao meu orientador do Instituto Politécnico de Bragança, o Professor José Eduardo Moreira Fernandes, pela sua paciência, disponibilidade e orientação durante este processo. Agradeço igualmente ao meu orientador de estágio na Megatic, o Dr. Pedro Parreiras, pelo acompanhamento dedicado, suporte técnico e pelos conselhos valiosos que me ajudaram a evoluir ao longo do estágio.

Aproveito também para deixar um sincero agradecimento a todos os meus colegas da Megatic, que me acolheram com grande cordialidade e disposição para ajudar. A colaboração, troca de ideias e o ambiente de trabalho proporcionado durante estes longos meses foram fundamentais para o sucesso do meu estágio.

Resumo

Este relatório de estágio compromete-se com a demonstração dos aspetos mais relevantes dos trabalhos realizados no estágio.

O projeto desenvolvido neste estágio curricular foi uma plataforma web abrangente para gestão empresarial, visando facilitar e otimizar os processos internos das empresas. Este projeto já tinha requisitos, funcionalidades e objetivos definidos para serem implementados e desenvolvidos.

As tecnologias foram apresentadas pela empresa, sendo utilizado a *framework Laravel*, como linguagem de programação *PHP* para a *backend* e *React DevExtreme* para a *frontend*.

O estágio teve a duração de sete meses e ao longo destes foram abordadas as várias etapas do desenvolvimento de um projeto web.

No final deste estágio o projeto foi concluído com sucesso.

Palavras-chave: Estágio, Gestão, *Laravel*, *PHP*, *React DevExtreme*

Abstract

This internship report is committed to demonstrating the most relevant aspects of the work carried out during the internship.

The project developed during this internship was a comprehensive web platform for business management, aimed at facilitating and optimising companies' internal processes. This project already had defined requirements, functionalities and objectives to be implemented and developed.

The technologies were presented by the company, using the Laravel framework as the PHP programming language for the backend and React DevExtreme for the frontend.

The internship lasted seven months and the various stages of developing a web project were covered throughout.

At the end of the internship, the project was successfully completed.

Keywords: Internship, Management Laravel, PHP, React DevExtreme

Índice Geral

Lista de Siglas/Abreviaturas

CRUD - *Create, Read, Update, and Delete onsulte*
IDE - Integrated Development Environment
MVC - Model-View-Controller
PHP - Hypertext Preprocessor
SQL - *Structured Query Language*

Índice de Figuras

1. Introdução

Neste capítulo, apresenta-se o tema geral abordado neste relatório de estágio estando dividido em quatro seções: objetivos, enquadramento, considerações sobre confidencialidade e organização do relatório.

Na seção de objetivos, pretende-se detalhar as metas estabelecidas para este relatório de estágio, destacando o que se pretende alcançar com o desenvolvimento e a implementação do projeto.

Em enquadramento, serão abordadas as diretrizes gerais do estágio, contextualizando os trabalhos desenvolvidos e a sua importância dentro do âmbito da empresa e da área de atuação.

Na subdivisão considerações sobre confidencialidade, é exposto o fato de existirem acordos de confidencialidade estabelecidos com a empresa na partilha de algumas informações específicas e detalhadas, consideradas sensíveis e de propriedade intelectual.

Por fim, na seção de organização do relatório, foi abordada a estrutura do relatório, detalhando como ele está organizado, incluindo os capítulos e os respetivos conteúdos.

1.1. Objetivo

O objetivo deste relatório de estágio é detalhar os trabalhos realizados e os conhecimentos adquiridos durante o período de estágio. Os objetivos específicos incluem a apresentação

da empresa, a descrição das tecnologias utilizadas, a análise do projeto desenvolvido e as conclusões obtidas.

No que se refere à empresa, o relatório pretende fornecer uma visão abrangente sobre sua história e suas metodologias de trabalho. Quando se discute as tecnologias, o objetivo é descrever detalhadamente as ferramentas e tecnologias empregadas no estágio, explicando suas funcionalidades, finalidades e como elas se integraram ao projeto desenvolvido.

Em relação ao projeto desenvolvido durante o estágio consiste na criação de uma plataforma web abrangente para a gestão empresarial, cujo objetivo principal é facilitar e otimizar os processos internos das empresas, promovendo maior eficiência operacional. A plataforma foi concebida para oferecer uma solução completa, simplificando a gestão de utilizadores, documentos, tarefas e coordenação de atividades empresariais.

Nesse sentido, a plataforma utiliza *PHP* com a *framework Laravel* escolhido principalmente pela sua capacidade de agilizar o desenvolvimento e garantir a escalabilidade do sistema (Laravel, 2024). Para a camada de *frontend*, foi utilizada a biblioteca *React* em conjunto com o *DevExtreme* (DevExtreme, 2024).

1.2. Enquadramento

Este relatório de estágio é elaborado no contexto de um projeto desenvolvido na empresa *Megatic*, uma organização reconhecida por sua inovação e excelência em soluções tecnológicas (Megatic, 2024). Durante o estágio, foi identificada a necessidade de criar uma plataforma web abrangente para gestão empresarial, com o objetivo de facilitar e otimizar os processos internos das empresas.

Este projeto surge em resposta à demanda crescente por ferramentas que não apenas automatizem tarefas rotineiras, mas também melhorem a eficiência operacional e a coordenação interna nas empresas.

Neste contexto, o estágio proporcionou uma oportunidade valiosa para aplicar conhecimentos teóricos num ambiente prático e real, contribuindo significativamente para o desenvolvimento profissional e a aquisição de novas competências técnicas.

1.3. Considerações sobre confidencialidade

Devido a acordos de confidencialidade estabelecidos com a empresa para a qual foi desenvolvida a plataforma web durante este estágio, algumas informações específicas e detalhadas sobre o projeto não podem ser reveladas neste relatório. Essas restrições visam proteger dados sensíveis e a propriedade intelectual da empresa.

Exemplos de algumas informações que poderão não ser partilhadas incluem:

- Código-fonte: Detalhes específicos do código-fonte considerado mais importante e sensível não serão divulgados. Isso inclui scripts, funções e algoritmos proprietários desenvolvidos exclusivamente para a empresa.
- Estrutura da base de dados: A arquitetura detalhada da base de dados, incluindo a estrutura das tabelas, relacionamentos entre dados e consultas SQL personalizadas, não pode ser revelada.

Apesar dessas restrições, o relatório visa proporcionar uma visão abrangente e clara do projeto. Todos os aspetos relevantes e permitidos do desenvolvimento da plataforma web serão descritos o mais detalhadamente possível.

1.4. Organização do Relatório

De maneira a proporcionar uma compreensão concisa e clara do relatório relativo ao estágio realizado, este foi dividido em diferentes capítulos. Sendo eles:

- Capítulo 1: Introdução - Apresenta o contexto do projeto, os objetivos, considerações sobre confidencialidade e a estrutura do relatório.
- Capítulo 2: A Empresa - Fornece uma visão geral da empresa onde o projeto foi realizado, incluindo sua apresentação e a metodologia de trabalho adotada.
- Capítulo 3: Plataformas de Gestão Documental - Destaca a importância das plataformas de gestão documental no ambiente empresarial atual, apresenta alguns exemplos já existentes deste tipo de plataformas e por último, descreve as

principais ferramentas e tecnologias usadas durante o desenvolvimento da plataforma web.

- Capítulo 4: O Projeto - Apresenta o projeto em si, detalhando o levantamento de requisitos, modelação e a planificação do trabalho.
- Capítulo 5: Desenvolvimento - Detalha o desenvolvimento do projeto, dividindo-o em *Backend* e *Frontend*, e descrevendo cada etapa e componentes desenvolvidos.
- Capítulo 6: Conclusões - Oferece uma análise final do projeto, incluindo reflexões sobre a empresa e as considerações finais sobre o trabalho realizado.

2. A Empresa

Neste capítulo pretende-se apresentar a empresa onde foi realizado o estágio. Este capítulo está estruturado em introdução, apresentação da empresa e metodologias de trabalho. A introdução tem como objetivo fornecer uma visão geral da história da empresa. Na apresentação da empresa, será descrito o setor em que a empresa atua e a sua situação atual. É igualmente importante destacar as metodologias de trabalho que a empresa adotou e que são praticadas no seu dia a dia.

2.1. Introdução

A empresa onde foi realizado o meu estágio tem por nome *Megatic* (Megatic, 2024), fundada em 2015 por um casal de Brigantinos está sediada no edifício *Brigantia Ecopark* (Brigantia Ecopark, 2024).

A *Megatic* tem como objetivo fornecer soluções e serviços especializados na área das tecnologias de informação e comunicação, tendo sempre em foco o objetivo final e a qualidade prestada. Acreditando que os clientes são parceiros e absorvendo as suas necessidades como se fossem as deles, implementando uma solução que responda não apenas às necessidades, mas também que seja segura e escalável. Integrações entre plataformas, desenvolvimento web, aplicações mobile e instalação/manutenção de redes informáticas, são apenas alguns dos nossos serviços (Megatic, 2024).

Esta empresa foi fundada em 2015 por um casal de Brigantinos, Pedro e Flávia, que nasceram, viveram e estudaram na cidade de Bragança. Após concluírem os estudos, tiveram a oportunidade de trabalhar na área do Grande Porto. Apesar da estabilidade nos seus empregos, sempre sonharam em retornar às suas origens e criar oportunidades na região de Bragança. Em 2020, decidiram que era o momento de concretizar esse sonho e voltaram para Bragança, dedicando-se em tempo integral à *Megatic* (Megatic, 2024).

2.2. Apresentação da Empresa

A *Megatic* atua no setor de tecnologias de informação e comunicação, oferecendo uma vasta gama de serviços que incluem integrações entre plataformas, desenvolvimento web, aplicações mobile, e instalação e manutenção de redes informáticas. Atualmente, a empresa foca-se em proporcionar soluções seguras e escaláveis que atendam às necessidades dos seus clientes, que são tratados como parceiros (Megatic, 2024).

Desde a sua fundação, a *Megatic* tem sido guiada por valores de dedicação, ética, integridade, inovação, ambição e rigor. Em meio a desafios como a pandemia, a empresa adaptou-se e investiu em marketing digital, procurando sempre aprender e implementar novas ideias para melhorar os seus serviços e ampliar sua base de clientes. Hoje, a *Megatic* não só oferece desenvolvimento de software e consultoria, mas também promove *workshops* e serviços pós-venda para garantir que os clientes alcancem os melhores resultados possíveis. A dedicação contínua à melhoria e à inovação posiciona a *Megatic* como um parceiro confiável no mercado de tecnologias de informação e comunicação (Megatic, 2024).

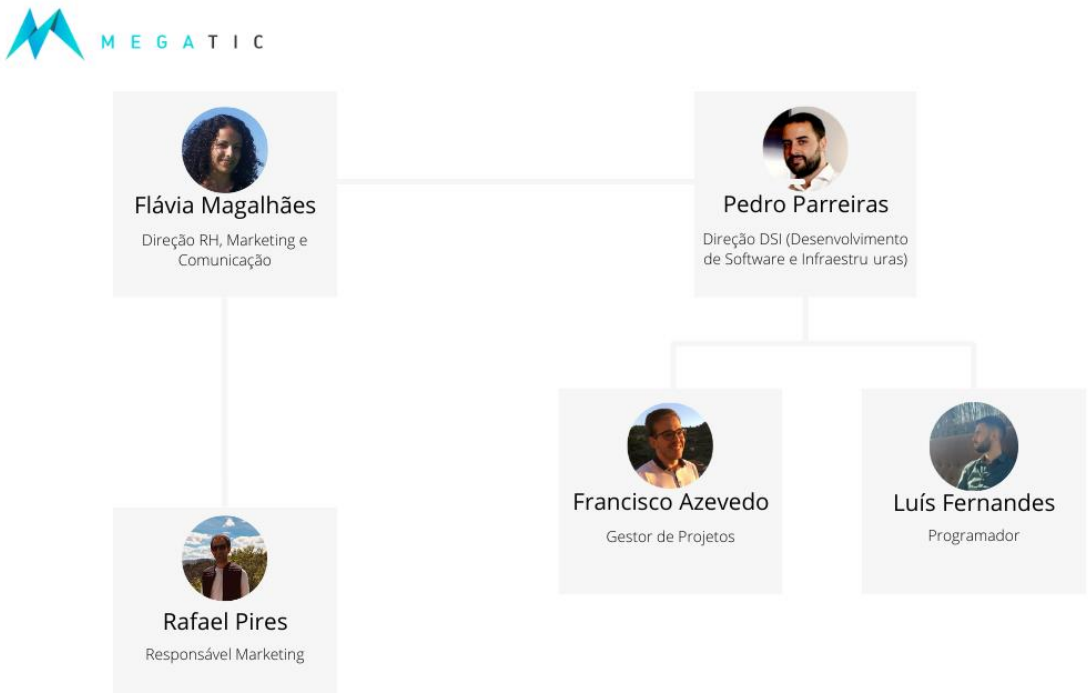


Figura 1 - Constituição da Empresa

2.3. A Metodologia de Trabalho

Nesta seção, serão apresentadas as metodologias de trabalho adotadas pela *Megatic* ao longo do meu período de estágio, detalhando os processos e as práticas que orientam as suas operações diárias.

A *Megatic* utiliza a plataforma *Microsoft Teams* (Microsoft Teams , 2024) como uma ferramenta central para a gestão de tarefas e comunicação da equipa. Esta prática facilita a organização do trabalho e melhora a colaboração entre os membros da equipa.

- Atribuição de Tarefas: As tarefas são atribuídas através do *Teams*, onde cada tarefa recebe um responsável e uma descrição detalhada do que precisa ser feito.
- Definição de Datas de Conclusão: Cada tarefa atribuída possui uma data de conclusão definida, ajudando a equipa a gerir o tempo de execução.
- Descrição e Prioridade das Tarefas: As tarefas são descritas detalhadamente e categorizadas por prioridade (alta, média, baixa), permitindo que a equipa se foque no que é mais importante.

- Monitoramento do Progresso: O progresso das tarefas é monitorizado através do Teams, com atualizações regulares e notificações automáticas sobre prazos e mudanças no status das tarefas.
- Comunicação e Colaboração: A plataforma permite uma comunicação eficiente através de *chats*, chamadas de vídeo e compartilhamento de arquivos, integrando todos os aspetos do trabalho em um único lugar.

3. Plataformas de Gestão Documental

Neste capítulo, foi discutida a importância da gestão documental no ambiente empresarial moderno, apresentando algumas plataformas de gestão documental existentes na atualidade destacando as suas funcionalidades. Além disso, foram introduzidas as principais ferramentas e tecnologias utilizadas no decorrer do estágio.

3.1. Introdução à Gestão Documental

No mundo empresarial atual, a quantidade de informação gerada diariamente é imensa. Para lidar com esse volume de dados, as empresas precisam de ferramentas que garantam a organização, acessibilidade e segurança das informações.

Assim, as plataformas de gestão documental emergem como soluções cruciais, permitindo às empresas gerirem de forma eficiente os seus documentos e otimizarem processos internos permitindo o armazenamento, organização, controle de acesso e gestão de documentos de maneira centralizada e automatizada.

Nesse sentido, uma das principais vantagens das plataformas de gestão documental é a centralização de todos os documentos em um único sistema, permitindo que várias pessoas acedam simultaneamente aos mesmos ficheiros. Em vez de informações espalhadas por vários departamentos ou armazenadas em diferentes formatos, estas

plataformas garantem que todos os documentos sejam organizados, categorizados e armazenados em um local central. Isso não só facilita a interação e a comunicação entre as equipas, como também reduz o tempo gasto na procura de documentos, aumentando a produtividade e promovendo a colaboração eficiente, mesmo em ambientes de trabalho híbridos ou remotos.

Além disso, outro benefício significativo é a automação de processos administrativos. As plataformas de gestão documental permitem a criação de fluxos de trabalho automatizados, como a aprovação de documentos, notificações e lembretes, o que reduz a necessidade de intervenção manual e minimiza erros humanos.

Em suma, as plataformas de gestão documental são ferramentas indispensáveis no mundo empresarial moderno. Ao centralizar e automatizar a gestão de documentos, essas plataformas não só melhoram a organização e a segurança das informações, mas também promovem a eficiência e a colaboração entre as equipas, independentemente do local de trabalho.

3.2. Estado de Arte

O mercado oferece uma ampla gama de plataformas de gestão documental, cada uma com características e funcionalidades específicas que atendem a diferentes necessidades empresariais utilizadas para otimizar a gestão de documentos.

A *M-Files* é uma plataforma de gestão de informações empresariais que se destaca por sua capacidade de organizar, gerir e proteger documentos e outros tipos de conteúdo em um ambiente corporativo. Ela utiliza uma abordagem baseada em metadados, o que significa que, em vez de organizar documentos em pastas tradicionais, o M-Files classifica e localiza informações com base em atributos específicos como data, tipo de documento, cliente, entre outros. Isso facilita a recuperação de informações e garante que os utilizadores sempre adiram a versão mais atualizada dos documentos (M-Files, 2024).

Além disso, uma outra plataforma amplamente utilizada no mercado é a *CTAIMA* (CTAIMA, 2024), que auxilia as empresas a manterem-se em conformidade com as normativas legais e regulatórias relacionadas à saúde, segurança e meio ambiente, contribuindo para criar organizações mais seguras e responsáveis. A *CTAIMA* abrange

diversas funcionalidades, incluindo a gestão de documentação, auditorias, inspeções e requisitos legais, garantindo que a empresa atenda a todas as suas obrigações regulatórias. Além disso, a plataforma facilita a gestão de fornecedores e contratados, assegurando que todos os parceiros externos também estejam em conformidade com as políticas e regulamentações da empresa (CTAIMA, 2024).

3.3. Apresentação das ferramentas/tecnologias

Durante o estágio, foram utilizadas diversas ferramentas e tecnologias voltadas para o desenvolvimento da plataforma web. De salientar que a empresa forneceu essas ferramentas, alinhando-as com suas metodologias e processos estabelecidos para garantir a execução eficiente das tarefas previstas.

As ferramentas e tecnologias utilizadas foram:

- *React*
- *Laravel*
- *PHP*
- *JavaScript*
- *DevExtreme React*
- *Postman*
- *Visual Studio Code*
- *phpMyadmin*
- *MySQL Workbench*
- *MySQL*
- *SQL*
- *Git*
- *GitLab*

O *React* é uma biblioteca *JavaScript* de código aberto, originalmente desenvolvida pelo Facebook e atualmente mantida por uma extensa comunidade de desenvolvedores e várias empresas. Desde o seu lançamento em 2013, esta biblioteca tem-se destacado no desenvolvimento de interfaces de utilizadores (UIs) para aplicações web, graças à sua abordagem eficaz e inovadora na criação de componentes interativos e reutilizáveis (React, 2024).

O *Laravel* é um *framework* para desenvolvimento web em *PHP*, reconhecido pela sua elegância, simplicidade e eficácia. Criado por *Taylor Otwell* e disponibilizado pela primeira vez em 2011, o *Laravel* conquistou rapidamente uma posição de destaque entre os desenvolvedores *PHP* devido à sua facilidade de utilização e às suas funcionalidades robustas (Laravel, 2024).

O *Postman* é uma ferramenta amplamente adotada por desenvolvedores de software para realizar testes, construir e documentar APIs (Interfaces de Programação de Aplicações). O Postman tem com principal função simplificar a criação de solicitações HTTP e a visualização das respostas correspondentes de maneira ágil e eficaz (Postman, 2024).

O *Visual Studio Code* é um IDE desenvolvido pela *Microsoft*. Este contém várias funcionalidades como *debug*, suporte do *git*, complementação inteligente de código, entre outras, que ajudam um programador a desenvolver melhor o seu código (Visual Studio Code, 2024).

O *phpMyAdmin* é uma ferramenta online que facilita a administração da base de dados *MySQL* e *MariaDB*. Com uma interface amigável, os utilizadores podem realizar uma série de tarefas, como criar bases de dados e tabelas, executar consultas para recuperar informações específicas, e gerir as permissões de acesso dos utilizadores à base de dados. Essa ferramenta simplifica bastante a administração, tornando-a acessível mesmo para utilizadores com menos experiência técnica (PhpMyAdmin, 2024).

O *MySQL Workbench* é uma ferramenta de design da base de dados visual que integra desenvolvimento *SQL*, administração, design da base de dados, criação e manutenção em um único ambiente de desenvolvimento integrado para o sistema da base de dados *MySQL* (MySQL Workbench, 2024)

O *Git* é um sistema de controle de versões distribuído, amplamente utilizado no desenvolvimento de software. Ele permite que os desenvolvedores acompanhem as

mudanças no código-fonte de um projeto ao longo do tempo, facilitando a colaboração em equipa e a gestão de diferentes versões do *software* (Git, 2024)

Por outro lado, o *GitLab* é uma plataforma de gestão de repositórios *Git*, oferecendo recursos adicionais para o trabalho colaborativo. Inclui funcionalidades como controle de acesso, rastreamento de problemas, integração contínua e implantação automatizada. Ao hospedar o *GitLab* em um servidor interno da empresa, é possível ter controle total sobre os repositórios e garantir a segurança dos dados sensíveis (GitLab, 2024).

4. O Projeto

Neste capítulo do relatório de estágio, apresento o projeto desenvolvido, identificação de funcionalidades e criação da estrutura do projeto e por último a planificação do trabalho ao longo deste.

4.1. Apresentação do projeto

O projeto consiste no desenvolvimento de uma plataforma web abrangente para gestão empresarial, visando facilitar e otimizar os processos internos das empresas. Utilizando tecnologias modernas como *PHP* com *Laravel* para a *backend* e *React DevExtreme* para a *frontend*, a plataforma visa fornecer uma experiência ao utilizador intuitiva e eficiente.

Assim, o principal objetivo da nossa plataforma é fornecer uma solução completa para a gestão empresarial, abrangendo desde a gestão de utilizadores até a manipulação de documentos e tarefas de coordenação. Procurando simplificar e automatizar tarefas rotineiras, permitindo que as empresas se foquem nas suas atividades principais

4.2. Levantamento de requisitos e modelação

Para criar este projeto, primeiro identificou-se as funcionalidades essenciais e os perfis dos profissionais necessários. Além disso, foi estabelecida a imagem do projeto desde o início, visando transmitir clareza e profissionalismo.

Assim, o projeto é composto por três atores, são eles:

- 'Admin': Este ator representa o administrador da plataforma, provavelmente uma pessoa com elevado nível de acesso e responsabilidade sobre a gestão global do sistema.
- 'Coorporate': Refere-se ao funcionário da empresa para a qual a plataforma está a ser desenvolvida. Essas pessoas são responsáveis por gerir processos no dia a dia da empresa, tendo um papel ativo na utilização da plataforma.
- 'Client': Este ator representa o cliente final da empresa, ou seja, o indivíduo ou organização externa que interage com a plataforma de forma limitada. O cliente utiliza a plataforma exclusivamente para a criação de um 'coordination' através de um formulário apropriado ('WebForm').

Desta forma a plataforma web deve permitir:

- Visualizar, criar, editar e eliminar Utilizadores;
- Visualizar, criar, editar e eliminar permissões associadas a cada 'Role';
- Visualizar, criar, editar e eliminar 'Priorities';
- Visualizar, criar, editar e eliminar 'Tags';
- Visualizar, criar, editar e eliminar 'Tasks';
- Visualizar, criar, editar e eliminar 'Task Types';
- Visualizar, criar, editar e eliminar 'Elements Type';
- Visualizar, criar, editar e eliminar 'Organization Type';
- Visualizar, criar, editar e eliminar 'Countries/Provinces';
- Interface para edição de configurações globais da plataforma;
- Visualizar, criar, editar e eliminar 'Issue Causes';

- Visualizar, criar, editar e eliminar ‘Work Places’;
- Visualizar, criar, editar e eliminar ‘Type Of Works’;
- Visualizar, criar, editar e eliminar ‘Client Status’;
- Visualização e gestão de “logs” globais do sistema;
- Visualizar, criar, editar e eliminar ‘Coordinations’;
- Visualizar, criar, editar e eliminar ‘Organization Contacts’;
- Visualizar, criar, editar e eliminar ‘Organization Relation;
- Visualizar, criar, editar e eliminar ‘Organizations’;
- Visualizar, criar, editar e eliminar “Follow-ups”;
- Visualizar, criar, editar e eliminar ‘Elements’;
- Visualizar, criar, editar e eliminar ‘External Url’;
- Menu de para gerar Relatórios;

4.3. Diagrama de Casos de Uso

Na Figura 2 o diagrama de casos de uso é composto por três atores principais: ‘Admin’, ‘Coorporate’ e ‘Client’. Cada ator interage com o sistema de formas diferentes, sendo o administrador aquele que tem maior acesso a todas as funcionalidades da plataforma web.

Nesse sentido, o ‘admin’ é o ator principal no diagrama, podendo gerir utilizadores, gerir permissões, operações Create, Read, Update, Delete (CRUD) em várias entidades, como ‘Work Places’, ‘Client Status’, ‘TypeOfWorks’, ‘OrganizationType’, ‘ElementType’, entre outras. Além disso, através da representação gráfica de herança, o 'Admin' possui ainda todas as permissões do ator ‘Coorporate'.

Por outro lado, o ator designado por 'Coorporate' tem interações mais restritas, focadas em operações CRUD de entidades como ‘Coordination’, ‘Organization’, ‘ExternalUrl’, ‘Element’, ‘IssueCause’, ‘Tasks’, ‘Followups’, entre outras. Pode também visualizar e alterar o seu perfil.

Por último, o ator 'Client' é o que possui menos permissões, tendo apenas acesso a uma página específica, chamada 'web_form', que permite a criação de uma 'Coordination' sem a necessidade de ter feito ‘login’.

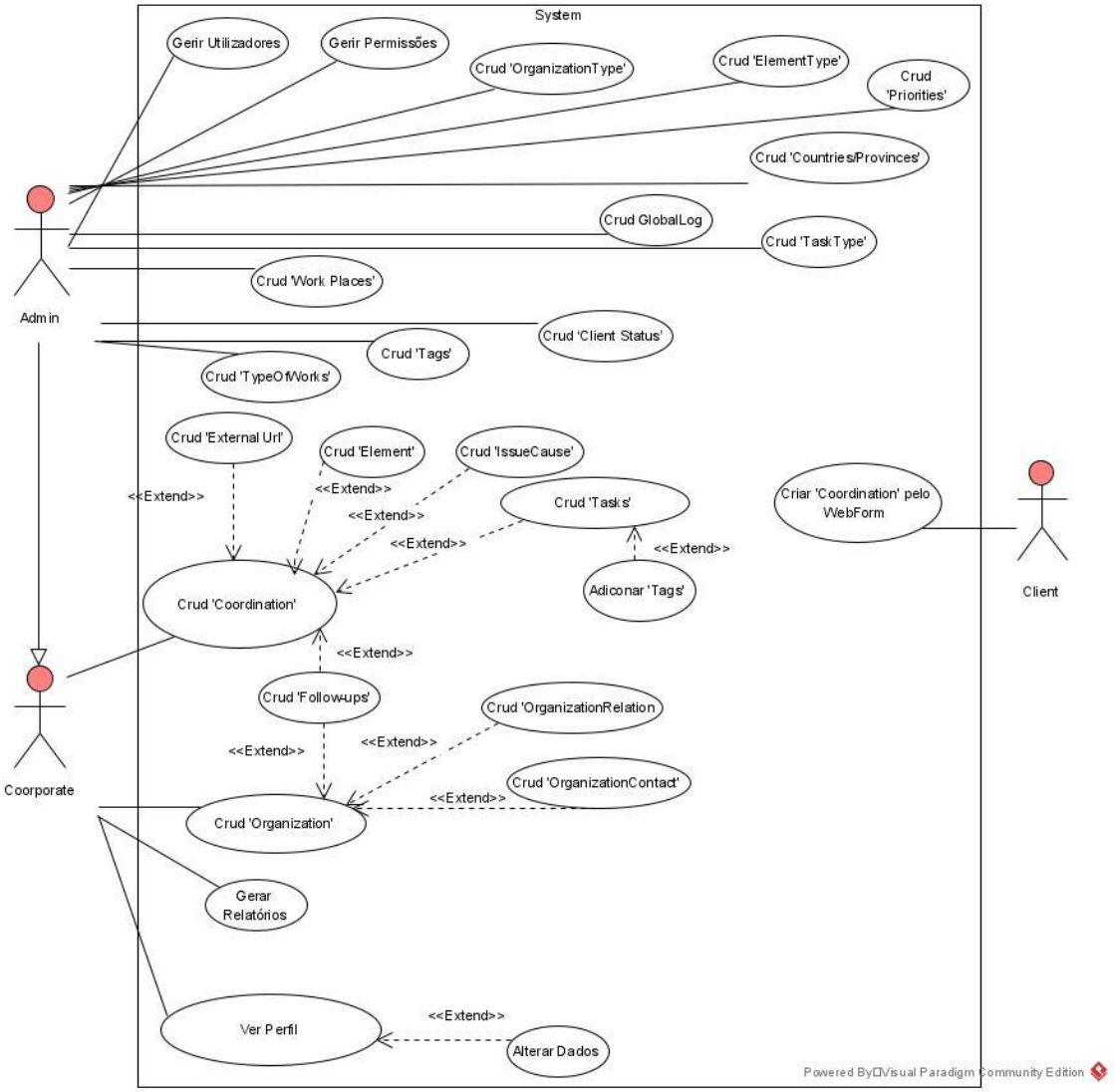


Figura 2 - Diagrama de casos de uso

4.4. Planificação do Trabalho

Ao longo do estágio, foi redigido um documento com referência à semana de trabalho e às correspondentes tarefas realizadas. De forma a proporcionar uma visão mais completa e detalhada das atividades desenvolvidas.

No mês de fevereiro:

- Integração na equipa e conhecimento sobre os processos e ferramentas utilizadas pela empresa;
- Discussão dos objetivos do projeto e definição das primeiras metas;
- Configuração do ambiente de desenvolvimento com as ferramentas necessárias;
- Compreensão do âmbito do projeto, incluindo o estudo sobre a plataforma web a ser desenvolvida e as necessidades específicas da empresa;

No mês de março:

- Elaboração do design da base de dados usando MySQL Workbench;
- Foco na construção do 'backend', incluindo 'migrations', 'models', 'controllers', 'requests', 'permissions' e definição de rotas da API;

No mês de abril:

- Adição de 'transformers' para todas tabelas;
- Modificação da base de dados e atualização com migrações;
- Implementação de validações para garantir a integridade dos dados;

No mês de maio:

- Desenvolvimento de filtros e funcionalidades específicas;
- Desenvolvimento de 'scripts' para automatizar processos recorrentes, como a criação de réplicas de coordenações e o envio automático de e-mails;

No mês de junho:

- Criação de 'seeders';
- Implementação de funções como 'calculateWorkingHours' para utilizar no cálculo de campo ao longo da plataforma;

No mês de julho:

- CRUD página 'Organizations';
- Ajustes na base de dados;

No mês de agosto:

- Definição dos requisitos específicos para os informes a serem gerados;

- Progresso na criação das ‘views’ para relatórios, incluindo a elaboração de consultas SQL para recuperar os dados necessários;

5. Desenvolvimento

Neste capítulo, será abordado o desenvolvimento da plataforma web que foi o foco principal do meu estágio, detalhando algumas das atividades e desafios enfrentados. A análise será dividida em duas seções principais: *backend* e *frontend*, permitindo uma visão clara das contribuições realizadas e das metodologias empregadas em cada parte do sistema.

5.1. Desenvolvimento Backend

O desenvolvimento *backend* é responsável pela lógica de negócio e pela manipulação de dados. Neste subcapítulo, abordou-se os principais componentes que compõem a arquitetura do *backend*, incluindo a elaboração da base de dados, o desenvolvimento das rotas da API, o envio de notificações e a criação de relatórios.

5.1.1. Elaboração da base de dados

Para a criação da base de dados, utilizou-se a ferramenta *MySQL Workbench*. O processo iniciou-se com o estudo detalhado dos requisitos funcionais e não funcionais fornecidos. Estes documentos forneceram uma visão clara das necessidades do projeto, especificando como a plataforma deveria funcionar e quais eram as expectativas de desempenho, segurança e usabilidade.

Após a análise dos requisitos, iniciou-se a projeção da estrutura da base de dados. Este projeto envolveu a definição das tabelas, campos, tipos de dados, chaves primárias e estrangeiras, bem como as relações entre as tabelas. A criação deste design foi um passo fundamental para garantir que a base de dados suportaria eficientemente todas as funcionalidades necessárias para a plataforma.

Por questões de confidencialidade, apresenta-se uma versão incompleta do design da base de dados visível no Anexo A. Para facilitar a visualização de partes essenciais do modelo, disponibiliza-se no Anexo B uma representação parcial e ampliada de uma seção específica do diagrama ER.

5.1.2. Elaboração da *Backend*

Como mencionado anteriormente na construção da *backend*, foi exigido a utilização da *framework Laravel* em *PHP* seguindo o modelo padrão Model-View-Controller (MVC) proporcionado uma estrutura bem organizada e modular.

Para o desenvolvimento do código, utilizou-se o *Visual Studio Code*. Esta escolha foi motivada pelo facto de se tratar de um editor de código leve de fácil usabilidade, oferecendo a integração com sistemas de controlo de versões neste caso específico com o *GitLab*.

Passo 1: Criação da 'Migration'

Através do comando 'php artisan make:migration create_coordinations_table' no Laravel cria-se um arquivo na pasta 'database/migrations' que irá permitir aos desenvolvedores definirem/alterarem/apagarem tabelas utilizando PHP. O arquivo gerado define a estrutura da tabela e qualquer alteração que deve ser feita na base de dados.

Na Figura 3, observa-se um exemplo de uma 'migration' onde a função 'up' é responsável por aplicar as mudanças no esquema da base de dados e a função 'down' é responsável por permitir desfazer as alterações caso seja necessário.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('coordinations', function (Blueprint $table) {
            $table->id();
            $table->integer('status');
            $table->string('identification');
            $table->unsignedBigInteger('organization_id');
            $table->foreign('organization_id')->references('id')->on('organizations');
            $table->unsignedBigInteger('sub_organization_id');
            $table->foreign('sub_organization_id')->references('id')->on('sub_organizations');
            $table->unsignedBigInteger('coordination_type_id');
            $table->foreign('coordination_type_id')->references('id')->on('coordination_types');
            $table->unsignedBigInteger('organization_relation_id');
            $table->foreign('organization_relation_id')->references('id')->on('organization_relations');
            $table->datetime('start_date')->nullable();
            $table->datetime('end_date')->nullable();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('coordinations');
    }
};
```

Figura 3 - Exemplo de uma “migration”

Para este primeiro passo ficar completo é necessário executar o comando ‘ph artisan migrate’ para processar todas as ‘migrations’ pendentes e aplicar as mudanças descritas nesses arquivos.

Passo 2: Criação do arquivo ‘Model’

Para a criação deste arquivo o comando a ser usado será o seguinte: ‘php artisan make:model Coordination -m’.

No Laravel, a classe ‘Coordination’ é um modelo que representa a tabela ‘coordinations’ na base de dados. Este modelo utiliza o Eloquent ORM (Object-Relational Mapping) do *Laravel* para facilitar a interação com a tabela, fornecendo várias funcionalidades que simplificam a gestão de dados. A seguir, serão detalhadas algumas dessas funcionalidades utilizadas ao longo deste projeto.

- Atributos ‘fillable’ e ‘dates’

São utilizados na definição dos modelos para especificar quais colunas podem ser atribuídas em massa e quais campos são tratados como datas, respetivamente. Ver Figura 4.

```php
protected $fillable = [
    'status',
    'identification',
    'organization_id',
    'sub_organization_id',
    'coordination_type_id',
    'organization_relation_id',
    'start_date',
    'end_date',
    'has_lifting_platform',
    'has_work_height',
    'has_preventive_resources',
    'finish_time',
    'web_form',
    'access_third_date',
    'parent_id',
    'third_suborganization_id',
    'deadline',
    'deadline_date',
    'tag_id',
    'type_of_work_id',
    'validated',
    'time_from_first_task',
    'time_from_start_to_end',
    'user_id'
];

protected $dates = ['start_date', 'end_date', 'access_third_date'];
```

Figura 4 - Atributos ‘fillable’ e ‘dates’

• Atributos ‘casts’

É uma funcionalidade poderosa que permite definir como os valores dos atributos de um modelo devem ser transformados ao serem recuperados da base de dados e antes de serem armazenados novamente, a Figura 5 é um exemplo destes atributos.

```php
protected $casts = [
    'has_lifting_platform' => 'boolean',
    'has_work_height' => 'boolean',
    'has_preventive_resources' => 'boolean',
];
```

Figura 5 - Atributos ‘casts’

• Métodos ‘Setters’

São utilizados para manipular e configurar valores específicos antes de serem atribuídos aos atributos correspondentes do modelo. No exemplo da Figura 6 os métodos garantem que os atributos ‘has_lifting_platform’, ‘has_work_height’, ‘has_preventive_resources’ sejam definidos como ‘false’ caso nenhum valor seja fornecido.

```php
public function setHasLiftingPlatformAttribute($value)
{
    $this->attributes['has_lifting_platform'] = $value ?? false;
}

public function setHasWorkHeightAttribute($value)
{
    $this->attributes['has_work_height'] = $value ?? false;
}

public function setHasPreventiveResourcesAttribute($value)
{
    $this->attributes['has_preventive_resources'] = $value ?? false;
}
```

Figura 6 - Métodos 'Setters'

• Relações do 'Eloquent'

Composto por vários métodos que definem como um certo modelo está relacionado com outro. Nesse sentido, esses métodos são usados para definir as ligações entre as tabelas da base de dados e as entidades no código, permitindo consultas e manipulação dos dados de forma simples.

Por exemplo na Figura 7, o método 'coordinationElements()' indica que um objeto 'coordination' pode ter muitos objetos 'coordinationElements', o que é equivalente a dizer que para cada registo na tabela 'coordinations' podem estar associados vários na tabela 'coordinationElements'.

```php
public function coordinationTasks()
{
    return $this->hasMany(CoordinationTask::class, 'coordination_id');
}

public function coordinationIssues(): HasMany
{
    return $this->hasMany(CoordinationIssue::class);
}

public function coordinationElements(): HasMany
{
    return $this->hasMany(CoordinationElement::class, 'coordination_id');
}
public function tag(): BelongsTo
{
    return $this->belongsTo(Tag::class);
}
```

Figura 7 - Exemplo Relações do 'Eloquent'

• Método ‘Scope Filter’

O método ‘scopeFilter’ no *Eloquent ORM* do *Laravel* simplifica a criação de consultas dinâmicas, permitindo a aplicação de filtros condicionais com base nos parâmetros recebidos através de requisições HTTP.

Um dos filtros presentes na Figura 8 filtra registos cuja ‘end_date’ seja maior ou igual ao valor fornecido com hora inicial ‘00:00:00’. Por outro lado, ‘end_date’ será filtrado para valores menores ou iguais ao valor fornecido com hora final, neste caso, ‘23:59:59’.

```php
/**
 * Scope a query to only include coordination based on the request.
 */
public function scopeFilter(Builder $query, Request $request): void
{
    if (isset($request->organization_id)) {
        $query->where('organization_id', $request->organization_id);
    }
    if (isset($request->identification)) {
        $query->where('identification','LIKE', '%' . $request->identification . '%');
    }
    if (isset($request->tag_id)) {
        $query->where('tag_id', $request->tag_id);
    }
    if (isset($request->status)) {
        $query->where('status', $request->status);
    }
    if (isset($request->start_date)) {
        $query->where('start_date', 'LIKE', '%' . $request->start_date . '%');
    }
    if (isset($request->sub_organization_id)) {
        $query->where('sub_organization_id', $request->sub_organization_id);
    }
    if (isset($request->organization_relation_id)) {
        $query->where('organization_relation_id', $request->organization_relation_id);
    }
    if (isset($request->validated)) {
        $query->where('validated', $request->validated);
    }
    if (isset($request->web_form)) {
        $query->where('web_form', $request->web_form);
    }
    if (isset($request->end_date_start)) {
        $query->where('end_date', '>=', $request->end_date_start . ' 00:00:00');
    }
    if (isset($request->end_date_end)) {
        $query->where('end_date', '<=', $request->end_date_end . ' 23:59:59');
    }
```

Figura 8 - Exemplo método ‘scopeFilter’

• **Passo 3: Criação dos arquivos ‘Request’**

A criação destes arquivos no desenvolvimento de plataformas é bastante comum e tem como objetivo fazer a validação e autorização dos dados transmitidos via HTTP.

Nesse sentido, esses arquivos são essenciais para garantir que os dados recebidos estão no formato pretendido, verificar se são campos de carater obrigatório ou não, ou seja, se atendem a todos os critérios pré-estabelecidos antes de serem processados pelo sistema.

Na Figura 9, é definida a classe 'CoordinationUpdateRequest' que define as regras de validação que serão aplicadas às entradas da solicitação. Por exemplo, para verificar se um 'status' é inválido ou não é utilizada a classe 'ReflectionClass' retornando todas as constantes definidas em 'CoordinationStatus' dentro do *namespace* 'App\Enums\Coordination'. Essas constantes são armazenadas em um array, sendo 'validStatusValues' uma lista de todos os valores de 'status' válidos.

Nesse sentido, essa lista será utilizada posteriormente para verificar a validade do campo 'status'.

```php
class CoordinationUpdateRequest extends FormRequest
{
    /**
    /**
     * Get the validation rules that apply to the request.
     *
     * @return array<string, \Illuminate\Contracts\Validation\Rule|array|string>
     */
    public function rules(): array
    {
        // Obtenha todos os valores definidos em CoordinationStatus
        $validStatusValues = array_values((new \ReflectionClass(CoordinationStatus::class))->getConstants());

        return [
            'identification' => ['string'],
            'status' => [
                'integer',
                function ($attribute, $value, $fail) use ($validStatusValues) {
                    if (!in_array($value, $validStatusValues)) {
                        $fail('Status inválido.');
                    }
                }
            ],
            'end_date' => ['string', new DateFormat(['Y/m/d H:i:s', 'Y-m-d\TH:i:s.u\Z'])],
            'coordination_type_id' => ['integer'],
            'has_lifting_platform' => ['boolean'],
            'has_work_height' => ['boolean'],
            'has_preventive_resources' => ['boolean'],
            'validated'=> ['boolean'],
            'web_form' => ['boolean'],
            'parent_id' => ['integer'],
            'deadline'=> ['integer'],
            'tag_id' => ['integer'],
            'type_of_work_id'=> ['integer'],
            'user_id' => ['integer']
        ];
    }
}
```

Figura 9 - Exemplo de um "Request" usado para a operação 'update' de uma "Coordination"

- **Passo 4: Criação de 'Permissions'**

A criação de 'permissions' permite o controlo do acesso a diferentes partes da plataforma e a diferentes funcionalidades, ver Figura 10.

No projeto, as várias 'permissions' estão associadas a cada "role" e estas por sua vez estão atribuídas a cada utilizador. Por exemplo, cada administrador tem uma "role" chamada "admin" e esta tem todas as 'permissions' atribuídas.

```php
<?php

namespace App\Enums\Permissions;

enum CoordinationPermission: string
{
    // case NAMEINAPP = 'name-in-database';
    case CreateCoordination = 'create coordination';
    case ViewCoordination = 'view coordination';
    case UpdateCoordination = 'update coordination';
    case DeleteCoordination = 'delete coordination';


    // extra helper to allow for greater customization of displayed values,
    //without disclosing the name/value data directly
    public function label(): string
    {
        return match ($this) {
            static::CreateCoordination => 'Create coordinations',
            static::ViewCoordination => 'View coordinations',
            static::UpdateCoordination => 'Update coordinations',
            static::DeleteCoordination => 'Delete coordinations',
        };
    }
}
```

Figura 10 - Exemplo de uma 'Permission'

- **Criação do arquivo 'Controller'**

Este tipo de ficheiros são os responsáveis por receberem as requisições HTTP dos utilizadores e retornaram as respostas pretendidas, podendo ser elas operações do tipo: 'Create', 'Read', 'Update' e "Delete".

Neste exemplo da Figura 11, o método 'index' é responsável por exibir uma lista de 'coordinations'. Neste caso utiliza-se método 'filter' para aplicar filtros com base na requisição do utilizador. Este método foi utilizado com o objetivo de reduzir os dados transferidos e consequentemente, diminuir o tempo de espera do pedido.

```php
public function index(Request $request): JsonResponse
{
    try {

        // Filtra as coordenações com base no tipo
        $coordinations = Coordination::select('id', 'identification','status', 'organization_id','tag_id',
        'start_date', 'end_date', 'coordination_type_id','organization_relation_id','sub_organization_id')
        ->with([
            'organization:id,company_name',
            'subOrganization:id,name',
            'organizationRelation:id,organization2_id',
            'organizationRelation.organization2:id,company_name',
            'tag:id,name',
            'coordinationType:id,name'
        ])->filter($request)->get();

        // Retorna o modelo personalizado com os dados
        return responder()->success($coordinations, CoordinationIndexTransformer::class)->meta(['totalCount' => $coordinations->count()])->respond();
    } catch (Exception $e) {
        return responder()->error(null, $e->getMessage())->respond();
    }
}
```

Figura 11 - Exemplo método 'index'

O método 'store' (Figura 12) no arquivo 'CoordinationController' tem como objetivo criar uma 'coordination'. Por questões de confidencialidade e integridade de dados não será possível partilhar o código completo. Como se pode verificar na figura anterior na criação de uma 'coordination' é verificado se o utilizador tem a permissão necessária para essa operação.

```php
public function store(CoordinationStoreRequest $request): JsonResponse
{

    DB::beginTransaction();
    try {
        $this->middleware('permission:' . CoordinationPermission::CreateCoordination->value);

        if (isset($request->organization_id)) {···
        }
    } catch (Exception $e) {
        DB::rollBack();
        return responder()->error(null, $e->getMessage())->respond();
    }
}
```

Figura 12 -Exemplo método 'store'

Uma das verificações feitas na criação de uma 'coordination' diz respeito à atribuição da 'coordination_type' como na Figura 13.

Nesse sentido, foi começado por calcular a diferença em meses entre a data atual e a data armazenada na variável '$end_date' e guardado esse valor em '$diff'. Antes desse calculo é necessário através da função "Carbon::parse()" converter essa data em uma objeto do tipo 'Carbon' que é uma biblioteca de manipulação de datas em PHP, permitindo operações como cálculos de diferenças entre datas. Portanto o valor resultante de "$diff" representa quantos meses existem entre a data atual e a data de término da coordenação.

A determinação do tipo de coordenação ('coordination_type') depende da condição de criação da coordenação. Se a coordenação for criada através de um formulário da web

(‘$coordination [web_form] == ‘true’’), a lógica avança para verificar o valor de “$diff” em relação a um valor definido em ‘config(‘app.month_coordination_type’)’. Esse valor é obtido do arquivo de configuração “config/app.php” da aplicação. Se ‘$diff’ for maior que o valor especificado em ‘config(‘app.month_coordination_type’)’, a coordenação é considerada do tipo "x". Caso contrário, é considerada do tipo "y”.

Por outro lado, se a coordenação não estiver sendo criada através de um formulário da web (‘$coordination [web_form] != ‘true’’), então simplesmente atribuiu-se o tipo de coordenação que já está especificado em ‘$coordination['coordination_type']’.

```
$end_date = Carbon::parse($coordination["end_date"]);
$diff = now()->diffInMonths($end_date);

if ($coordination["web_form"] == "true") {
    if ($diff > config("app.month_coordination_type")) {
        $coordination_type = CoordinationType::where("name", "x")->first()->id;
    } else {
        $coordination_type = CoordinationType::where("name", "y")->first()->id;
    }
} else {
    $coordination_type = $coordination['coordination_type']; // assuming this is passed by the user
}
```

Figura 13 - Calcular ‘coordination_type’

Como nos outros métodos aqui também é feita a verificação se o utilizador tem permissão para realizar esta operação através do ‘middelware’.

Como se trata de um método que vai atualizar campos de uma certa ‘coordination’ é necessário verificar se esta existe e caso não seja encontrada (‘$coordination == null’) deve ser lançado uma exceção a indicar isso mesmo. Antes de fazer a atualização de qualquer campo é necessário guardar em um ‘array’ ‘$oldValues’ os campos atuais, este ‘array’ é importante para no futuro serem comparados aos valores atualizados e verificar quais alterações existiram, como pode ser observado na Figura 14.

```php
public function update(CoordinationUpdateRequest $request, string $id)
{
    try {
        $this->middleware('permission:' . CoordinationPermission::UpdateCoordination->value);

        $coordination = Coordination::find($id);

        if ($coordination == null) {
            throw new Exception(trans("errors.not_found", ['attribute' => 'Coordination']));
        }

        // Armazena os valores originais antes de atualizar
        $oldValues = $coordination->toArray();
```

Figura 14 - Parte 1 exemplo método 'update'

Seguidamente, pode-se proceder à atualização da 'coordination', o método 'validated()' apenas irá retornar os campos atualizados depois destes passarem pelas regras estabelecidas no arquivo 'CoordinationUpdateRequest', como abordado anteriormente.

Posteriormente, será abordada superficialmente uma parte do código responsável por verificar e calcular o tempo de trabalho quando o status de uma 'coordination' é atualizado. Para mais detalhes, consulte a Figura 15.

Inicialmente, o código verifica se o campo 'status' está presente na requisição e se o novo 'status' é um dos valores 5, 6 ou 7. Isso garante que a verificação só continue se o 'status' for um desses valores específicos. Em seguida, é feita uma procura ao status anterior da 'coordination', ignorando o registo mais recente que está a ser criado agora. Isso é feito para comparar o "status" antigo com o 'status' novo.

O código então verifica se o status anterior não está entre os valores 5, 6 ou 7. Se estiver, a verificação é interrompida. Caso contrário, continua com a lógica. O próximo passo é encontrar o registo mais recente na tabela de 'logs' de "status" da coordenação ('CoordinationStatusLog') que tenha um status de 5, 6 ou 7. Se o registo encontrado tiver o mesmo 'status' que o novo 'status' na requisição, o código prossegue para calcular a diferença de tempo entre quando a coordenação foi criada e quando o status foi alterado.

O código encontra as informações da 'organization' associada à 'coordination' e, posteriormente, os dados do utilizador associado a essa 'organization'. Verifica se o utilizador tem um local de trabalho associado e, se tiver, obtém o horário de trabalho correspondente. Isso é crucial para calcular corretamente as horas de trabalho. Finalmente, uma instrução SQL é executada para atualizar o campo 'finish_time' na

tabela ‘coordiantion’. Isso é feito utilizando uma função que calcula as horas de trabalho com base nos horários de criação e alteração da ‘coordinatiton’.

```
// Verifica se o status está mudando para 5, 6 ou 7
if ($request->filled('status') && in_array($request->status, [5, 6, 7])) {
    // Verifica se o status anterior é 1, 2, 3 ou 4
    $previousStatus = CoordinationStatusLog::where('coordination_id', $id)
        ->orderBy('created_at', 'desc')
        ->skip(1) // Pula o registro mais recente (o que está sendo criado agora)
        ->first();

    if (!$previousStatus || !in_array($previousStatus->status, [5, 6, 7])) {
        // Consulta o registro mais recente na CoordinationStatusLog com status 5, 6 ou 7
        $endLog = CoordinationStatusLog::where('coordination_id', $id)
            ->whereIn('status', [5, 6, 7])
            ->orderBy('created_at', 'desc')
            ->first();

        if ($endLog && $endLog->status == $request->status) {
            // Calcula a diferença de tempo entre o momento em que o status muda para 5, 6 ou 7 e o momento em que a coordenação
            if ($coordination) {
                $organization = $coordination->organization;

                $user = $organization->user;

                Log::info('WorkSchedule ID: ' . $user->workPlace);
                // Verifica se o utilizador existe e se tem locais de trabalho associados
                if ($user && $user->workPlace) {
                    // Acessa o relacionamento WorkPlaces
                    $workPlace = $user->workPlace;

                    Log::info('WorkSchedule ID: ' . $workPlace->workingSchedule);
                    // Verifica se há um WorkingSchedule associado ao local de trabalho
                    if ($workPlace->workingSchedule) {
                        // Obtém o ID do workingSchedule
                        $workingScheduleId = $workPlace->workingSchedule->id;
                        $workingScheduleId = intval($workPlace->workingSchedule->id);

                        // Execute a raw SQL statement para atualizar o atributo time_from_first_task
                        DB::statement("
            UPDATE coordinations
            SET finish_time = calculateWorkingHours(
                '{$coordination->created_at}',
                '{$endLog->created_at}',
                $workingScheduleId
            )
            WHERE id = {$coordination->id}
        ");
                    }
                }
```

Figura 15 - Parte 2 exemplo método 'update'

Uma das funções criadas ao longo deste projeto foi a ‘calculateWorkingHours’ inicialmente foi construída no ‘phpMyAdmin’ e futuramente implementada através de uma ‘migration’ para praticidade a longo prazo.

```
            DB::statement("
    UPDATE coordinations
    SET finish_time = calculateWorkingHours(
        '{$coordination->created_at}',
        '{$endLog->created_at}',
        $workingScheduleId
    )
    WHERE id = {$coordination->id}
");
```

Figura 16 - Uso da função 'calculateWorkingHours'

Esta função começa por declarar todas as variáveis e os seus respetivos tipos, como pode ser observado na Figura 17.

```
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        // Call the MySQL function using DB::statement
        DB::unprepared('
            CREATE FUNCTION `calculateWorkingHours`(startDate DATETIME, endDate DATETIME, workingScheduleId INT) RETURNS time
            BEGIN
                DECLARE totalHours INT;
                DECLARE workingHours INT;
                DECLARE currentDate DATETIME;
                DECLARE result TIME;
                DECLARE sundaySecs INT;
                DECLARE mondaySecs INT;
                DECLARE tuesdaySecs INT;
                DECLARE wednesdaySecs INT;
                DECLARE thursdaySecs INT;
                DECLARE fridaySecs INT;
                DECLARE saturdaySecs INT;
                DECLARE startDateTime DATETIME;
                DECLARE endDateTime DATETIME;

                DECLARE mondayWorkTime INT;
                DECLARE fridayWorkTime INT;
                DECLARE lunchBreakTime INT;

                DECLARE sundayStartLunch TIME;
                DECLARE sundayEndLunch TIME;
                DECLARE mondayStartLunch TIME;
                DECLARE mondayEndLunch TIME;
                DECLARE mondayStart TIME;
                DECLARE mondayEnd TIME;
                DECLARE tuesdayStartLunch TIME;
                DECLARE tuesdayEndLunch TIME;
                DECLARE wednesdayStartLunch TIME;
                DECLARE wednesdayEndLunch TIME;
                DECLARE thursdayStartLunch TIME;
                DECLARE thursdayEndLunch TIME;
                DECLARE fridayStartLunch TIME;
                DECLARE fridayEndLunch TIME;
                DECLARE fridayStart TIME;
                DECLARE fridayEnd TIME;
                DECLARE saturdayStartLunch TIME;
                DECLARE saturdayEndLunch TIME;
```

Figura 17 - Parte 1: Função 'calculateWorkingHours'

Posteriormente, é feito o 'select' ao campo 'working_seconds' para cada dia da semana da tabela 'working_schedule_details' associado ao 'workingScheduleId', este é um argumento da função.

Vai buscar o ‘start_time’, ‘end_time’, ‘start_lunch_time’ e ‘end_lunch_time’ para transformar em horas e minutos (‘"%H:%i"’) e caso seja ‘null’ será atribuído o valor ‘00:00:00’. Este processo será realizado para todos os dias da semana. Foi utilizado o ‘mondayWorkTime’ para calcular os segundos de trabalho pois seria um dia completo de trabalho, e a ‘fridayWorkTime’ um dia especial havendo menos horas de trabalho por parte da empresa. Através da função ‘calculateWorkDaySeconds’ que também foi implementada durante o estágio calcular-se os segundos entre essas quatro horas, obtendo o tempo que cada utilizador esteve a trabalhar. Por último, é ainda calculado recorrendo a ‘TIMEDIFF()’ o tempo de pausa para almoço. Ver Figura 18.

```
SELECT COALESCE(STR_TO_DATE(start_lunch_time, "%H:%i"), "00:00:00") INTO mondayStartLunch FROM working_schedule_details Where day_id = 2 AND working_schedule_id = workingScheduleId;
SELECT COALESCE(STR_TO_DATE(end_lunch_time, "%H:%i"), "00:00:00") INTO mondayEndLunch FROM working_schedule_details Where day_id = 2 AND working_schedule_id = workingScheduleId;
SELECT COALESCE(STR_TO_DATE(start_time, "%H:%i"), "00:00:00") INTO mondayStart FROM working_schedule_details Where day_id = 2 AND working_schedule_id = workingScheduleId;
SELECT COALESCE(STR_TO_DATE(end_time, "%H:%i"), "00:00:00") INTO mondayEnd FROM working_schedule_details Where day_id = 2 AND working_schedule_id = workingScheduleId

SELECT COALESCE(STR_TO_DATE(start_lunch_time, "%H:%i"), "00:00:00") INTO fridayStartLunch FROM working_schedule_details Where day_id = 6 AND working_schedule_id = workingScheduleId;
SELECT COALESCE(STR_TO_DATE(end_lunch_time, "%H:%i"), "00:00:00") INTO fridayEndLunch FROM working_schedule_details Where day_id = 6 AND working_schedule_id = workingScheduleId;
SELECT COALESCE(STR_TO_DATE(start_time, "%H:%i"), "00:00:00") INTO fridayStart FROM working_schedule_details Where day_id = 6 AND working_schedule_id = workingScheduleId;
SELECT COALESCE(STR_TO_DATE(end_time, "%H:%i"), "00:00:00") INTO fridayEnd FROM working_schedule_details Where day_id = 6 AND working_schedule_id = workingScheduleId;

SET mondayWorkTime = calculateWorkDaySeconds(mondayStart, mondayStartLunch, mondayEndLunch, mondayEnd);
SET fridayWorkTime = calculateWorkDaySeconds(fridayStart, fridayStartLunch, fridayEndLunch, fridayEnd);
SELECT TIME_TO_SEC(TIMEDIFF(mondayEndLunch, mondayStartLunch)) INTO lunchBreakTime;
```

Figura 18 - Parte 2: Função "calculateWorkingHours"

Caso a data de início (‘startDate’) e a data de fim (‘endDate’) sejam iguais, ou seja, a ‘coordination’ é realizada no mesmo dia, sendo o dia da semana ‘Friday’ que corresponde na função ‘DAYOFWEEK’ ao inteiro ‘6’, verificou-se se a data atual é menor ou igual que a hora de início de almoço e se a hora de fim é maior ou igual que a hora de fim de almoço. Se assim for é retirado os segundos da pausa de almoço, e por fim foi calculado a diferença a data atual e a data de fim de trabalho nesse dia.

Por outro lado, se o dia da semana não for ‘Friday’ o processo é repetido, sendo que a pausa para o almoço corresponde a um dia normal. No entanto, se a ‘coordination’ for terminada sem passar pela pausa de almoço calcula-se diretamente o tempo de trabalho. Como se vê na Figura 19.

```
IF Date(currentDate) = Date(endDate) THEN
    IF DAYOFWEEK(currentDate) In (6)
        IF HOUR(currentDate) <= fridayStartLunch AND HOUR(endDate) >= fridayEndLunch THEN
            SET workingHours = workingHours - fridaylunchBreakTime;
            SET workingHours = workingHours + TIMESTAMPDIFF(SECOND, currentDate, endDate);
        END IF;
    ELSE
        IF HOUR(currentDate) <= mondayStartLunch AND HOUR(endDate) >= mondayEndLunch THEN
            SET workingHours = workingHours - lunchBreakTime;
            SET workingHours = workingHours + TIMESTAMPDIFF(SECOND, currentDate, endDate);
        ELSE
            SET workingHours = workingHours + TIMESTAMPDIFF(SECOND, currentDate, endDate);

        END IF;
    END IF;
ELSE
```

Figura 19 - Parte 3: Função 'calcuteWorkingHours'

O código percorre as datas entre ‘startDate’ e ‘endDate’, verificando se cada dia é um dia útil, ou seja, se não é feriado e está dentro do cronograma de trabalho. Para cada data, ele realiza o cálculo das horas trabalhadas. Se o dia for o primeiro (‘startDate’), o código calcula o tempo trabalhado até o fim do expediente. Se o dia for o último (‘endDate’), ele calcula o tempo desde o início do expediente. Para os dias intermediários, adiciona o tempo total de trabalho diário, considerando se é uma sexta-feira ou outro dia da semana. Além disso, o código ajusta as horas trabalhadas caso o cálculo envolva o período de almoço. No final, o total de horas trabalhadas é convertido em tempo e retornado, conforme mostrado na Figura 20.

```
    END IF;
ELSE
    WHILE currentDate <= endDate DO
        -- Check if the current date is a non-working day or a holiday
        IF DAYOFWEEK(currentDate) NOT IN (
            SELECT wsc.day_id
            FROM working_schedule_details wsc
            WHERE wsc.working_schedule_id = workingScheduleId
            AND wsc.working_day = false
        ) AND NOT isHolidayOnDate(currentDate, workingScheduleId) THEN
            IF currentDate = startDate THEN
                IF DAYOFWEEK(currentDate) = 6 THEN
                    SET endDateTime = CONCAT(DATE_FORMAT(currentDate, "%Y-%m-%d"), " ", TIME_FORMAT(fridayEnd, "%H:%i:%s"));
                ELSE
                    SET endDateTime = CONCAT(DATE_FORMAT(currentDate, "%Y-%m-%d"), " ", TIME_FORMAT(mondayEnd, "%H:%i:%s"));
                END IF;
                IF Hour(currentDate) >= mondayEndLunch THEN
                    SET workingHours = workingHours + TIMESTAMPDIFF(SECOND, currentDate, endDateTime);
                ELSE
                    SET workingHours = workingHours - lunchBreakTime;
                    SET workingHours = workingHours + TIMESTAMPDIFF(SECOND, currentDate, endDateTime);
                END IF;
            ELSEIF Date(currentDate) = Date(endDate) THEN
                SET startDateTime = CONCAT(DATE_FORMAT(currentDate, "%Y-%m-%d"), " ", TIME_FORMAT(mondayStart, "%H:%i:%s"));
                IF Hour(currentDate) >= mondayEndLunch THEN
                    SET workingHours = workingHours - lunchBreakTime;
                    SET workingHours = workingHours + TIMESTAMPDIFF(SECOND, startDateTime, currentDate);
                ELSE
                    SET workingHours = workingHours + TIMESTAMPDIFF(SECOND, startDateTime, currentDate);
                END IF;
            ELSE
                IF DAYOFWEEK(currentDate) = 6 THEN
                    SET workingHours = workingHours + fridayWorkTime;
                ELSE
                    SET workingHours = workingHours + mondayWorkTime;
                END IF;
            END IF;
        END IF;
        SET currentDate = DATE_ADD(currentDate, INTERVAL 1 DAY);
    END WHILE;
END IF;
SET result = SEC_TO_TIME(workingHours);
RETURN result;
END
```

Figura 20 - Parte 4: Função 'calcuteWorkingHours'

Após a explicação da função 'calcuteWorkingHours' retornar-se à instrução SQL que é executada para atualizar o campo “finish_time” presente na Figura 16. Para o cálculo deste campo a função 'calcuteWorkingHours' tem como parâmetros a data de início da ‘coordination’ (‘created_at’), a data do último ‘log’ no ‘status’ 5, 6 ou 7 e ‘ID’ do horário de trabalho correspondente ao utilizador.

Este cálculo é então usado para definir o valor do campo ‘finish_time’, refletindo o tempo total de trabalho entre a criação da ‘coordination’ e o registo do último ‘log’ relevante.

- **Criação do arquivo “Transformer”**

Antes de enviar os dados na resposta de uma API são utilizados os “Transformers” para formatar os mesmos tornado os diferentes endpoints da sua API consistentes e garantir que os dados enviados ao cliente esteja no formato desejado.

Cada transformer mencionado no array “$load” é uma classe que implementa a lógica para transformar um determinado conjunto de dados, Figura 21. Neste caso:

- 'OrganizationTransformer': transforma os dados da 'organization';
- 'SubOrganizationTransformer': transforma os dados da 'sub-organization';
- 'CoordinationTypeTransformer': transforma os dados do tipo de 'coordination';
- 'TagTransformer': transforma os dados das 'tags';
- 'TypeOfWorkTransformer': transforma os dados do tipo de trabalho;

```php
class CoordinationTransformer extends Transformer
{
    /**
     * List of available relations.
     *
     * @var string[]
     */
    protected $relations = [];

    /**
     * List of autoloaded default relations.
     *
     * @var array
     */
    protected $load = [
        'organization' => OrganizationTransformer::class,
        'subOrganization' => SubOrganizationTransformer::class,
        'coordinationType' => CoordinationTypeTransformer::class,
        'tag' => TagTransformer::class,
        'typeofwork'  => TypeOfWorkTransformer::class,
    ];
```

Figura 21 - Propriedade '$load'

No método "transform" da Figura 22 inicialmente são feitas diversas conversões. A primeira converte os dados "start_date" e "end_date" para objetos do tipo 'DateTime', caso esse ainda não o sejam. Seguidamente, é verificado se os campos "deadline_date" e "access_third_date" existem no objeto "Coordination" e caso isso se verifique é feita a conversão para objetos "DateTime" e a formatação para o padrão "Y-m-d H:i:s". Por último, o método obtém os detalhes da relação da organização associada ao objeto "Coordination" usando a propriedade "organizationRelation". Se essa relação existir, converte-a em um array. Caso contrário, define "organizationRelationDetails" como sendo "null".

Finalmente, para terminar o método “transform” é retornado um array associativo contendo diversas propriedades do objeto “Coordination”, incluindo as datas formatadas, os detalhes da relação de organização e outras informações relevantes.

```php
public function transform(Coordination $coordination)
{
    // Convertendo start_date e end_date em objetos DateTime, se ainda não forem
    $startDate = is_string($coordination->start_date) ? new \DateTime($coordination->start_date) : $coordination->start_date;
    $endDate = is_string($coordination->end_date) ? new \DateTime($coordination->end_date) : $coordination->end_date;

    if ($coordination->deadline_date) {
        $deadline_date = is_string($coordination->deadline_date) ? new \DateTime($coordination->deadline_date) : $coordination->deadline_date;
        $formatted_deadline_date = $deadline_date->format('Y-m-d H:i:s');
    } else {
        $formatted_deadline_date = null;
    }

    if ($coordination->access_third_date) {
        $access_third_date = is_string($coordination->access_third_date) ? new \DateTime($coordination->access_third_date) : $coordination->access_third_date;
        $formatted_access_third_date = $access_third_date->format('Y-m-d H:i:s');
    } else {
        $formatted_access_third_date = null;
    }
    // Acessa os detalhes da relação organization_relation_id usando a view OrganizationRelationView
    $organizationRelation = $coordination->organizationRelation;
    $organizationRelationDetails = $organizationRelation ? $organizationRelation->toArray() : null;

    return [
        'id' => (int) $coordination->id,
        'status' => (int) $coordination->status,
        'identification' => (string) $coordination->identification,
        'organization_id' => (int) $coordination->organization_id,
        'sub_organization_id' => (int) $coordination->sub_organization_id,
        'coordination_type_id' => (int) $coordination->coordination_type_id,
        'start_date' => $startDate->format('Y-m-d H:i:s'),
        'end_date' => $endDate->format('Y-m-d H:i:s'),
        'has_lifting_platform' => (int) $coordination->has_lifting_platform,
        'has_work_height' => (int) $coordination->has_work_height,
        'has_preventive_resources' => (int) $coordination->has_preventive_resources,
        'finsih_time' => $coordination->work_time,
        'organization_relation_details' => $organizationRelationDetails,
        'web_form' => $coordination->web_form,
        'access_third_date'=> $formatted_access_third_date,
        'parent_id' => $coordination->parent_id,
        'third_suborganization_id' => (int) $coordination->sub_organization_id,
```

Figura 22 - Método ‘transform’

Para o arquivo ‘CoordinationTransformer’ ficar completo são criados vários métodos de inclusão para facilitar o adicionar de dados relacionados à resposta da API sem precisar de escrever lógica adicional nos ‘controllers’. Assim, funções ‘includeOrganization’ e ‘includeSubOrganization’ permitem incluir dados relacionados (neste caso, organizações e suborganizações) na resposta JSON ao recuperar um modelo do tipo ‘Coordination’ recorrendo à Figura 23, pode-se atentar em dois desses métodos.

```php
/**
 * Include related organization.
 *
 * @param  \App\Model\Document $document
 * @return mixed
 */
public function includeOrganization(Coordination $coordination)
{
    return $coordination->organization;
}
/**
 * Include related sub_organization.
 *
 * @param  \App\Model\Coordination $document
 * @return mixed
 */
public function includeSubOrganization(Coordination $coordination)
{
    return $coordination->subOrganization;
}
```

Figura 23 - Métodos de Inclusão de Relacionamentos

- **Criação do arquivo 'routes'**

O arquivo 'routes/api.php' é onde são definidas as rotas para a API da plataforma web, como se pode ver através da Figura 24.

```php
//Cooordination
Route::get('/coordination', [CoordinationController::class, 'index'])->name('coordination.index');
Route::get('/coordination/edit/{id}', [CoordinationController::class, 'edit'])->name('coordination.edit');
Route::put('/coordination/update/{id}', [CoordinationController::class, 'update'])->name('coordination.update');
Route::post('/coordination/store', [CoordinationController::class, 'store'])->name('coordination.store');
Route::delete('/coordination/destroy/{id}', [CoordinationController::class, 'destroy'])->name('coordination.destroy');


//CoordinationType
Route::get('/coordinationtype', [CoordinationTypeController::class, 'index'])->name('coordinationtype.index');
Route::get('/coordinationtype/edit/{id}', [CoordinationTypeController::class, 'edit'])->name('coordinationtype.edit');
Route::put('/coordinationtype/update/{id}', [CoordinationTypeController::class, 'update'])->name('coordinationtype.update');
Route::post('/coordinationtype/store', [CoordinationTypeController::class, 'store'])->name('coordinationtype.store');
Route::delete('/coordinationtype/destroy/{id}', [CoordinationTypeController::class, 'destroy'])->name('coordinationtype.destroy');

//CoordinationStatusLog
Route::get('/coordinationstatuslog', [CoordinationStatusLogController::class, 'index'])->name('coordinationstatuslog.index');

//CoordinationTask
Route::get('/coordinationtask', [CoordinationTaskController::class, 'index'])->name('coordinationtask.index');
Route::get('/coordinationtask/edit/{id}', [CoordinationTaskController::class, 'edit'])->name('coordinationtask.edit');
Route::put('/coordinationtask/update/{id}', [CoordinationTaskController::class, 'update'])->name('coordinationtask.update');
Route::post('/coordinationtask/store', [CoordinationTaskController::class, 'store'])->name('coordinationtask.store');
Route::delete('/coordinationtask/destroy/{id}', [CoordinationTaskController::class, 'destroy'])->name('coordinationtask.destroy');
```

Figura 24 - Exemplos de rotas

Por exemplo nas 'Coordinations':

- '/coordination' (GET): Lista todas as coordenações.
- '/coordination/edit/{id}' (GET): Exibe os detalhes de uma coordenação específica identificada por {id}.

- '/coordination/update/{id}' (PUT): Atualiza os detalhes de uma coordenação específica.
- '/coordination/store' (POST): Cria uma coordenação.
- '/coordination/destroy/'{id} (DELETE): Exclui uma coordenação específica.

5.1.3. Testes das Rotas da API

Depois de elaborar todos os passos explicados no subcapítulo 5.1.2 para todas as tabelas da base de dados, foi necessário garantir que todas as funcionalidades implementadas estavam a funcionar como era esperado.

Nesse sentido, foram realizados testes em cada uma das rotas para verificar se as operações de CRUD (Index, View, Create, Update, Destroy) estavam a ser executadas corretamente.

Assim, para a realização dos testes das rotas da API, foi utilizado a ferramenta (Postman, 2024) que permite testar endpoints de maneira eficiente, simulando requisições HTTP de diversos tipos, como GET, POST, PUT e DELETE.

Após a realização do login na API, conforme demonstrado na imagem Figura 25, um 'token' de autenticação é gerado e retornado na resposta da requisição. Este 'token' é essencial para realizar chamadas subsequentes às rotas protegidas da API, garantindo que apenas utilizadores autenticados possam aceder determinados recursos.

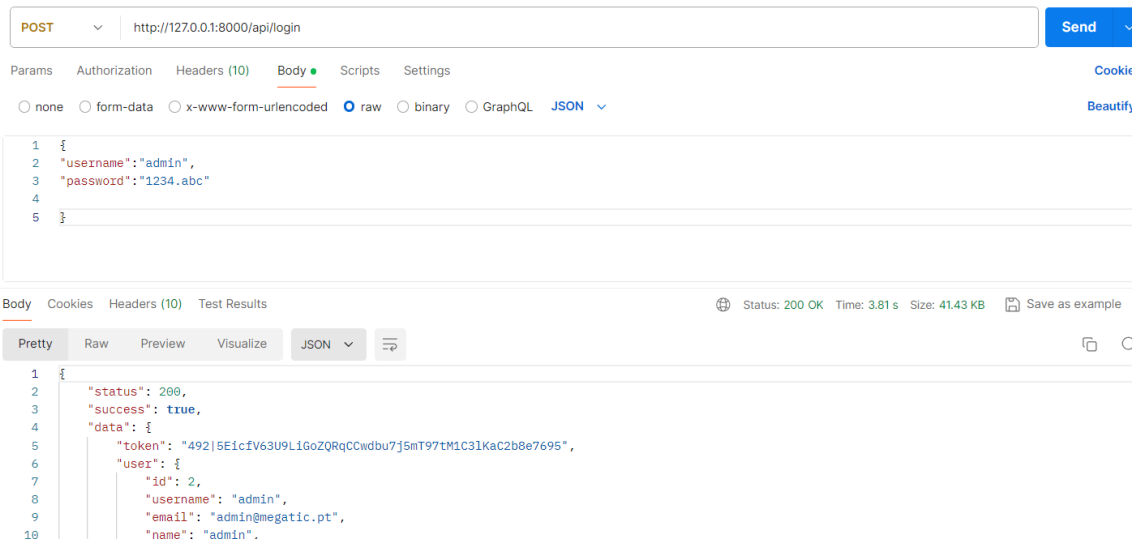


Figura 25 - Gerar 'token' através do Login

Na Figura 26, é demonstrado um exemplo de como realizar uma requisição POST para a criação de uma nova organização. Nesta requisição, o cliente inclui os dados necessários dentro do 'body' da requisição. O 'body' é a parte da mensagem HTTP onde são inseridas as informações que se deseja enviar ao servidor.

Nesse sentido, depois do processamento do pedido pelo servidor, a criação foi bem-sucedida obtendo 'HTTP 200 OK', indicando que a operação foi realizada com sucesso.

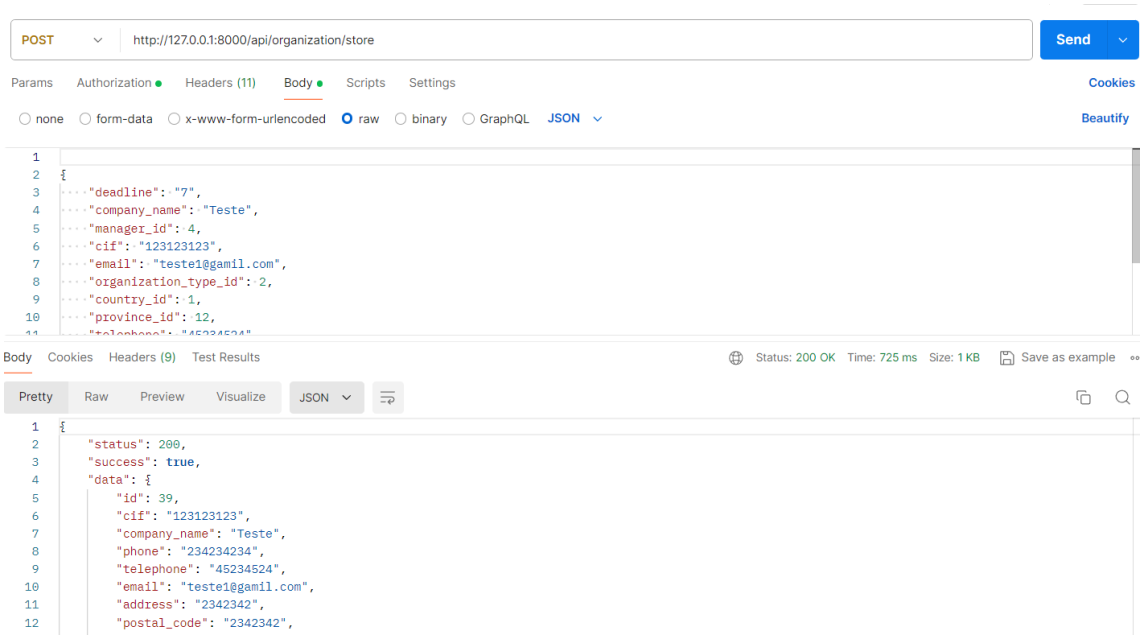


Figura 26 - Exemplo de uma requisição POST

Após criar uma organização, é possível recuperar os detalhes dessa organização específica utilizando uma requisição GET. No exemplo fornecido, foi feita uma requisição GET para o *endpoint* '/api/organization/edit/39', onde 39 é o ID da organização recém-criada. Como pode ser visto, além dos detalhes da 'organization' com ID 39, como 'company_name', 'email', 'cif', também é fornecida informações relacionadas, como detalhes por exemplo do gerente ('manager') e do utilizador ('user') associado, Figura 27.

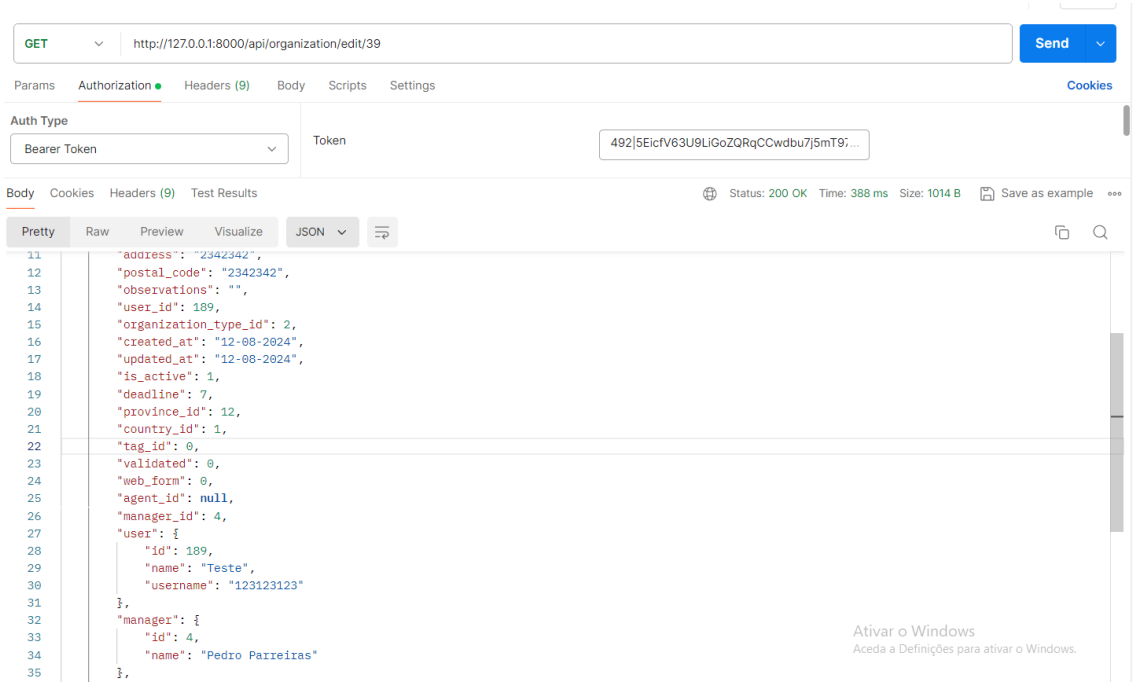


Figura 27 - Exemplo de uma requisição GET

5.1.4. Envio de Notificações

Neste seção, será abordada a implementação de um sistema automatizado para o envio de e-mails direcionados aos contatos de uma organização, a partir de condições específicas relacionadas a tarefas da coordenação. A automatização do envio de e-mails é uma ferramenta poderosa que pode otimizar significativamente a comunicação interna e externa, reduzindo a carga de trabalho manual e garantindo que as mensagens sejam entregues de maneira oportuna e precisa.

A implementação da classe chamada ‘NotificationCoordinationTaskController’ (Figura 28) é responsável por enviar notificações relacionadas a tarefas de uma determinada’Coordination. Dentro dessa classe, há uma função chamada ‘sendNotifications’, que será detalhada a seguir.

O início desta função tem como o objetivo é configurar o ambiente para que o envio de notificações seja executado corretamente. Para isso, é necessário obter algumas informações específicas que estão guardadas na tabela designada como ‘GlobalConfig’.

O valor do “ID” correspondente a ‘task_type’ sobre as quais serão enviadas notificações está guardado no campo ‘value’ quando a chave (‘key’) é

'coord_client_notif_task_types'. Esse valor é guardado na variável '$taskTypeConfig' caso seja encontrada, e será 'null' caso não seja.

Seguidamente, quando a '$taskTypeConfig' é encontrada o código continua e guarda o o "ID", que corresponde ao tipo de tarefa na variável '$taskTypeId' que será utilizado futuramente para filtrar as tarefas da coordenação.

Similar ao primeiro passo, agora é necessário saber outra configuração global identificada pela chave 'body_email'. Esta configuração específica contém o HTML do corpo do e-mail que será enviado nas notificações, sendo esse guardado na variável '$htmlEmailConfig'.

Depois de se obter essas informações é necessário fazer uma consulta SQL à base de dados para obter as tarefas de coordenação que se pretende. Essa consulta é sujeita a vários critérios por exemplo: o "status" da tarefa ser 4, o tipo da tarefa ser o mesmo que o guardado na variável '$taskTypeId' obtida anteriormente, o contato da organização deve permitir receber e-mails, entre outros.

Finalmente, a consulta é executada e o resultado armazenado na variável '$coordinationTasks'.

```
class NotificationCoordinationTaskController extends Controller
{
    public function sendNotifications($ctId)
    {
        $taskTypeConfig = GlobalConfig::where('key', 'coord_client_notif_task_types')->first();

        if ($taskTypeConfig) {
            $taskTypeId = $taskTypeConfig->value;

            $htmlEmailConfig = GlobalConfig::where('key', 'body_email')->first();

            if ($htmlEmailConfig) {

                $emailBody = $htmlEmailConfig->value;

                $queryCoordinationTasks = "
                SELECT  c.identification, ct.id, oc.email, o.company_name,o.agent_id,ct.task_type_id, usr.name, ct.external_observation
                FROM coordination_tasks ct
                JOIN coordinations c ON ct.coordination_id = c.id
                JOIN coordination_contact_infos oci ON ct.coordination_id = oci.coordination_id
                JOIN organization_contacts oc ON oci.organization_contact_id = oc.id
                JOIN organizations o ON oc.organization_id = o.id
                LEFT JOIN users as usr ON usr.id = o.agent_id
                WHERE ct.status IN (4)
                AND ct.task_type_id IN (:task_type_id)
                AND oc.send_email = 1 AND ct.id =:ct_id
                ";

                // Executar a consulta SQL para obter as tarefas de coordenação
                $coordinationTasks = DB::select($queryCoordinationTasks, ['task_type_id' => $taskTypeId, 'ct_id' => $ctId]);
```

Figura 28 - Parte 1 Função 'sendNotifications'

Para a função 'sendNotifications' ficar completa o código verifica se a variável '$coordinationTasks' não está vazia. Se não houver tarefas (ou seja, se a variável estiver vazia), o bloco "if" não será executado, e nenhum e-mail será enviado.

O código usa um laço ‘foreach’ para iterar sobre cada tarefa de ‘coordination’ retornada na consulta. Antes de enviar o e-mail, o código verifica se há alguma observação externa associada à tarefa (‘external_observation’). Se houver, essa observação é incluída no texto do e-mail. Caso contrário, a variável ‘$externalObservationText’ é definida como uma ‘string’ vazia.

O corpo do e-mail contém marcadores de texto que são substituídos pelos valores reais das informações da tarefa de coordenação atual. Essa substituição é feita utilizando o método ‘str_replace’, que substitui esses marcadores pelos dados específicos. O resultado é armazenado na variável ‘$emailBodyDynamic’, que representa o corpo do e-mail já personalizado e pronto para ser enviado.

O e-mail é enviado para o endereço especificado ‘$coordinationTask->email’ usando a classe ‘NotificationMailCoordinationTask’, que encapsula os dados da tarefa e o corpo do e-mail dinâmico, ver na Figura 29.

```php
// Verificar se existem tarefas
if (!empty($coordinationTasks)) {
    // Iterar sobre as tarefas para enviar e-mails
    foreach ($coordinationTasks as $coordinationTask) {
        // Verificar se external_observation é nulo e definir o texto apropriado
        $externalObservationText = $coordinationTask->external_observation ? 'Notas: ' . $coordinationTask->external_observation . '.' : '';

        // Substituir as variáveis no corpo do e-mail com os valores das tarefas de coordenação
        $emailBodyDynamic = str_replace(
            ['[company_name]', '[identification]', '[name]', '[external_observation]'],
            [$coordinationTask->company_name, $coordinationTask->identification, $coordinationTask->name, $externalObservationText],
            $emailBody
        );

        // Enviar o e-mail
        Mail::to($coordinationTask->email)->send(new NotificationMailCoordinationTask($coordinationTask, $emailBodyDynamic));
    }
    // Carregar o CoordinationTask do banco de dados
    $task = CoordinationTask::find($coordinationTask->id);

    if ($task) {
        // Obter o valor atual de nbr_alert
        $currentNbrAlert = $task->nbr_alert;

        // Incrementar o campo nbr_alert
        $task->nbr_alert = ($task->nbr_alert ?? 0) + 1;
        $task->save();

        // Criar um registro na tabela CoordinationTaskLog
        CoordinationTaskLog::create([
            'coordination_task_id' => $task->id,
            'user_name' => auth()->user()->username,
            'status' => $task->status,
            'action' => 'update_nbr_alert',
            'last_changes' => json_encode(['nbr_alert' => $currentNbrAlert]),
            'new_changes' => json_encode(['nbr_alert' => $task->nbr_alert]),
        ]);
    }
}
```

Figura 29 - Parte 2 Função ‘sendNotifications’

Nesse sentido, essa classe é responsável por personalizar o email a ser enviado, acrescentando por exemplo a assinatura da empresa e um assunto de e-mail adequado, Figura 30.

```php
class NotificationMailCoordinationTask extends Mailable
{
    use Queueable, SerializesModels;

    public $coordinationTask;
    public $emailBodyDynamic;

    public function __construct($coordinationTask, $emailBodyDynamic)
    {
        $this->coordinationTask = $coordinationTask;
        $this->emailBodyDynamic = $emailBodyDynamic;
    }

    public function build()
    {
        // Recuperar o HTML do corpo do e-mail da configuração global

        $assinatura = GlobalConfig::where('key', 'company_signature')->value('value');

        return $this->view('emails.notification')
                    ->with([
                        'emailBody' => $this->emailBodyDynamic,
                        'assinatura' => $assinatura,
                        ])
                    ->subject("Solicitação de Documento para a {$this->coordinationTask->identification}");
    }
}
```

Figura 30 - Classe “NotificationMailCoordinationTask”

Esta implementação garante que os e-mails sejam enviados de forma personalizada e que o sistema registe de maneira eficaz cada vez que um alerta é enviado, permitindo um controle adequado sobre o processo de notificação. Na Figura 31 pode-se ver um exemplo de um e-mail enviado para a organização ‘teste2’.

Figura 31 - Exemplo e-mail

5.1.5. Criação de Relatórios

Uma das tarefas realizadas durante este período de estágio foi a implementação de relatórios. Esta tarefa envolvia a extração e análise de dados já armazenados no sistema, por esse motivo a mesma só foi desenvolvida quando a base de dados estava completa e bem estruturada.

Nesse sentido, com a criação de 'Views' foi possível agregar vários dados provenientes de tabelas diferentes permitindo que os mesmo fossem filtrados e apresentados em forma de relatório.

Assim, foi utilizado a linguagem SQL para construir 'queries' de consulta, responsáveis por extrair e manipular os dados necessários para a criação dos relatórios. Essas 'queries' foram desenvolvidas e testadas diretamente no 'phpMyAdmin' utilizando 'JOINs', 'WHERE' e 'GROUP BY' para agregar os dados conforme necessário. Após as 'queries' serem testadas são utilizadas para criar uma 'View' na base de dados, a criação ocorre através de uma 'migration', utilizando o método 'DB::statement' para executar as 'queries' SQL diretamente na base de dados.

Posteriormente à criação da 'View', ela é integrada no sistema utilizando o modelo padrão Model-View-Controller (MVC). O arquivo 'model' serve como uma camada de abstração que permite que a aplicação interaja de forma eficiente e segura com a 'View'.

Seguidamente, como visto no subcapítulo 5.1.2 é implementado o método 'scopeFilter' responsável por filtrar os dados da 'View' com base nos parâmetros fornecidos.

Além disso, o 'Model' implementa a interface 'Transformable' e define um método 'transformer', que retorna um transformador específico, responsável por formatar os dados para serem retornados na API, garantindo que os dados estejam no formato desejado e esperado para a aplicação.

Por último, para cada relatório é implementado apenas um método no arquivo 'Controller' que irá retornar os dados da 'View' correspondente. Como os relatórios são apenas de visualização, não há necessidade de métodos para criar, atualizar ou eliminar dados e consequentemente, não existem ficheiros 'Request' específicos para validação dos mesmos.

Uma vez abordado o processo geral de criação de relatórios e a implementação do modelo Model-View-Controller (MVC), passarei a explicar alguns exemplos práticos, com foco

apenas na criação das ‘queries’ SQL. Como mencionado anteriormente, a implementação e integração completa no sistema, incluindo os ‘Models’, ‘Transformers’ e ‘Controllers’, já foi detalhada no capítulo 5.1.2.

Um dos relatórios que foi elaborado pretendia saber os o tempo de dedicação especificamente no contexto da solicitação de documentos. Como pode ser visto na ‘query’ da Figura 33 - 'Query' Tempos de dedicação é feito uma seleção a vários campos de tabelas diferentes com recurso a ‘INNER JOIN’ e ‘LEFT JOIN’. O número de alertas (‘nbr_alert’), calcula-se através do uso de um ‘count’ sempre que o método de enviar notificação é utilizado, cuja implementação foi abordada no capítulo 5.1.4 e o tempo que leva desde a solicitação de um documento ao cliente até que ele o envia (‘waiting_time’). Este ‘waiting_time’ foi calculado usando a função apresentada nas figuras: Figura 17Figura 18Figura 19Figura 20 passando os seguintes parâmetros: ‘created_at_status_4’, ‘created_at_next_transition’ e ‘workingScheduleId’.

O primeiro que diz respeito à data/hora que corresponde ao último registo quando o ‘status’ da ‘coordination_task’ estava a 4, ou seja, como ‘Requested_to_the_client’. O ‘created_at_next_transition’ é outra data/hora que indica quando a ‘coordination_task’ mudou para um novo ‘status’ (diferente de 4). Por último, ‘workingScheduleId’ que diz respeito ao ‘ID’ do horário de trabalho associado ao local do utilizador responsável por essa tarefa como é visível na Figura 32.

```
DB::statement("
    UPDATE coordination_tasks
    SET waiting_time = waiting_time + calculateWorkingHours(
        '{$created_at_status_4}',
        '{$created_at_next_transition}',
        '{$workingScheduleId}'
    )
    WHERE id = $coordinationTask->id
");
```

Figura 32 - Chamada da função para calcular 'waiting_time'

Após chamar todos os campos pretendidos na ‘query’ aplica-se um filtro para que só sejam mostradas as tarefas com ‘task_type_id’ igual a 6 (‘Solicitud de documentacion’). Além disso, o objetivo é filtrar apenas as ‘coordination_task’ que tem como ‘status’ os valores de 5, 6, e 7 (‘Incident’, ‘Cancelled’, ‘Validated’).

Assim sendo, este relatório pretende apresentar as 'coordination_tasks' que são do tipo 'Solicitud de documentacion' e cujo 'status' é 'Incident', 'Cancelled' ou 'Validated', mostrando quanto foi o tempo que o utilizador esperou pelas respostas do cliente e quantas vezes este foi alertado, Figura 33.

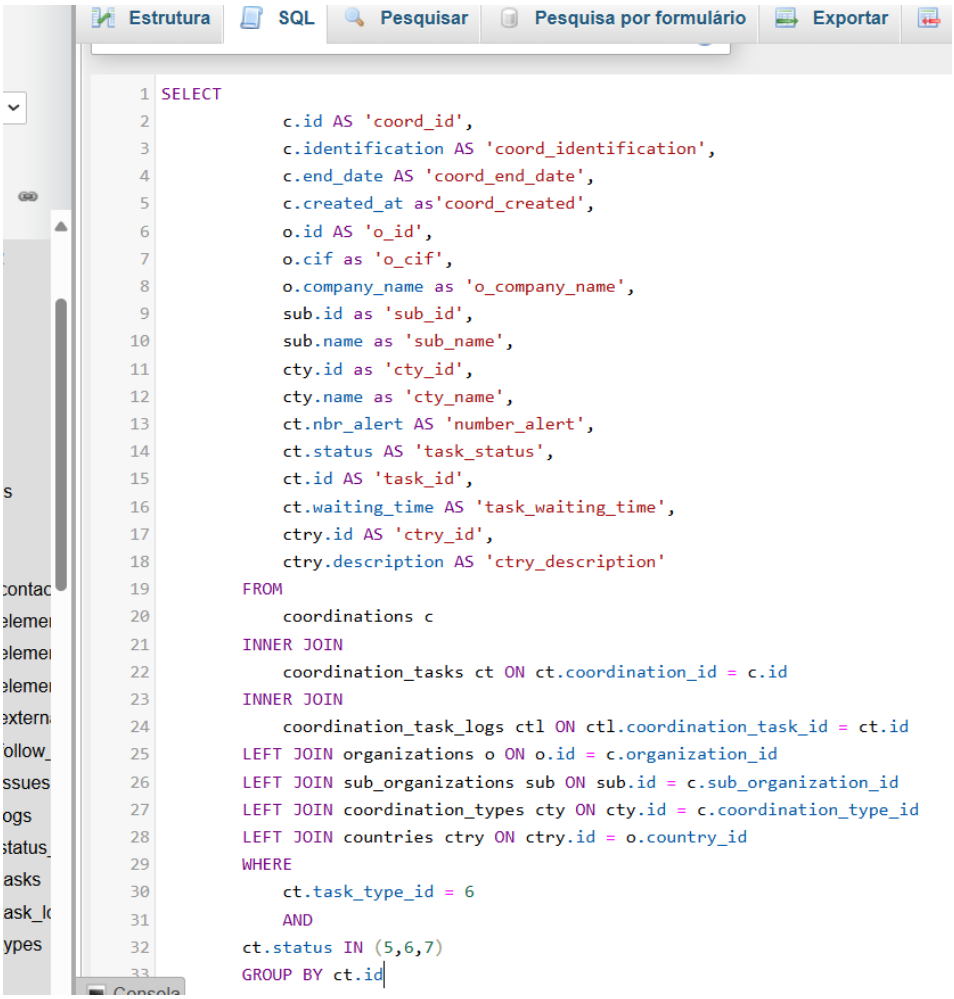


Figura 33 - 'Query' Tempos de dedicação

Outra 'query' implementada teve como intuito seleção de diversos campos de várias tabelas como: 'coordination', 'sub_organization', 'organization', 'issue_causes' e 'country'. Apos esse 'select' aos campos que seriam necessários apresentar, foram feitos vários cálculos personalizados com o objetivo de saber se uma determinada 'coordination' teria cumprido os prazos estipulados.

Nesse sentido, para saber a diferença em dias úteis entre a data limite de cada 'coordination' ('c.deadline_date') e a data do último registro do status ('csl.created_at), esta data é registada na tabela 'coordinationStatusLog', utiliza-se uma função similar àquela apresentada na Figura 32, no entanto esta calcula os dias úteis entre essas datas

considerando o horário de trabalho ('wp.working_schedule_id'). Se a data do último 'status' for posterior é classificado como 'NOK' e valor binário '0', casos seja anterior ou igual é classificado como 'OK' e valor binário '1', como é possível ver recorrendo à Figura 34.

```
SELECT
    `c`.`id` AS `id`,
    `c`.`identification` AS `c_identification`,
    `c`.`status` AS `c_status`,
    `c`.`created_at` AS `c_create_date`,
    `c`.`sub_organization_id` AS `sub_id`,
    `c`.`third_suborganization_id` AS `third_id`,
    `c`.`web_form` AS `c_web_form`,
    `sub`.`name` AS `sub_name`,
    `thirdsub`.`name` AS `thirdsub_name`,
    `o`.`company_name` AS `o_company_name`,
    `o`.`agent_id` AS `agent_id`,
    `agent`.`name` AS `agent_name`,
    `usr`.`name` AS `usr_name`,
    `usr`.`id` AS `usr_id`,
    `ic`.`type` AS `ic_type`,
    `ic`.`description` AS `ic_description`,
    `c`.`deadline_date` AS `c_deadline_date`,
    `csl`.`created_at` AS `csl_create`,
    `ctry`.`id` AS `ctry_id`,
    `ctry`.`description` AS `ctry_description`,
    `wp`.`working_schedule_id` AS `wp_working_schedule_id`,
    `wp`.`id` AS `wp_id`,
    CASE
        WHEN `csl`.`created_at` > `c`.`deadline_date` THEN `calculateWeekDays`(
            `c`.`deadline_date`,
            `csl`.`created_at`,
            `wp`.`working_schedule_id`
        ) * -1
        ELSE `calculateWeekDays`(
            `csl`.`created_at`,
            `c`.`deadline_date`,
            `wp`.`working_schedule_id`
        )
    END AS `diff_last_status`,
    CASE
        WHEN `csl`.`created_at` > `c`.`deadline_date` THEN 'NOK'
        ELSE 'Ok'
    END AS `status`,
    CASE
        WHEN `csl`.`created_at` > `c`.`deadline_date` THEN '0'
        ELSE '1'
    END AS `isok`
```

Figura 34 - Parte 1 'Query' Coordenações Realizadas

Por último, é necessário filtrar todos os registos, retornando dados relacionados às 'coordinations' e seus 'status' mais recentes (baseado no MAX(id)), com foco nas

'coordinations' que possuem status específicos 5, 6 ou 7 ('Cancelled', 'Archived', 'Done') como se poder ver através da Figura 35.

```
FROM
    `e-coordina_demo_v2`.`coordinations` `c`
    LEFT JOIN `e-coordina_demo_v2`.`coordination_issues` `ci` ON `ci`.`coordination_id` = `c`.`id`
    LEFT JOIN `e-coordina_demo_v2`.`issue_causes` `ic` ON `ci`.`issue_cause_id` = `ic`.`id`
    JOIN `e-coordina_demo_v2`.`coordination_status_logs` `csl` ON `csl`.`coordination_id` = `c`.`id`
    JOIN `e-coordina_demo_v2`.`organizations` `o` ON `o`.`id` = `c`.`organization_id`
    JOIN `e-coordina_demo_v2`.`sub_organizations` `sub` ON `sub`.`id` = `c`.`sub_organization_id`
    JOIN `e-coordina_demo_v2`.`sub_organizations` `thirdsub` ON `thirdsub`.`id` = `c`.`third_suborganization_id`
    LEFT JOIN `e-coordina_demo_v2`.`users` `usr` ON `usr`.`id` = `o`.`user_id`
    LEFT JOIN `e-coordina_demo_v2`.`users` `agent` ON `agent`.`id` = `o`.`agent_id`
    LEFT JOIN `e-coordina_demo_v2`.`work_places` `wp` ON `wp`.`id` = `usr`.`work_place_id`
    LEFT JOIN `e-coordina_demo_v2`.`working_schedules` `wsch` ON `wsch`.`id` = `wp`.`working_schedule_id`
    LEFT JOIN `e-coordina_demo_v2`.`countries` `ctry` ON `ctry`.`id` = `o`.`country_id`
WHERE
    `csl`.`id` IN (
        SELECT
            MAX(`e-coordina_demo_v2`.`coordination_status_logs`.`id`)
        FROM
            `e-coordina_demo_v2`.`coordination_status_logs`
        WHERE
            `e-coordina_demo_v2`.`coordination_status_logs`.`coordination_id` = `csl`.`coordination_id`
            AND `e-coordina_demo_v2`.`coordination_status_logs`.`status` IN (5, 6, 7)
        GROUP BY
            `e-coordina_demo_v2`.`coordination_status_logs`.`coordination_id`
    )
```

Figura 35 - Parte 2 'Query' Coordenações Realizadas

Concluindo, esta consulta tem como objetivo gerar um relatório detalhado das 'coordinations' que se encontram em determinados estados específicos, avaliando o cumprimento dos prazos das mesmas (Figura 36).

```
SELECT * FROM `finished_coordination_view`
```

Perfil [Editar em linha] [Edita] [Explicar SQL] [

Mostrar tudo | Restaurar ordem da coluna | Número de registos: 25 Filtrar registos: Pesquisar esta tabela

Opções

	id	c_identification	c_status	csl_create	c_deadline_date	diff_last_status	status	isok
Edita Copiar Apagar	4	IPB Megatic	7	2024-06-21 13:16:54	2024-06-10 00:00:00	-9	NOK	0
Edita Copiar Apagar	85	IPB Megatic	6	2024-06-02 10:34:25	2024-06-07 00:00:00	5	Ok	1
Edita Copiar Apagar	66	IPB Beta Innovations	5	2024-06-26 14:26:59	2024-06-13 00:00:00	-10	NOK	0

Figura 36 - Exemplo da 'View' Coordenações Realizadas

5.1.6. Criação de 'Controllers' específicos para recriar Coordenações

Neste capítulo, apresenta-seum 'controller' com uma função que pretende desempenhar um papel fundamental na recriação de dados essenciais. O objetivo é chamar esta função de forma automatizada.

O 'controller' RecreatePreviousMonthCoordinationController, tem como objetivo de recriar 'coordinations' do tipo 2 ('Monthly'). No entanto existem algumas restrições que foram implementadas. Inicialmente foi feita uma pesquisa à base de dados através de uma consulta SQL para encontrar todas as 'coordinations' cujo 'coordination_type' é 2, que a 'organization_relation_id' tenha uma prioridade 15 ('Alta') e por último filtrava pelo maior 'ID' quando a combinação dos campos 'organization_id', 'sub_organiztion_id' e 'organization_relation_id' fosse igual. Como apenas uma 'coordination' teria esses campos todos eles iguais, o maior 'ID' encontra a 'coordination' mais recente, como se pode ver através da Figura 37.

```
// Obtém a data atual
$currentDate = Carbon::now();
// Encontrar todas as coordenações do tipo 2 com organização relacionada com prioridade 15
$coordinations = DB::select('SELECT
    c.id, c.status, c.identification, c.organization_id, c.sub_organization_id,
    c.coordination_type_id, c.organization_relation_id, c.start_date, c.end_date, c.created_at, c.updated_at,
    c.has_lifting_platform, c.has_work_height, c.has_preventive_resources, c.finish_time, c.parent_id, c.web_form,
    c.access_third_date, c.third_suborganization_id, c.deadline, c.deadline_date, c.tag_id,c.type_of_work_id
    FROM coordinations AS c
    INNER JOIN organization_relations AS orgR ON c.organization_relation_id = orgR.id AND orgR.priority_id = 15
    WHERE c.coordination_type_id = 2
    AND c.id IN (
        SELECT MAX(id)
        FROM coordinations
        WHERE organization_id = c.organization_id
        AND sub_organization_id = c.sub_organization_id
        AND organization_relation_id = c.organization_relation_id
        GROUP BY organization_id, sub_organization_id, organization_relation_id
    )');
```

Figura 37 - Função 'recreatepreviousmonth' consulta SQL

Depois de constatar a presença de 'coordinations' resultantes dessa pesquisa, é definido o dia do mês em que as 'coordinations' deveram ser recriadas, este dia é guardado no campo 'day_job_coordination' da tabela 'globalConfig', caso não exista nenhuma configuração serão recriadas dia 1. Da mesma forma o campo 'deadline' para as novas 'coordinations' é atribuído pelo valor vindo da tabela 'globalConfig'.

Seguidamente, é necessário definir a data de início e de término. A 'start_date' é o valor da data atual, por outro lado o 'end_date' é calculado da seguinte forma: se o dia atual é menor ou igual ao dia configurado ('dayJobCoordination'), o 'end_date' será no mesmo

mês, um dia antes do 'dayJobCoordination'. Caso o dia atual seja maior que o 'dayJobCoordination', o end_date será no próximo mês, um dia antes do 'dayJobCoordination'. Após calcular essas datas, procede-se à criação dessa nova 'coordination', visível na Figura 38.

```php
if (!empty($coordinations)) {
    $globalConfig = GlobalConfig::where('key', 'day_job_coordination')->first();

    $dayJobCoordination = $globalConfig ? intval($globalConfig->value) : 1;

    $globalConfigDeadline = GlobalConfig::where('key', 'deadline')->first();

    $deadlineCoordination = $globalConfigDeadline ? intval($globalConfigDeadline->value) : 5;

    $currentDate = Carbon::now();

    $start_date = $currentDate->copy()->startOfDay();

    foreach ($coordinations as $coordination) {
        if ($currentDate->day <= $dayJobCoordination) {
            $end_date = $currentDate->copy()->endOfMonth()->day($dayJobCoordination - 1); // Último d
        } else {
            // Se o dia atual for maior que o dia de coordenação do trabalho
            $end_date = $currentDate->copy()->addMonth()->endOfMonth()->day($dayJobCoordination - 1);
        }

        $atual_coordination = Coordination::where('id', $coordination->id)->first();

        $atual_coordination->status = CoordinationStatus::Done;

        $atual_coordination->save();

        // Calcula a data limite subtraindo o número de dias do prazo da data de término
        $deadline_date = $end_date->copy()->startOfDay()->subDays($coordination->deadline);

        // Cria uma nova coordenação com os detalhes especificados
        $newCoordination = Coordination::create([
            'organization_relation_id' => $coordination->organization_relation_id,
            'coordination_type_id' => 2, // Coordenação mensal
            'deadline' => $deadlineCoordination, // Deadline
            'organization_id' => $coordination->organization_id,
            'sub_organization_id' => $coordination->sub_organization_id,
            'identification' => $coordination->identification,
            'start_date' => $start_date,
            'end_date' => $end_date,
            'deadline_date' =>  $deadline_date,
            'status' => CoordinationStatus::New,
        ]);
```

Figura 38 - Cálculo de datas e criação de novas coordenações 'recreatepreviousmonth'

Para finalizar, é necessário proceder à replicação dos 'elements' e dos 'documents' associados à 'coordination'. A cópia desses dados é criada por meio da função 'replicate()' e associada à nova 'coordination', conforme pode ser confirmado na Figura 39.

```
        }
        foreach ($coordinations as $coordinationData) {
            // Inicializar um objeto Coordination com os dados brutos da consulta
            $coordination = new Coordination();
            $coordination->id = $coordinationData->id;
            $coordination->status = $coordinationData->status;
            // Preencher outros atributos conforme necessário

            $elements = $coordination->coordinationElements()->get();

            foreach ($elements as $element) {
                $newElement = $element->replicate();
                $newElement->coordination_id = $newCoordination->id;
                $newElement->save();
            }

            $documents = $coordination->coordinationElementDocuments()->get();
            foreach ($documents as $document) {
                $newDocument = $document->replicate();
                $newDocument->coordination_id = $newCoordination->id;
                $newDocument->save();
            }
        }
    } else {
        return responder()->error("Nenhuma coordenação correspondente encontrada.")->respond();
    }

    // Confirma todas as alterações no banco de dados
```

Figura 39 - Replicação de Elementos e Documentos 'recreatepreviousmonth'

5.1.7. Criação de ‘Seeders’

Os ‘seeders’ foram utilizados para preencher a base de dados com dados iniciais, como utilizadores, configurações padrões ou quaisquer outros dados que comprometam o funcionamento da plataforma.

Assim, os ‘seeders’ desempenham um papel crucial na transição da plataforma para uma nova base de dados. Após utilizar o comando ‘php artisan migrate’ para criar as tabelas e colunas necessárias, recriando assim a estrutura da nova base de dados, os ‘seeders’ entram em ação. Depois das ‘migrations’, foi utilizado o comando ‘php artisan db:seed’, que vai ‘correr’ todos os ‘seeders’. Esses ‘seeders’ inserem automaticamente dados pré-definidos na base de dados, facilitando o processo de inicialização com informações básicas ou de configuração já prontas. Seguidamente serão apresentados os ‘seeders’ criados ao longo do desenvolvimento da plataforma. O arquivo ‘AdminSeeder’ é utilizado para criar 3 tipos diferentes de utilizadores ‘admin’, ’coorporate’ e ‘client’ como existiam dados pertencentes a outras tabelas que eram obrigatórios para criar um utilizador, foram

inseridos um horário de trabalho padrão (‘Working Schedule’), um local de trabalho (‘WorkPlace’) e um idioma (‘Language’), como se vê na Figura 40.

```php
$workplaces = WorkPlace::create([
    'description' => 'Bragança',
    'working_schedule_id' => $workingSchedule->id
]);

$language = Language::create([
    'id' => '3',
    'description' => 'Pt',
    'created_at' => now(),
    'updated_at' => now(),
]);

$userAdmin = User::create([
    'name' => 'admin',
    'username' => 'admin',
    'email' => 'admin@megatic.pt',
    'email_verified_at' => now(),
    'password' => Hash::make('1234.abc'),
    'remember_token' => Str::random(10),
    'work_place_id' => $workplaces->id,
    'language_id' => $language->id
]);
```

Figura 40 - Exemplo do arquivo 'AdminSeeder'

O arquivo ‘InitialSeeder’, Figura 41, é um ‘seeder’ utilizado para inserir dados garantindo que certas tabelas tenham registos fundamentais para o funcionamento da aplicação, como por exemplos nas tabelas ‘coordinationTypes’, ‘taskType’, ‘IssueCause’, ‘GlobalConfig’, ‘Country’ e ‘Province’.

```php
GlobalConfig::firstOrCreate(
    ['key' => 'body_email_document'],
    [
        'value' => '<p>Caro(a) [company_name],</p><p>Venho por este meio informar que, como parte do processo em curso com a Coordenação [identificati
        este documento o mais brevemente possível para que
        possamos dar continuidade ao processamento do seu pedido de forma eficiente e sem atrasos.
        </p><p>Em caso de dúvidas ou necessidade de assistência, não hesite em contactar-nos. Estamos disponíveis para ajudar.</p><p>Agradecemos desde
        'datatype' => 'string',
        'created_at' => now(),
        'updated_at' => now(),
    ]
);
GlobalConfig::firstOrCreate(
    ['key' => 'company_signature'],
    [
        'value' => 'Ecoordina',
        'datatype' => 'string',
        'observations' => 'nome da empresa',
        'created_at' => now(),
        'updated_at' => now(),
    ]
);
GlobalConfig::firstOrCreate(
    ['key' => 'coord_client_notif_hour'],
    [
        'value' => '2',
        'datatype' => 'int',
        'observations' => 'Coordination Client Notifications (Hours)',
        'created_at' => now(),
        'updated_at' => now(),
    ]
);
```

Figura 41 - Exemplos do arquivo 'InitialSeeder'

5.2. Desenvolvimento do Frontend

O desenvolvimento *frontend* refere-se à parte visível da aplicação com a qual os utilizadores interagem diretamente. Neste subcapítulo, irá ser discutido a estrutura do *front-end*, abordando a implementação de uma interface intuitiva e responsiva. Além disso, explora-se a criação de funcionalidades essenciais, como o CRUD ('Create, Read, Update, Delete') para a entidade 'Organization'.

5.2.1. Estrutura da Front-End

Para a implementação da estrutura do *frontend* foram definidos vários diretórios e arquivos, facilitando a sua manutenção, escalabilidade e reutilização do código. Seguidamente, irão ser detalhadas as principais pastas e ficheiros que compõem o projeto, Figura 42.

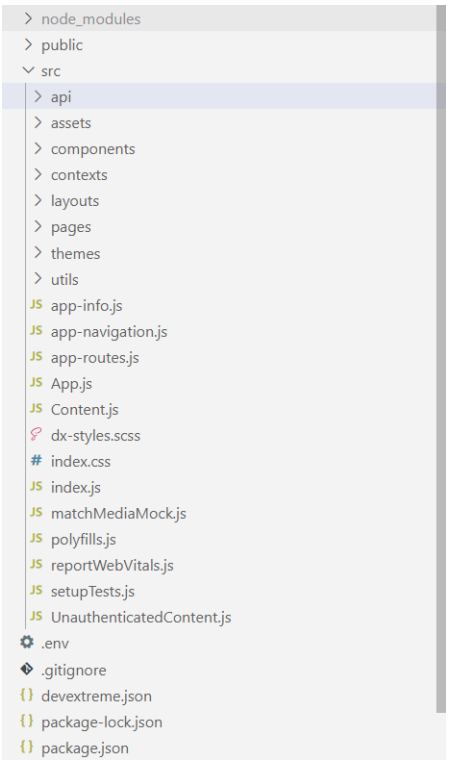


Figura 42 - Arquitetura do projeto

Pastas Principais

'api'

- A camada ‘api’, Figura 43, é responsável por organizar e centralizar todas as interações com a API da aplicação. Ela armazena funções e métodos que realizam pedidos HTTP (GET, POST, PUT, DELETE) e gerem a comunicação com a ‘backend’.

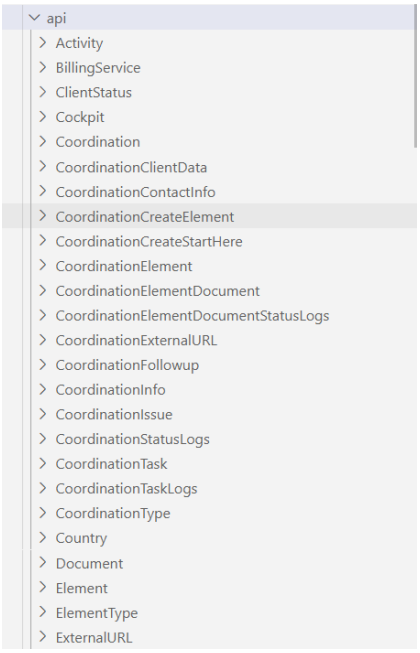


Figura 43 - Pasta ‘api’

‘assets'

- A pasta ‘assets’ armazena recursos estáticos como imagens, ícones, fontes e outros ficheiros que não são componentes de código, mas são utilizados na interface da aplicação.

  Exemplo de Conteúdo: Logos da empresa, ícones SVG, arquivos de fontes e imagens de fundo.

‘components’

- A pasta ‘components’ contém todos os componentes reutilizáveis da aplicação, como botões, formulários, alertas, rodapés, entre outros. Esses componentes são modulares e devem ser projetados para serem reutilizados em diferentes partes da aplicação.

‘contexts’

- A pasta ‘contexts’, Figura 44, é destinada a gerir o estado global da aplicação, especialmente no que diz respeito à autenticação e navegação. Aqui são definidos

os contextos ‘React’ que controlam o ‘login’, ‘logout’ e a lógica que decide quais componentes devem ser exibidos dependendo do estado de autenticação do utilizador.

Figura 44 - Pasta ‘contexts’

‘layouts’

- A pasta layouts define a estrutura visual da aplicação, como a disposição de menus, barras de navegação, cabeçalhos e rodapés. Esses layouts são aplicados nas páginas para garantir consistência na interface. Ela serve para garantir consistência no layout em toda a aplicação.

‘pages’

- A pasta ‘pages’, Figura 45, contém todas as páginas da aplicação. Cada ficheiro ou subpasta dentro de ‘pages’ representa uma página distinta, como a página inicial, página de login, página de perfil, etc. Esta estrutura facilita a navegação e organização das diferentes telas da aplicação.

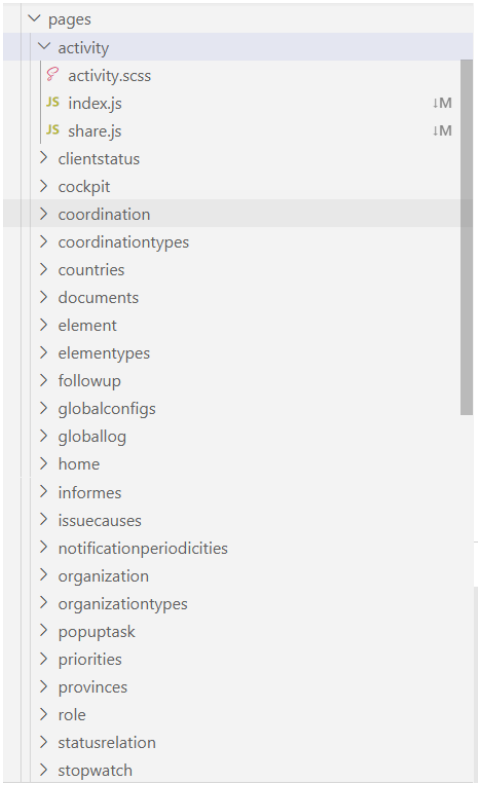


Figura 45 - Pasta 'pages'

‘themes’

- A pasta ‘themes’ contém todos os ficheiros necessários para definir e aplicar o tema principal da aplicação. Isso inclui definições de cores, fontes, espaçamentos e outros estilos globais que garantem uma aparência coerente em toda a interface.

‘utils’

- A camada ‘utils’ abriga funções utilitárias e ‘helpers’ que são amplamente utilizados ao longo do projeto. Estas funções podem incluir manipuladores de data, funções de formatação, validadores e qualquer outra lógica que seja compartilhada entre diferentes partes da aplicação.

  Exemplo de Conteúdo: Funções como formatDate, validateEmail.

Arquivos Principais

app-info

- Contém informações gerais sobre a aplicação, como configurações globais ou constantes utilizadas em todo o projeto.

app-navigation

- Define as rotas e caminhos da navegação dentro da aplicação, especificando para onde cada link deve direcionar o utilizador.

app-routes

- Responsável pela configuração das rotas no aplicativo, mapeando os caminhos definidos no 'app-navigation' para os componentes de página correspondentes.

5.2.2. Crud 'Organization'

O componente React 'Organization' desempenha um papel crucial na gestão das organizações dentro do sistema. Ele é responsável por exibir e manipular os dados das organizações através de uma interface de tabela de dados (DataGrid), fornecendo funcionalidades avançadas como procura, filtragem, exportação e navegação entre diferentes páginas relacionadas às organizações. O componente interage diretamente com várias APIs para carregar, atualizar e remover dados, garantindo que as operações de CRUD (Create, Read, Update, Delete) sejam realizadas de maneira eficiente e integrada. Na Figura 46, é exposto o 'Crud Organization' que será abordado em seguida.

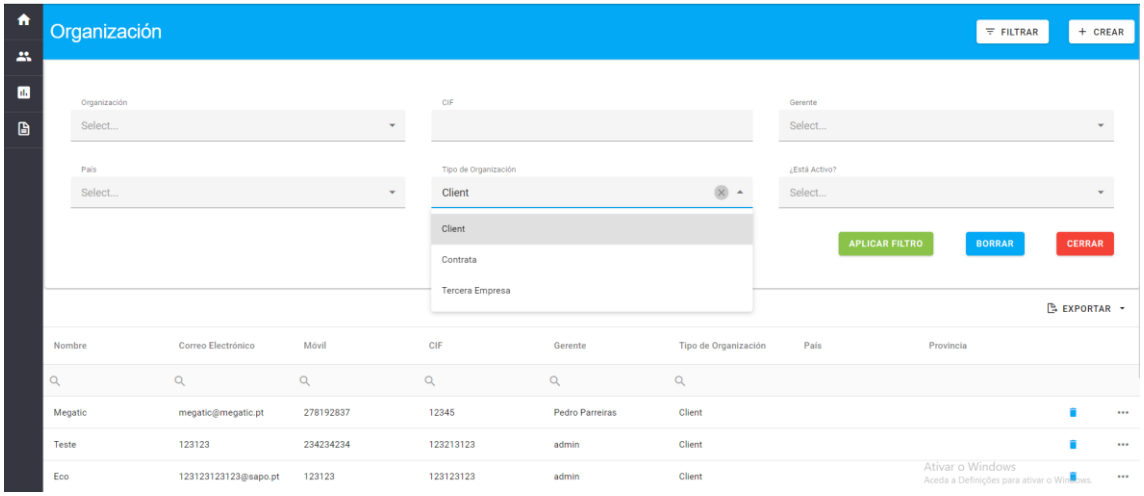


Figura 46 - Crud 'organization'

Estrutura e Funcionalidades do Componente

1. **Estrutura de Estado**

O componente utiliza o 'useState' para gerir diversos estados internos, como:

- 'data': Armazena os dados das organizações carregados da API.
- 'selectedOrganizationId': Guarda o ID da organização atualmente selecionada.

- 'filterCardVisible': Controla a visibilidade do card de filtro.
- 'loading': Indica se os dados estão a ser carregados.
- 'formData': Armazena os dados do formulário de filtro.

2. **Carregamento de Dados**

• '**useEffect**': o 'useEffect', Figura 47, é chamado para carregar os dados das 'organizations' utilizando a função 'getOrganizationForm'. Esse carregamento inicial define o estado 'data' com os dados retornados da API.

```
useEffect(() => {
  const fetchData = async () => {
    setLoading(true);
    try {
      const loadDataOptions = { organization_type_id: 2 };
      const response = await getOrganizationForm(loadDataOptions);
      setData(response.data);
    } catch (error) {
      console.error("Failed to fetch data:", error);
    } finally {
      setLoading(false);
    }
  };
  fetchData();
}, []);
```

Figura 47 - Função 'useEffect'

• A interação do componente com a API é central para seu funcionamento. As funções de API (getOrganizationForm, removedOrganization, etc.) são invocadas para carregar, e remover dados das'organizations'.

Por exemplo, através da função **'getOrganizationForm' (**Figura 48**)** é realizada uma solicitação à API para a procura de dados de uma 'organization' com base em determinados parâmetros de consulta. A função recebe um objeto 'loadOptions', que contém os parâmetros necessários para a solicitação. Após a solicitação, a função retorna um objeto contendo os dados principais da resposta ('data'), o número total de registos ('totalCount'), um resumo dos dados ('summary'), e a contagem dos grupos ('groupCount'), se aplicável.

```
export async function getOrganizationForm(loadOptions) {
  const paramNames = [
    "skip",
    "take",
    "requireTotalCount",
    "requireGroupCount",
    "sort",
    "filter",
    "totalSummary",
    "group",
    "groupSummary",
  ];

  const queryString = paramNames
    .filter((paramName) => isNotEmpty(loadOptions[paramName]))
    .map(
      (paramName) => `${paramName}=${JSON.stringify(loadOptions[paramName])}`
    )
    .join("&");

  try {
    const responseData = await axios.get(`/api/organization?${queryString}`, {
      params: {
        id: loadOptions.organization_id,
        cif: loadOptions.cif,
        is_active: loadOptions.is_active,
        organization_type_id: loadOptions.organization_type_id,
        manager_id: loadOptions.manager_id,
        agent_id: loadOptions.agent_id,
        country_id: loadOptions.country_id,
      },
    });
    return {
      data: responseData.data.data,
      totalCount: responseData.data.totalCount,
      summary: responseData.data.summary,
      groupCount: responseData.data.groupCount,
    };
  } catch (err) {
    throw new Error("Data Loading Error");
  }
}
```

Figura 48 - Função 'getOrganizationForm'

3. **Manipulação e Envio de Alterações**

• **‘onSaving’**: Este manipulador de evento é acionado quando o utilizador tenta guardar alterações na grade de dados. A função intercepta o evento, cancela a ação padrão e processa as alterações através da função ‘processBatchRequest’, Figura 49.

```
const onSaving = (e) => {
  e.cancel = true;
  if (e.changes.length) {
    e.promise = processBatchRequest(
      `/api/organization/edit/${e.changes[0].key}`,
      e.changes,
      e.component,
      e.changes[0].key
    );
  }
};
```

Figura 49 - Função 'onSaving'

● **‘processBatchRequest’**: Esta função é responsável por enviar as alterações para a API. A ‘sendBatchRequest’ é a função que interage diretamente com a API. Ela é chamada dentro de ‘processBatchRequest’.

A função verifica o tipo de alteração usando ‘changes[0].type’. Atualmente, o código lida com remover ‘organizations’ (‘remove’), mas pode ser estendido para outros tipos de mudanças, como criação ou atualização. Se o tipo de alteração for "remove", a função ‘removedOrganization’ é chamada com o ‘key’ da ‘organization’ a ser removida, e o sistema tenta remover a mesma da base de dados através da API. Após a remoção, a função ‘handleSubmit’ é chamada para recarregar os dados da ‘DataGrid’ e refletir as alterações recentes, Figura 50.

```
async function sendBatchRequest(url, changes, organizationId) {
  let organization = {};

  switch (changes[0].type) {
    case "remove":
      await removedOrganization(changes[0].key, translate);
      await handleSubmit();
      break;
    default:
      console.error("Unknown change type:", changes[0].type);
      break;
  }
}

async function processBatchRequest(url, changes, component, organizationId)
  await sendBatchRequest(url, changes, organizationId);
  await component.refresh(true);
  component.cancelEditData();
}

const onSaving = (e) => {
  e.cancel = true;
  if (e.changes.length) {
    e.promise = processBatchRequest(
      `/api/organization/edit/${e.changes[0].key}`,
      e.changes,
      e.component,
      e.changes[0].key
    );
  }
};
```

Figura 50 - Função 'onSaving', 'sendBatchRequest' e ´'processBatchRequest'

4. Navegação

● **‘**navigateToOrganizationDetails’: Função responsável por navegar para a página de detalhes de uma ‘organization’ selecionada, utilizando o ‘useNavigate’.

● ‘navigateToOrganizationCreate’: Similarmente, esta função navega para a página de criação de uma nova ‘organization’.

Essas duas funções podem ser vistas na Figura 51.

```
const navigateToOrganizationDetails = useCallback(
  (OrganizationId) => {
    setSelectedOrganizationId(OrganizationId);
    navigate(`/organization/details/${OrganizationId}`);
  },
  [navigate]
);

const navigateToOrganizationCreate = useCallback(() => {
  navigate(`/organization/create`);
}, [navigate]);
```

Figura 51 - Funções 'navigate'

5. Procura e Filtragem

A funcionalidade de filtragem e pesquisa é essencial para permitir que os utilizadores encontrem e visualizem organizações específicas com base em critérios definidos.

Para criar um formulário que captura os critérios de filtragem, utiliza-se o componente 'Form' da biblioteca 'devextreme-react'. Cada campo do formulário (organization_id, cif, manager_id, country_id, etc.) é configurado para ser um 'SelectBox' ou um campo de texto, permitindo que o utilizador selecione ou insira valores para filtrar. Quando o utilizador interage com o formulário, os dados são atualizados no estado 'formData' por meio do manipulador 'onFieldDataChanged', ver na Figura 52.

```
{filterCardVisible && (
  <div className={"content dx-card responsive-paddings"}>
    <Form
      ref={formRef}
      formData={formData}
      onFieldDataChanged={({ component }) =>
        setFormData(component.option("formData"))
      }
    >
      <Item itemType="group" colCount={3}>
        <Item
          dataField="organization_id"
          label={{ text: translate("Organization") }}
          editorType="dxSelectBox"
          editorOptions={{
            searchEnabled: true,
            showClearButton: true,
            dataSource: organizationStore,
            valueExpr: "id",
            searchMode: "contains",
            searchExpr: "company_name",
            minSearchLength: minSearchLength,
            onInput: onInput,
            displayExpr: "company_name",
            showDataBeforeSearch: true,
          }}
        />

        <Item
          dataField="cif"
          label={{ text: translate("CIF") }}
          value="cif"
        />

        <Item
          dataField="manager_id"
          label={{ text: translate("Manager") }}
          editorType="dxSelectBox"
          editorOptions={{
```

Figura 52 - Manipulador 'onFieldDataChanged'

Em seguida, o botão ‘Apply Filter’ (ou ‘Aplicar Filtro’) aciona a função ‘handleSubmit’. A função ‘handleSubmit’ (Figura 53) é responsável por coletar os dados do formulário de filtro, preparar os parâmetros necessários e, em seguida, atualizar a fonte de dados (‘dataSource’) da ‘DataGrid’ com os filtros aplicados.

```
const handleSubmit = async () => {
  setLoading(true);
  try {
    const {
      organization_id = "",
      cif = "",
      manager_id = "",
      agent_id = "",
      country_id = "",
      is_active = "",
      organization_type_id = "",
    } = formData;

    const loadDataOptions = {
      organization_id,
      manager_id,
      cif,
      agent_id,
      country_id,
      is_active,
      organization_type_id,
    };

    await fetchData(loadDataOptions);
    setFormSubmitted(true);
  } catch (error) {
    console.error("Error loading data:", error);

    setFormSubmitted(false);
  } finally {
    setLoading(false);
  }
};
```

Figura 53 - Função 'handleSubmit'

6. Exportação de Dados

● ‘**onExporting’**: Esta função permite a exportação dos dados da grade em formatos PDF ou Excel. Para PDFs, utiliza a biblioteca jsPDF, enquanto para Excel, utiliza exceljs.

Após a explicação detalhada do CRUD da entidade 'Organization', considera-se desnecessário repetir o mesmo nível de detalhamento para as demais entidades, uma vez que seguem uma estrutura semelhante. Assim, opta-se por não abordar cada operação de CRUD individualmente. No entanto, o CRUD foi implementado para as entidades ‘Priorities’, ‘logs’ globais do sistema, configurações globais, além de alguns relatórios específicos como ‘tempos de dedicação’, ‘Tags’, e ‘Tasks Type’. Os resultados obtidos para essas e outras entidades estão detalhados nos Anexos C, D, E, F, G e H, respetivamente.

6. Conclusões

Neste capítulo final, são apresentadas as reflexões sobre o estágio e o projeto desenvolvido, conectando-se diretamente com os objetivos e enquadramento inicial descritos na introdução do relatório. O foco principal é avaliar em que medida as metas definidas para o estágio foram cumpridas e destacar os conhecimentos adquiridos ao longo do processo.

6.1. O projeto

O projeto desenvolvido ficou na fase de teste para os utilizadores da empresa para a qual foi implementada. Sobre este projeto resta aferir que foram cumpridos todos os requisitos apresentados pelo cliente.

A utilização de um sistema de controlo de versões foi também muito importante permitindo reverter versões quando necessário e ter um servidor central onde todo o código fica registado. Além disso foi fundamental também na formação do estagiário no que toca a adquirir competências no trabalho em equipa.

Com o desenvolvimento deste projeto foi possível colocar em prática todos os conhecimentos relativos a desenvolvimento web. Também várias e novas competências foram adquiridas, quer na área anteriormente referida quer em outras.

Desta forma pode-se concluir que o projeto desenvolvido neste estágio foi enriquecedor do ponto vista da aquisição de conhecimento e experiência adquirida.

6.2. O estágio

Durante este estágio, os conhecimentos aplicados e adquiridos superaram significativamente aqueles previamente mencionados nas conclusões do projeto. A experiência de trabalho em equipa, o uso de metodologias estabelecidas e a aplicação de procedimentos compartilhados por colegas mais experientes foram extremamente valiosos para o desenvolvimento profissional.

Trabalhar em conjunto com outras pessoas proporcionou uma troca rica de experiências e conhecimentos. As reuniões da empresa desempenharam um papel crucial ao permitir a revisão de estratégias e a consideração de diferentes perspetivas, o que resultou em melhorias contínuas no trabalho realizado. Ao longo do desenvolvimento do projeto, o aconselhamento do orientador da empresa foi essencial para superar desafios, além de ajudar na definição de metas e estratégias.

Assim, concluir-se que o estágio foi altamente produtivo para a formação profissional, oferecendo uma experiência enriquecedora em um ambiente empresarial onde os projetos são executados por equipas colaborativas.

6.3. Considerações finais

Desta forma é possível concluir que este estágio se revelou uma experiência extremamente valiosa e enriquecedora para o meu desenvolvimento profissional. Os vários desafios enfrentados, tanto no domínio das tecnologias quanto nas metodologias de trabalho e colaboração em equipa, desempenharam um papel fundamental no sucesso alcançado.

O exercício do pensamento crítico e da capacidade analítica foi central ao longo do estágio, com essas competências sendo continuamente aprimoradas. A habilidade de identificar problemas, analisá-los e propor soluções eficazes foi testada repetidamente, contribuindo significativamente para o crescimento profissional.

Além disso, o estágio proporcionou a oportunidade de desenvolver competências complementares, como a criação de funções e a automatização de procedimentos. Em conclusão, o estágio foi concluído com êxito, tanto em termos dos trabalhos realizados quanto da aprendizagem adquirida.

Bibliografia

Brigantia Ecopark. (25 de Setembro de 2024). Obtido de Brigantia Ecopark: https://www.brigantia-ecopark.pt/

CTAIMA. (15 de Setembro de 2024). Obtido de CTAIMA: https://www.ctaima.pt/

DevExtreme. (4 de Setembro de 2024). Obtido de DevExtreme: https://js.devexpress.com/React/

Git. (25 de Setembro de 2024). Obtido de Git: https://git-scm.com/

GitLab. (25 de Setembro de 2024). Obtido de GitLab: https://about.gitlab.com/

Laravel. (4 de Setembro de 2024). Obtido de Laravel: https://laravel.com/docs/11.x

Megatic. (4 de Setembro de 2024). Obtido de Megatic: https://www.megatic.pt/

M-Files. (15 de Setembro de 2024). Obtido de M-Files: https://www.m-files.com/

Microsoft Teams . (25 de Setembro de 2024). Obtido de Microsoft Teams : https://www.microsoft.com/pt-pt/microsoft-teams/log-in

MySQL Workbench. (25 de Setembro de 2024). Obtido de MySQL Workbench: https://www.mysql.com/products/workbench/

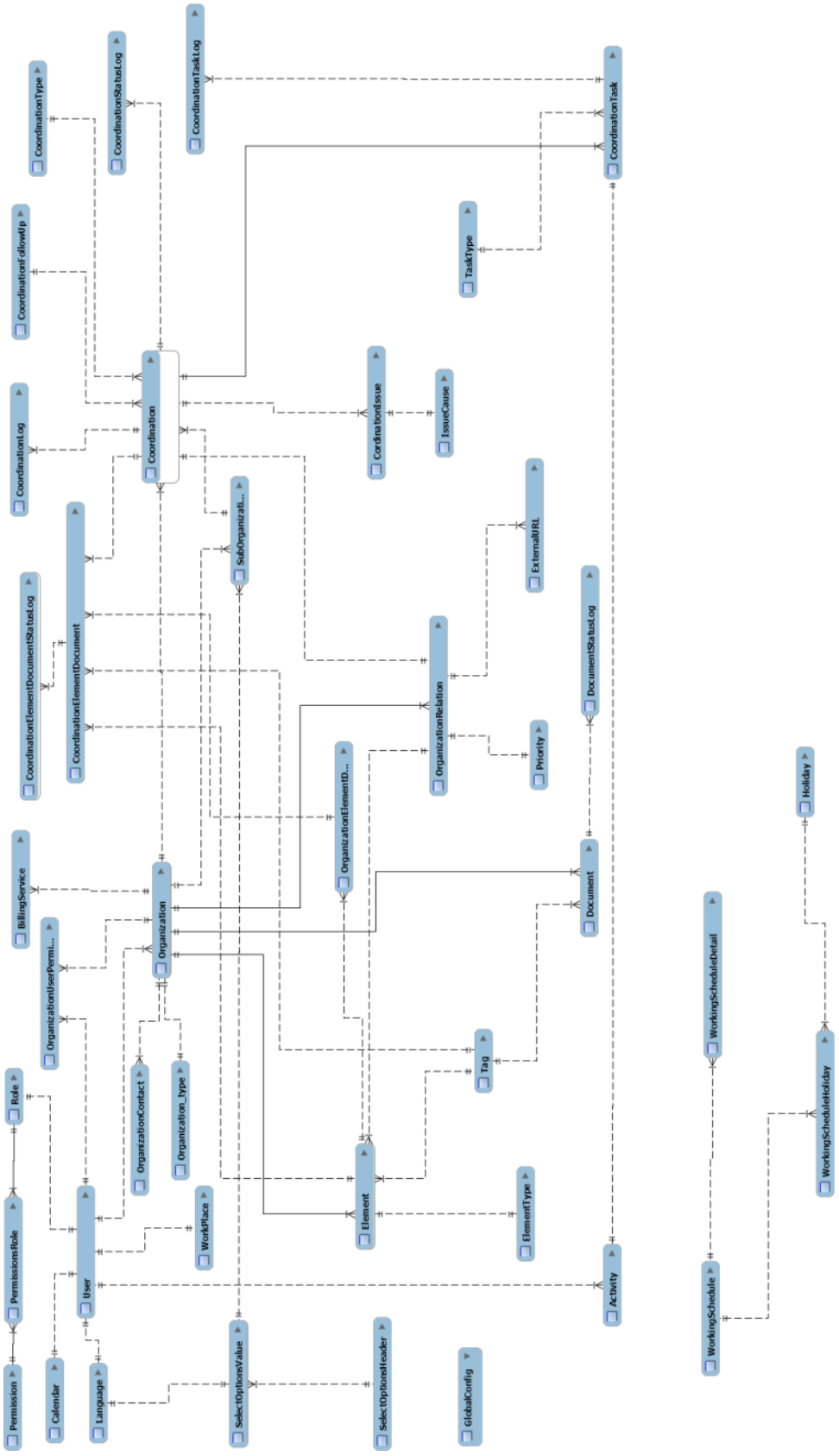
PhpMyAdmin. (14 de Setembro de 2024). Obtido de PhpMyAdmin: https://www.phpmyadmin.net/

Postman. (19 de Setembro de 2024). Obtido de Postman: https://www.postman.com/

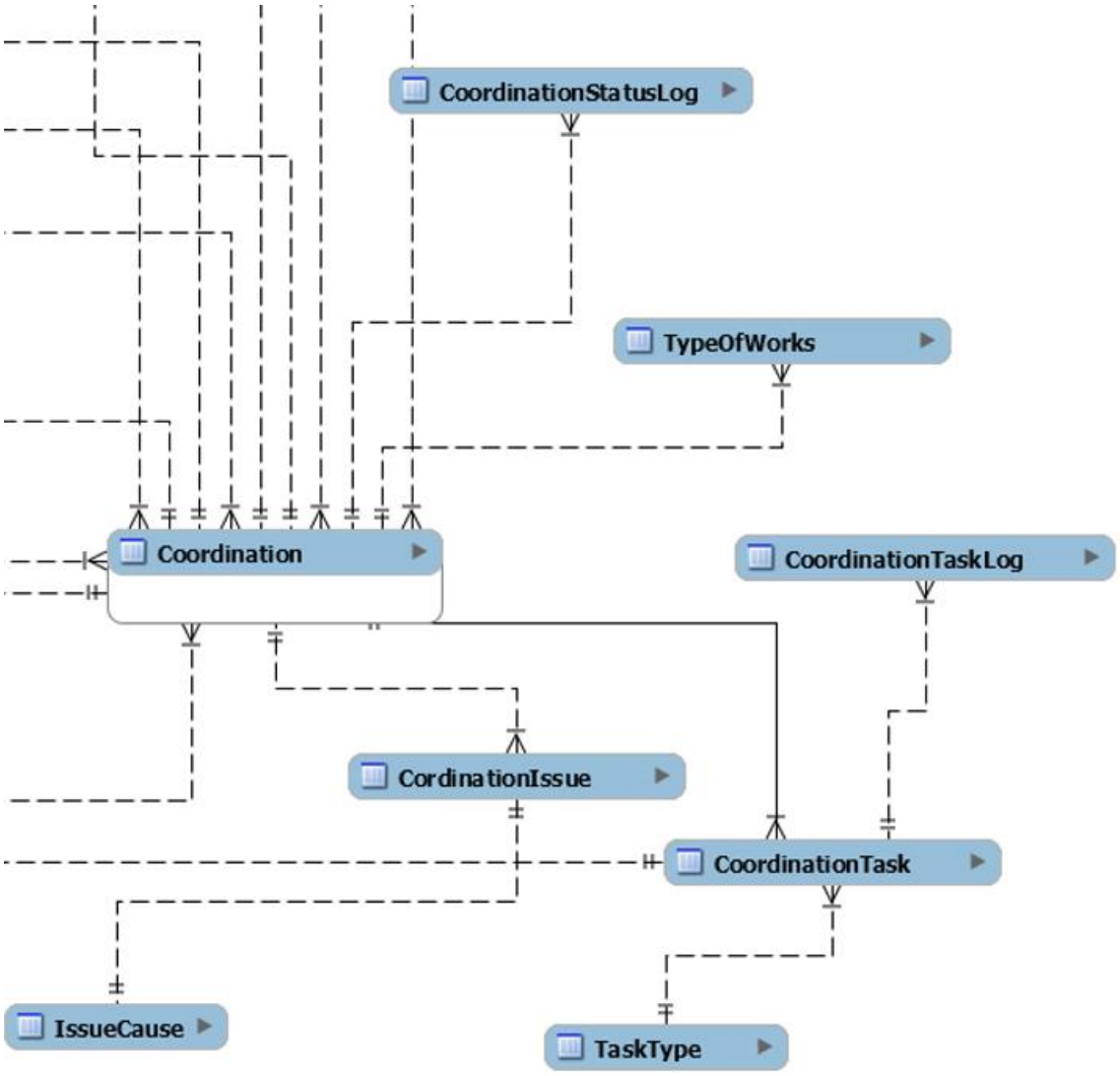
React. (4 de Setembro de 2024). Obtido de React: https://react.dev/learn

Visual Studio Code. (14 de Setembro de 2024). Obtido de Visual Studio Code: https://code.visualstudio.com/

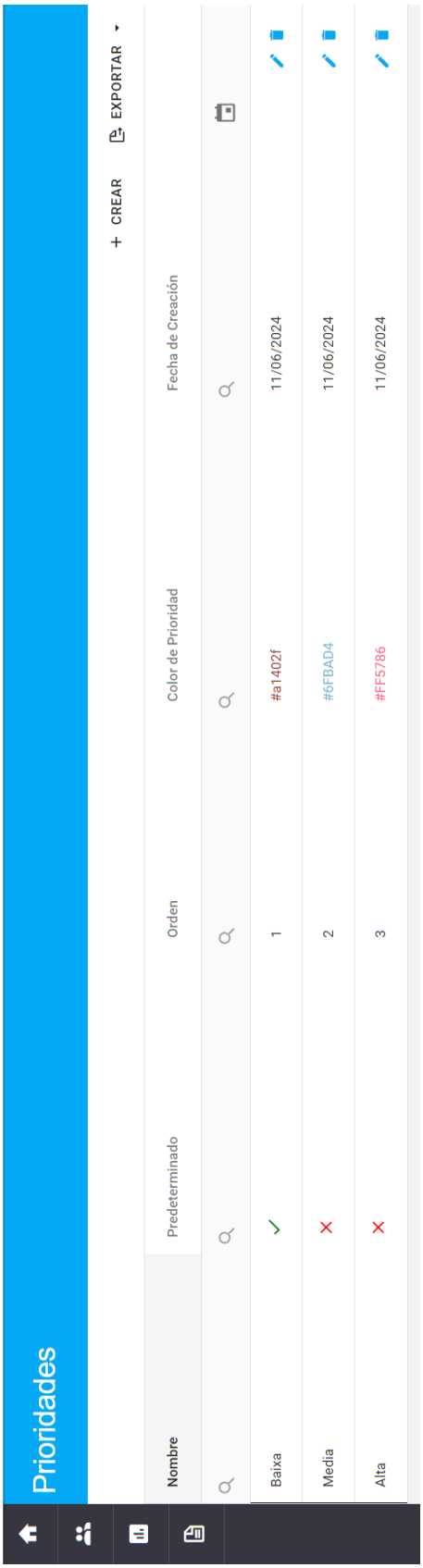
Anexo A


CoordinationType
CoordinationStatusLog
CoordinationTaskLog
CoordinationTask
CoordinationFlowHp
TaskType
CoordinationLog
Coordination
CordinationIssue
IssueCause
SubOrganizati...
ExternalURL
CoordinationElementDocumentStatusLog
CoordinationElementDocument
OrganizationRelation
DocumentStatusLog
Priority
OrganizationElementD...
BillingService
Organization
Document
OrganizationUserPermi...
Holiday
WorkingScheduleDetail
Tag
Role
OrganizationContact
Organization_type
WorkingScheduleHolday
Element
PermissionsRole
User
WorkPlace
ElementType
Activity
WorkingSchedule
Permission
Calendar
Language
SelectOptionsValue
SelectOptionsHeader
GlobalConfig

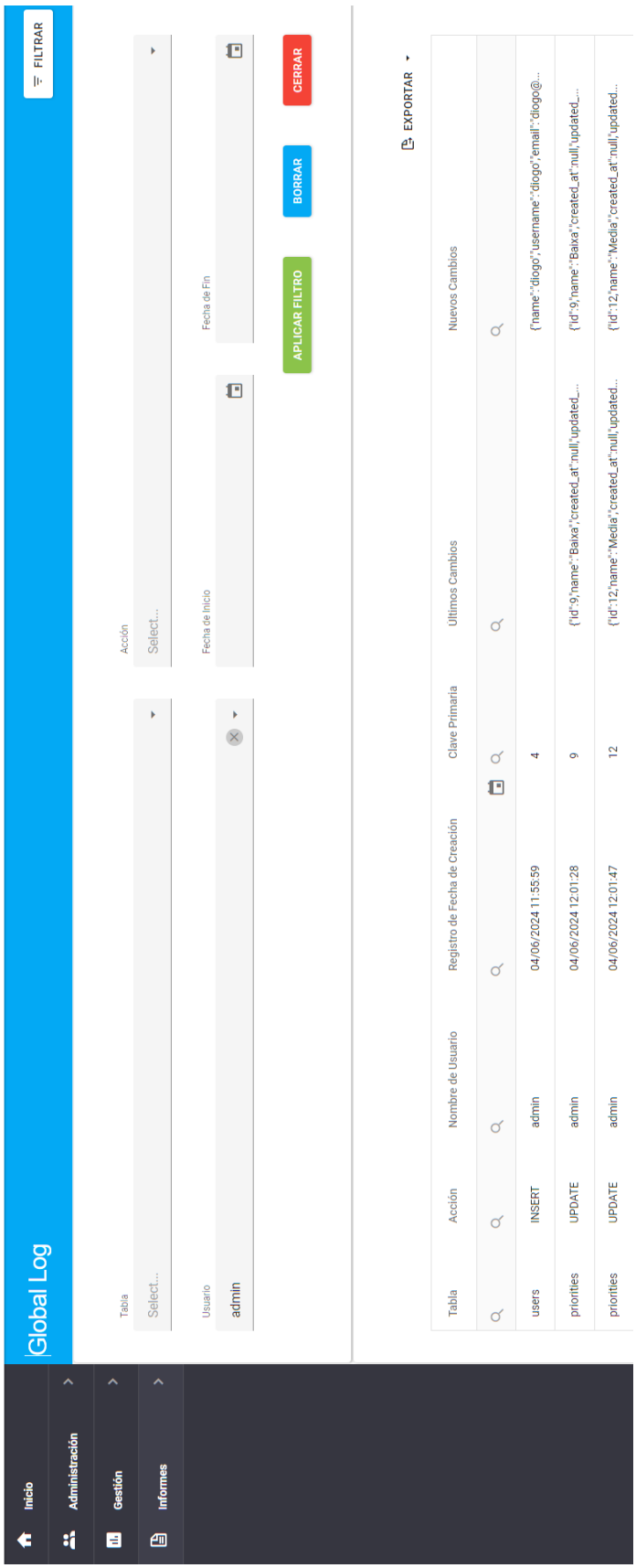
Anexo B


CoordinationStatusLog
TypeOfWorks
Coordination
CoordinationTaskLog
CordinationIssue
CoordinationTask
IssueCause
TaskType

Anexo C


Prioridades
+ CREAR
EXPORTAR
Nombre
Predeterminado
Orden
Color de Prioridad
Fecha de Creación
Baixa
1
#a1402f
11/06/2024
Media
2
#6FBAD4
11/06/2024
Alta
3
#FF5786
11/06/2024

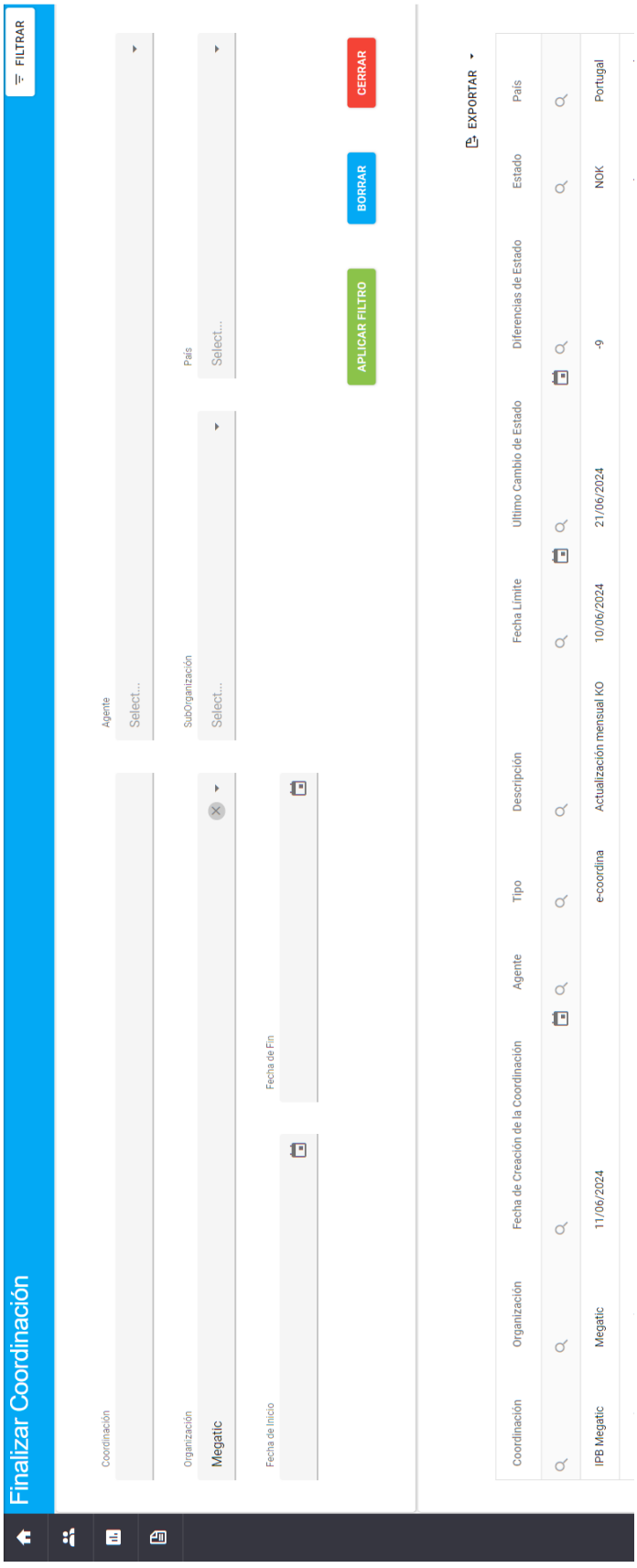
Anexo D


Inicio
Administración
Gestión
Informes
Global Log
FILTRAR
Tabla
Select...
Acción
Select...
Usuario
admin
Fecha de Inicio
Fecha de Fin
APLICAR FILTRO
BORRAR
CERRAR
EXPORTAR
Tabla
Acción
Nombre de Usuario
Registro de Fecha de Creación
Clave Primaria
Últimos Cambios
Nuevos Cambios
users
INSERT
admin
04/06/2024 11:55:59
4
{"name":"diogo","username":"diogo","email":"diogo@...
priorities
UPDATE
admin
04/06/2024 12:01:28
9
{"id":9,"name":"Baixa","created_at":null,"updated_...
{"id":9,"name":"Baixa","created_at":null,"updated_...
priorities
UPDATE
admin
04/06/2024 12:01:47
12
{"id":12,"name":"Media","created_at":null,"updated...
{"id":12,"name":"Media","created_at":null,"updated...

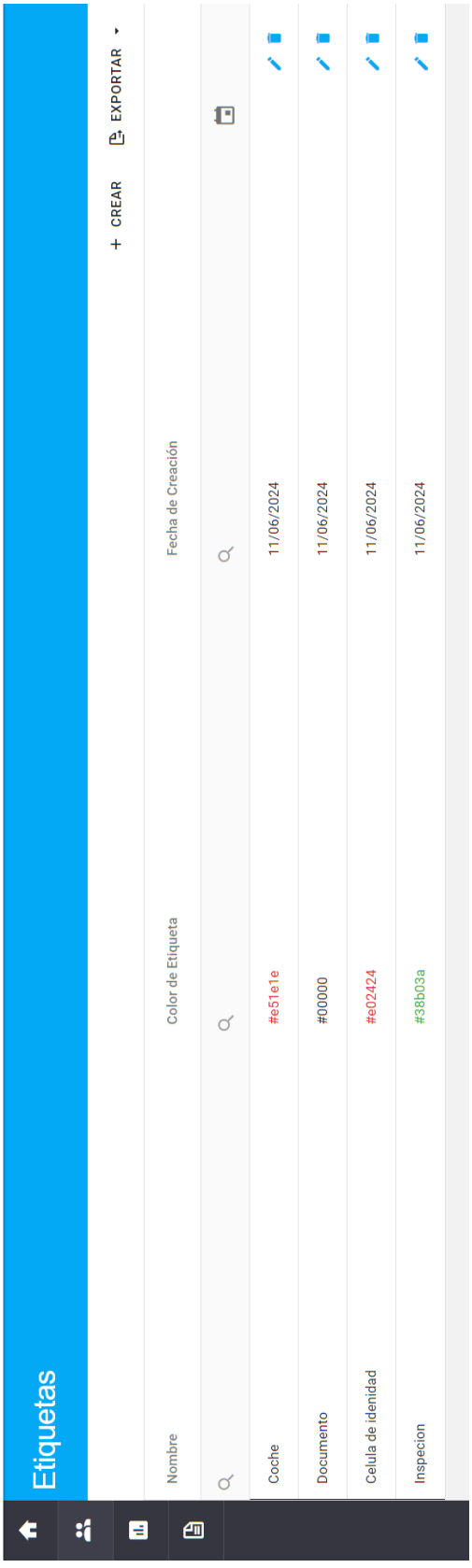
Anexo E

Configuraciones Globales
EXPORTAR
Clave
body_email
body_email_document
company_signature
coord_client_notif_hour
coord_client_notif_task_types
day_job_coordination
deadline
Configuraciones Globales
Clave
body_email_document
Observaciones
Tipo de Datos
string
Valor
Caro(a) [company_name], Venho por este meio informar que, como parte do processo em curso com a Coordenação [identification], necessitamos do documento [name] solicitado. Por favor, providencie
SAVE
CANCEL

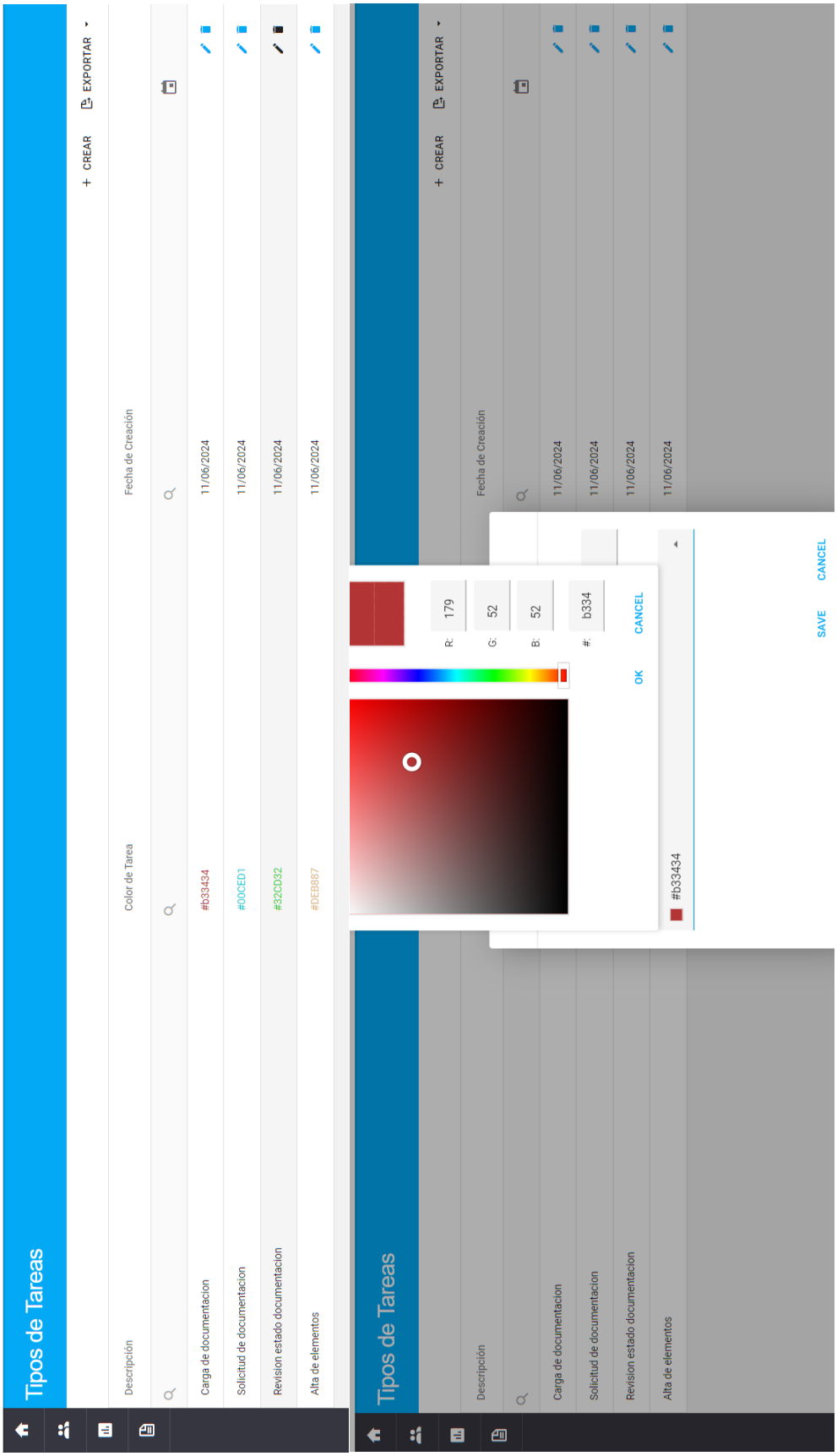
Anexo F


Finalizar Coordinación
FILTRAR
Coordinación
Agente
Select...
Organización
Megatic
SubOrganización
Select...
País
Select...
Fecha de Inicio
Fecha de Fin
APLICAR FILTRO
BORRAR
CERRAR
EXPORTAR
Coordinación
Organización
Fecha de Creación de la Coordinación
Agente
Tipo
Descripción
Fecha Límite
Ultimo Cambio de Estado
Diferencias de Estado
Estado
País
IPB Megatic
Megatic
11/06/2024
e-coordina
Actualización mensual KO
10/06/2024
21/06/2024
-9
NOK
Portugal

Anexo G


Etiquetas
+ CREAR
EXPORTAR
Nombre
Color de Etiqueta
Fecha de Creación
Coche
#e51e1e
11/06/2024
Documento
#00000
11/06/2024
Celula de idenidad
#e02424
11/06/2024
Inspecion
#38b03a
11/06/2024

Anexo H


Tipos de Tareas
+ CREAR
EXPORTAR
Descripción
Color de Tarea
Fecha de Creación
Carga de documentacion
#b33434
11/06/2024
Solicitud de documentacion
#00CED1
11/06/2024
Revision estado documentacion
#32CD32
11/06/2024
Alta de elementos
#DEB887
11/06/2024
R: 179
G: 52
B: 52
#: b334
OK
CANCEL
#b33434
SAVE
CANCEL