Daniel Madureira
João Luís
José Gameiro
Pedro Ramos
Rodrigo Aguiar

Dissertações

Relatório Técnico

Projeto em Informática 2023/24

4 de junho de 2024

Universidade de Aveiro
2024

Dissertações

Daniel Madureira
João Luís
José Gameiro
Pedro Ramos
Rodrigo Aguiar

Relatório Técnico apresentado à Universidade de Aveiro para cumprimento dos requisitos necessários à conclusão da unidade curricular Projeto em Informática, condição necessária para obtenção do grau de Licenciado em Engenharia Informática, realizado sob a orientação científica do Professor Doutor Diogo Nuno Pereira Gomes, Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

agradecimentos

Gostaríamos de expressar o nosso mais profundo agradecimento ao nosso orientador, Professor Diogo Gomes, por toda a orientação e partilha de conhecimento dada ao longo do semestre. Queremos agradecer também aos professores Beatriz Sousa Santos, Telmo Cunha, Paulo Dias e Samuel Silva por toda a disponibilidade demonstrada na discussão dos requisitos e testes de Usabilidade, tornando assim o nosso projeto mais forte e completo. Por último, agradecemos também a todos aqueles que contribuíram, seja através do levantamento de problemas existentes, participação nos testes de Usabilidade ou sugestões para o nosso projeto.

Palavras Chave

Correspondência entre alunos e orientadores, Escolha de Dissertações, Fluxo de trabalho de documentos

Resumo

Nos dias de hoje, o Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro gere a oferta e atribuição de dissertações/estágios através de uma plataforma antiga que já não endereça/corresponde às necessidades do Departamento e que contém funcionalidades abandonadas. Assim, a nova plataforma Dissertações, tinha como objetivos resolver estas lacunas, apresentando também novas funcionalidades que não existiam anteriormente, evitando alterações diretas no código, e apresentando ainda um novo interveniente, o Diretor de Curso. Simultaneamente, o sistema foi desenvolvido em tecnologias mais recentes, que sejam mais fáceis de manter.

Desta forma, através de um processo de levantamento de requisitos detalhado e contínuo, foi desenvolvido um sistema totalmente novo, acompanhado por uma interface moderna adaptada aos dias de hoje e adaptada ao seu uso em ecrãs mais pequenos como *smartphones*, mantendo-se na mesma fácil de usar, tanto a alunos como docentes. Aliados à plataforma, foram desenvolvidos manuais de utilização que guiam os utilizadores passo a passo pela mesma, e ainda alguma documentação orientada àqueles que queiram alterar e manter a plataforma no futuro.

Conteúdo

Lista de Figuras

Lista de Tabelas

Glossário

API	Application Programming Interface
REST	Representational State Transfer
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
PDF	Portable Document Format
JSON	JavaScript Object Notation
SSL	Secure Sockets Layer
CDN	Content Delivery Network
SQL	Structured Query Language
NoSQL	Not Only Structured Query Language
DoS	Denial of Service
XSS	Cross-site Scripting
CSRF	Cross-site Request Forgery
UI	User Interface
SUS	System Usability Scale
UML	Unified Modeling Language
MVP	Minimum Viable Product
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
URL	Uniform Resource Locator
WSO2	Web Services Oxygenated
IdP	Identity Provider
OAuth2	Open-Authorization
CA	Certificate Authority
SMTP	Simple Mail Transfer Protocol
SSO	Single Sign-On
SAML	Security Assertion Markup Language
OIDC	OpenID Connect
IUPI	Identificador Universal Pessoal Intransmissível
DETI	Departamento de Eletrónica, Telecomunicações e Informática
UA	Universidade de Aveiro
DSD	Distribuição de Serviço Docente
UC	Unidade Curricular
PACO	Portal Académico Online
STIC	Serviços de Tecnologias de Informação e Comunicação
IT	Instituto de Telecomunicações
IHC	Interação Humano-Computador
LEI	Licenciatura em Engenharia Informática
ECTS	European Credit Transfer System
VM	Virtual Machine

CAPÍTULO 1

Introdução

1.1 Contexto e Motivação

No contexto de escolha de dissertações e estágios, a Universidade de Aveiro (UA) apresenta nos dias de hoje diferentes formas de lidar com esta problemática. Se em alguns departamentos este problema é resolvido através da partilha de folhas *Excel* via secretaria ou diretor de curso, no DETI esta escolha é suportada por uma plataforma desenvolvida nos anos dois mil por um grupo de estudantes do próprio DETI.

Assim, com o passar dos anos e devido à pouca manutenção realizada, a plataforma acabou por ficar antiquada e desfasada da atualidade, com funcionalidades não usadas e com outros problemas identificados que surgiram do uso natural da própria plataforma, pelo que havia necessidade de a renovar.

1.2 Objetivos

Os objetivos e requisitos para este Projeto de Informática foram evoluindo ao longo do desenvolvimento do projeto, mas a proposta inicial consistia em:

- Extrair requisitos da plataforma existente;
- A nova plataforma seguir regras definidas de estabilidade e documentação para permitir o suporte a quem vai manter a plataforma depois da conclusão do projeto;
- A nova plataforma ter formas melhores de indexar e procurar propostas de dissertações/estágios;
- Apresentar um fluxo de trabalho bem definido para a submissão de propostas, o processo de atribuição e ainda envio das mesmas à secretaria;
- A plataforma a desenvolver ser *web* com uma API disponível, para futura expansão/automação;
- A plataforma ser flexível para suportar dissertações realizados com outros departamentos, estágios com empresas e também áreas de estudo, descrição, etc.

Além disto, havia também o objetivo de desenvolver todo o Minimum Viable Product (MVP) a tempo da abertura da época de dissertações do ano letivo 2024/25, que ocorreu no início de maio, sendo expectável que todo o DETI o viesse a usar a partir desse momento, abandonando por completo a plataforma antiga.

1.3 Estrutura do Documento

Para além da introdução, este documento apresenta mais cinco capítulos. No capítulo 2, é apresentado o estado da arte, detalhando o sistema existente. O capítulo 3 apresenta o processo de elicitação de requisitos, os requisitos do sistema e ainda a arquitetura e os seus diagramas. No capítulo 4, são descritos os detalhes de implementação tanto do frontend como backend e também outros aspetos de implementação como a segurança e *backups*. Depois, o capítulo 5 apresenta os testes de usabilidade realizados numa fase inicial do projeto, incluindo a amostra, o método e avaliação do resultados obtidos. O último capítulo (6) é uma vista geral do trabalho, as suas vantagens em relação ao sistema antigo e ainda trabalho futuro que ficou por fazer.

Por último, seguem-se os apêndices que contêm documentação e guiões usados ou produzidos durante o desenvolvimento do projeto.

CAPÍTULO 2

Estado da Arte

2.1 Distribuição de Serviço Docente

Com o objetivo de entender quais seriam os objetivos a cumprir e familiarizarmos-nos com o contexto do sistema a desenvolver, foi-nos explicada e demonstrada a plataforma Distribuição de Serviço Docente (DSD) existente, que se encontra disponível no *link* `http://sd.web.ua.pt`

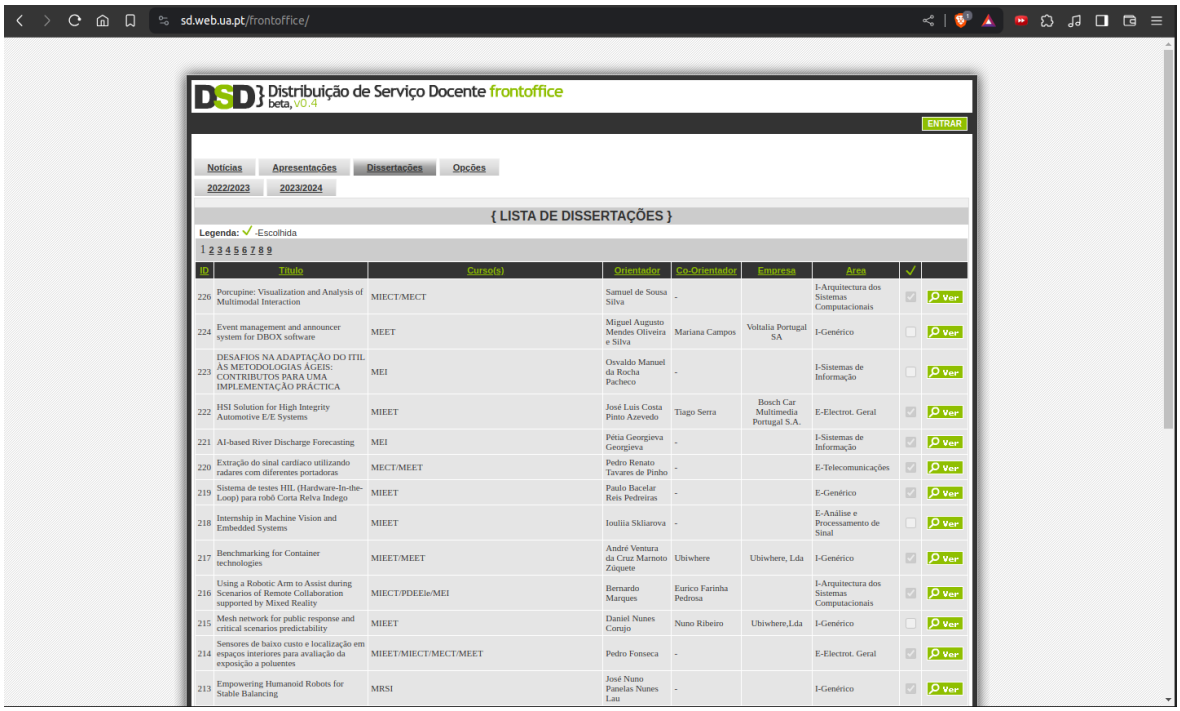


Figura 2.1: Vista da lista de dissertações na antiga plataforma

Esta é uma plataforma que ao longo dos anos tem tido manutenção mínima e, para além disso, nenhuma funcionalidade adicional foi desenvolvida, o que levantou alguns problemas:

- Vulnerabilidades de Segurança - Devido à pouca manutenção, as tecnologias usadas como o [1] que é uma *framework* usada para produzir páginas web dinâmicas do lado do

servidor, mantiveram-se em versões antigas, o que levanta cada vez mais problemas;

- Bugs inexplorados - A plataforma apresenta *bugs* conhecidos, mas resolver alguns deles, mesmo que simples, pode causar o *crash* da mesma.
- Funcionalidades Abandonadas - Para além da temática do processo de atribuição de dissertações, funcionalidades como a escolha de uma Unidade Curricular (UC) de opção, calendário das apresentações de dissertações, relatórios de estágio ou teses, reserva de salas e ainda escolha de horários por parte dos docentes, foram deixando de ser usadas em detrimento do uso do Portal Académico Online (PACO) e mais recentemente o PACO2, ou passaram a ser tratados por correio-eletrónico, o que resultou numa plataforma que apresenta cada vez mais conteúdo visível inútil e com apenas parte das suas funcionalidades ainda ativas;
- User Interface (UI) desatualizada - A plataforma apresenta uma UI com aspeto antigo, não contando com modo escuro como a maior parte dos *websites* modernos dos dias de hoje, e também filtros demasiado básicos;

Assim, a nova plataforma, cujo foco é apenas o processo de publicitação e escolha de dissertações ou estágios, deve apresentar interface e funcionalidades renovadas que permitam assistir tanto estudantes como professores nestas questões.

CAPÍTULO 3

Requisitos do Sistema e Arquitetura

3.1 Requisitos do Sistema

Esta secção apresenta a especificação dos requisitos do sistema desenvolvido. Estes foram obtidos ao longo do desenvolvimento do projeto nas várias reuniões semanais realizadas com o professor orientador do projeto, o professor Diogo Gomes, em brainstorms entre todos os elementos do grupo e também em reuniões com o administrador da plataforma existente, o professor Telmo Cunha.

Assim, nas subsecções seguintes irão ser apresentados os atores, devidamente classificados, diagramas de caso de uso, requisitos funcionais e não funcionais, e ainda pressupostos e dependências do sistema.

3.1.1 Atores

Os utilizadores alvo do nosso sistema são todos aqueles que tenham interesse no processo de publicitação e escolha de dissertações/estágios em cursos associados ao DETI. À data de escrita deste relatório, os cursos presentes no sistema encontram-se listados na tabela 3.1.

Sigla	Código Curso	Nome
MEI	9263	Mestrado em Engenharia Informática
MECT	9291	Mestrado em Engenharia de Computadores e Telemática
MEET	9295	Mestrado em Engenharia Eletrónica e Telecomunicações
MRSI	9283	Mestrado em Robótica e Sistemas Inteligentes
MCS	9281	Mestrado em Cibersegurança
MEAI	9163	Mestrado em Engenharia e Automação Industrial
MDJD	9305	Mestrado em Desenvolvimento de Jogos Digitais
MTIM	9251	Mestrado em Tecnologias de Imagem Médica
MEC	9294	Mestrado em Engenharia Computacional
MCD	9306	Mestrado em Ciência de Dados
MSCA	9307	Mestrado em Sistemas de Computação Aeroespaciais
MBC	9280	Mestrado em Bioinformática Clínica
MIECT	8240	Mestrado Integrado em Engenharia de Computadores e Telemática
MIEET	8204	Mestrado Integrado em Engenharia Eletrónica e Telecomunicações

Tabela 3.1: Lista de mestrados oferecidos pelo DETI, respetiva sigla e código de curso

É expectável que os utilizadores envolvidos tenham pelo menos alguma experiência com computadores e a navegar na Internet. No entanto, o nível de especialização que é necessário para usar o sistema sem problema é reduzido.

A vista da lista de dissertações, vista comum a todos os atores, apresenta uma secção de filtros similar a *websites* de uso comum, contando ainda com uma barra de pesquisa mais direcionada para *power users*.

A nossa plataforma conta com a adição de dois atores novos: o Diretor de curso, que está envolvido em todas as propostas que mencionem o seu curso, e a secretaria, que agora tem o estado de todas as propostas de dissertação/estágio à distância de um clique, que gera um ficheiro Excel.

O administrador conta também com bastantes mais funcionalidades à sua disposição relativamente à antiga plataforma. Para além de gerir as áreas, cursos e os seus diretores, anteriormente mencionados, consegue ainda adicionar novos administradores e monitorizar alguns aspetos da plataforma como verificar o estado das APIs envolvidas e verificar quotas para os professores.

Os atores estão descritos na lista seguinte:

- **Estudante**: Utiliza o sistema como utilizador potencial dos serviços prestados por esse sistema;
- **Professor**: Utiliza o sistema como utilizador potencial dos serviços prestados por esse sistema;
- **Diretor de Curso**: Responsável por gerir as propostas relacionadas com o seu curso;
- **Administrador**: Entidade com mais autoridade no sistema. Responsável por verificar se o mesmo se encontra a funcionar corretamente. É o contacto intermediário entre os outros atores;
- **Secretaria**: Faz uso do sistema para obter o estado de todas as propostas;

3.1.2 Casos de Uso

A figura 3.1 apresenta os casos de uso principais para a nossa plataforma:

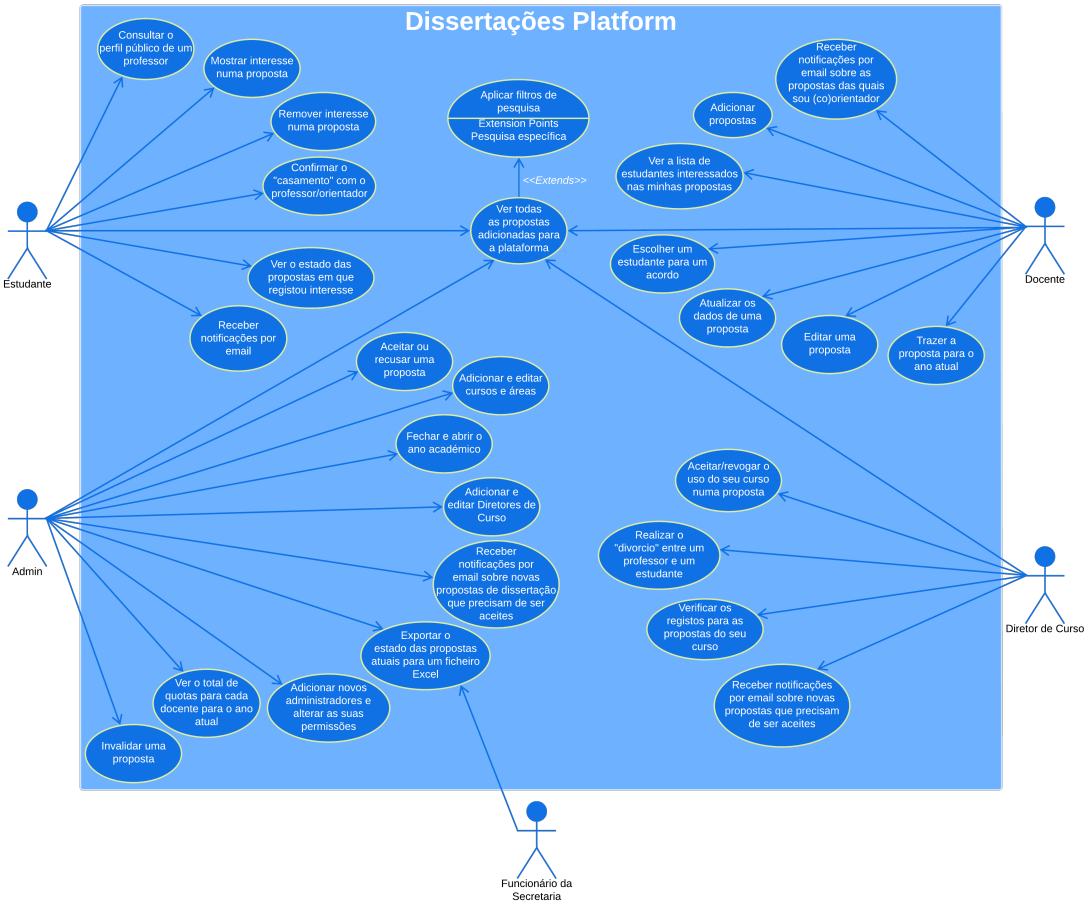


Figura 3.1: Diagrama UML que descreve os casos de uso

De seguida, apresentamos uma descrição mais detalhada de cada caso de utilização, separada por ator:

- **Estudante**:
  1. Ver todas as propostas adicionadas para a plataforma - Ver todas as propostas de dissertação/estágio sob a forma de uma lista, com a ajuda de filtros e uma barra de pesquisa, para chegar mais rapidamente a propostas específicas;
  2. Consultar o perfil publico de um professor - Através do *email* de um docente, ser possível consultar todas as propostas submetidas na plataforma por aquele docente;
  3. Mostrar interesse numa proposta - Selecionar uma proposta e registar-se como interessado na mesma;
  4. Confirmar o "casamento"com o professor/orientador - Após a aceitação do professor, confirmar o interesse na proposta, assinando o acordo;
  5. Remover interesse numa proposta - Remover o interesse numa proposta à qual foi anteriormente mostrado interesse;

6. Ver o estado das propostas em que registou interesse - Manter-se a par do estado das propostas em que mostrei interesse;
7. Receber notificações por *email* - Receber notificações por correio eletrónico sobre o estado das propostas em que demonstrei interesse;

- **Professor**:
  1. Ver todas as propostas adicionadas para a plataforma - Ver todas as propostas de dissertação/estágio sob a forma de uma lista, com a ajuda de filtros e uma barra de pesquisa, para chegar mais rapidamente a propostas específicas;
  2. Trazer a proposta para o ano atual - Transferir uma proposta que não tenha sido atribuída a um estudante num ano académico anterior para o ano académico atual;
  3. Adicionar propostas - Adicionar uma proposta de dissertação/estágio, composta por um ficheiro e metadados (título, descrição, etc...);
  4. Ver a lista de estudantes interessados nas minhas propostas - Poder ver os alunos que mostraram interesse numa proposta para de seguida escolher um aluno para trabalhar na mesma;
  5. Escolher um estudante para um acordo - Finalizar o processo confirmando um aluno para trabalhar numa das propostas por mim submetidas;
  6. Atualizar os dados de uma proposta - Ser capaz de atualizar os dados de uma proposta, como a descrição, o título, o ficheiro, etc;
  7. Receber notificações por *email* sobre as propostas das quais sou (co)orientador - Receber notificações por correio eletrónico sobre o estado das minhas propostas, as que sou co-orientador e os alunos que mostraram interesse nas mesmas;
- **Diretor de Curso**:
  1. Ver todas as propostas adicionadas à plataforma - Ver todas as propostas de dissertação/estágio sob a forma de uma lista, com a ajuda de filtros e uma barra de pesquisa, para chegar mais rapidamente a propostas específicas;
  2. Aceitar/revogar o uso do seu curso numa proposta - Depois de uma determinada proposta ter sido *uploaded* e aceite pelo administrador, o diretor de curso pode aceitar/recusar a utilização do seu curso nessa proposta específica;
  3. Realizar o "divorcio"entre um aluno e um estudante - Depois do aluno e o professor concordarem em quebrar o acordo anterior, o diretor do curso pode confirmar essa intenção;
  4. Verificar os registos para as propostas do seu curso - Verificar que propostas estão marcadas com aquele curso e a data a que foram aceites pelo administrador nos últimos 31 dias;
  5. Receber notificações por *email* sobre novas propostas que precisam ser aceites - Receber notificações por correio eletrónico sobre propostas que tenham marcado o curso daquele diretor e que precisam de ser aceites;

- **Administrador**:
    1. Ver todas as propostas adicionadas para a plataforma - Ver todas as propostas de dissertação/estágio sob a forma de uma lista, com a ajuda de filtros e uma barra de pesquisa, para chegar mais rapidamente a propostas específicas;
    2. Aceitar ou recusar uma proposta - Aceitar ou recusar uma proposta de dissertação/estágio, dependendo da qualidade da mesma e de outros aspetos que influenciam a decisão de a aceitar ou declinar;
    3. Gerar uma lista de propostas, respetivos orientadores e estudantes - Gerar uma lista de todas as propostas, respetivos orientadores e estudantes que estão a trabalhar nelas;
    4. Adicionar e editar cursos e áreas;
    5. Fechar e abrir o ano académico - Poder abrir e fechar a época, de modo a que não possam ser acrescentadas mais propostas para o respetivo ano letivo;
    6. Adicionar e editar Diretores de curso;
    7. Receber notificações por correio *email* sobre novas propostas de dissertações que precisam de ser aceites - Receber notificações por correio eletrónico sobre novas dissertações que foram apresentadas, para as aprovar ou declinar;
    8. Exportar o estado das propostas atuais para um ficheiro Excel;
    9. Adicionar novos administradores e alterar as suas permissões - Atribuir novos administradores para ajudar na gestão do sistema, mas não com as mesmas permissões que o administrador original;
    10. Ver o total de quotas para cada docente para o ano atual;
    11. Invalidar uma proposta - Depois de uma proposta ser aceite pelo administrador e/ou diretores de curso, a proposta pode voltar a ser enviada para o docente, para o mesmo a editar;

3.1.3 Requisitos Funcionais

Dos requisitos obtidos durante a fase de apuramento foram definidas algumas funcionalidades essenciais:

Gestão da dissertação/estágio:

- O sistema deve permitir aos professores carregar/editar/eliminar propostas de dissertação/estágio;
- O sistema deve permitir que os alunos mostrem interesse em várias dissertações e aceitem apenas uma;
- O sistema deve permitir que os administradores aceitem ou recusem as propostas dos professores;
- O sistema deve permitir que os professores adicionem propostas de dissertação/estágio e aceitem alunos para as mesmas.

Gestão do utilizador:

- O sistema deve permitir aos estudantes verificar e gerir o estado das propostas em que mostrou interesse e fases seguintes;
- O sistema deve permitir aos professores gerir as suas propostas, verificar os alunos interessados e aceitá-los para as suas propostas;
- O sistema deve permitir aos administradores gerir a época das dissertações e gerir os utilizadores do sistema.

Autenticação do Utilizador:

- O sistema deve fornecer um mecanismo de *login* utilizando o IdP da UA através de Open-Authorization (OAuth2);
- O sistema deve restringir o acesso dos utilizadores a determinadas funcionalidades com base na sua função (por exemplo, aluno, professor, diretor de curso, administrador);
- O sistema deve efetuar várias verificações com base nos atributos do utilizador para garantir que apenas os utilizadores autorizados podem aceder a determinadas funcionalidades.

Pesquisa:

- A funcionalidade de pesquisa deve permitir aos utilizadores procurar dissertações/estágios com base em vários critérios, tais como título, orientador, cursos, áreas, data de publicação;
- O sistema deve oferecer opções avançadas de pesquisa, como filtros e ordenação.

Sistema de notificações:

- O sistema incluirá um serviço de notificações por correio eletrónico para alertar os utilizadores sobre eventos importantes, tais como novas propostas de dissertação/estágio para aprovação, aceitação de propostas e alterações no estado de uma proposta.

Automatização do fluxo de trabalho:

- O sistema deve automatizar os fluxos de trabalho de aprovação das propostas, garantindo que estas são revistas e aprovadas pelas partes adequadas antes de serem publicadas;
- Devem ser enviadas notificações aos utilizadores relevantes quando uma proposta é submetida, aprovada ou rejeitada.

3.1.4 Requisitos não Funcionais

A seguir, é apresentada uma lista que especifica requisitos não funcionais, ou seja, uma descrição de uma propriedade ou caraterística que um sistema de software deve apresentar ou de uma restrição que deve respeitar [2].

Performance:

- O sistema deve manter tempos de resposta consistentes à medida que a base de utilizadores ou o volume de dados aumenta;
- O sistema deve ser capaz de tratar um número crescente de transações ou pedidos por unidade de tempo.

Escalabilidade:

- O sistema deve acomodar um volume crescente de dados sem degradação significativa do desempenho;
- O sistema deve escalar horizontalmente, distribuindo as cargas de trabalho por vários servidores ou nós;
- O sistema deve escalar verticalmente, permitindo a adição de mais recursos a um único servidor;
- Garantir que escalar verticalmente proporciona um aumento proporcional do desempenho.

Disponibilidade e tolerância a falhas

- Em caso de falha de um servidor, o sistema deve redirecionar automaticamente o tráfego para um servidor disponível num curto espaço de tempo, para minimizar a interrupção do serviço;
- O sistema deve atingir um tempo de atividade mínimo de 99,99 % em qualquer período consecutivo de 30 dias, excluindo as janelas de manutenção programadas;
- O sistema deve manter-se disponível e responder em caso de falhas de componentes ou aumento de carga;
- Deve haver capacidade suficiente para acomodar a falha de serviços sem afetar o desempenho.

Balanceamento de Carga

- O sistema deve distribuir os pedidos de entrada de forma uniforme pelos vários servidores, para garantir uma utilização equilibrada dos recursos;
- Os balanceadores de carga devem escalar sem problemas para lidar com o aumento do tráfego de rede.

Partição e integridade de dados

- O sistema deve ser capaz de fragmentar os dados de forma eficaz para os distribuir por vários nós;
- O sistema deve aplicar mecanismos completos de validação dos dados introduzidos, para garantir que só são aceites dados válidos e corretamente formatados.

Manutenibilidade

- O sistema deve ser concebido com uma arquitetura modular, facilitando atualizações e modificações independentes de componentes específicos, sem afetar toda a base de código;
- Todos os módulos e funções de código devem ser adequadamente documentados, fornecendo explicações claras sobre o seu objetivo, entradas, saídas e utilização para ajudar os programadores durante a manutenção;
- Deve ser desenvolvido um manual de utilizador para as diferentes entidades ativas no sistema, explicando todo o *workflow* associado ao processo de publicação e atribuição de uma proposta de dissertação/estágio, clarificando outros termos "técnicos"associados ao assunto em causa;
- As definições de configuração do sistema devem ser armazenadas em bases de dados facilmente modificáveis e bem organizadas, reduzindo a necessidade de alterações ao código para configurar atualizações;
- Implementar *logging* de todos os eventos, erros e avisos significativos do sistema;
- Os *logs* devem incluir *timestamps* para facilitar a análise dos eventos ao longo do tempo;
- *Logs* devem ser armazenados centralmente para facilitar a monitorização e análise.

Usabilidade e Interface do Utilizador

- Incluir na interface do utilizador, elementos comuns ao mesmo, como menus e botões de navegação, que devem estar colocados e ter um comportamento consistentes em toda a aplicação, reduzindo a carga cognitiva dos utilizadores;
- Minimizar o *load* dos elementos de UI, de forma a obter respostas rápidas às interações dos utilizadores;
- Fornecer mensagens de erro claras e concisas que ajudem os utilizadores a compreender os problemas e os orientem para ações corretas;
- Incorporar nos sítios devidos ajuda *inline* e *tooltips* para fornecer informações ou orientações adicionais sem sobrecarregar a interface;
- Realizar testes de usabilidade para validar que a interface desenvolvida é de uso e aprendizagem fácil, e que passa no *System Usability Scale (SUS)*.

Portabilidade

- Os utilizadores irão aceder à aplicação web através de uma ligação à Internet. Assim, o sistema deve ser acedível, operável e consistente para os vários web *browsers* existentes.
- O sistema deve adaptar-se a ecrãs de tamanho mais reduzido, como ecrãs de telemóveis;

3.1.5 Pressupostos e Dependências

Para implementar este projeto, temos de ter em conta algumas considerações:

- É pressuposto haver mínima ligação à Internet para interação com a plataforma. Em caso de não haver conexão e esta for restabelecida antes do tempo de autenticação expirar, os dados permanecerão guardados no *frontend* se a página não for atualizada;

- O sistema está dependente que a autenticação com o IdP da UA fornecida pelos Serviços de Tecnologias de Informação e Comunicação (STIC) esteja funcional;
- O sistema está dependente que a máquina onde o mesmo se encontra no Instituto de Telecomunicações (IT) opere sem problemas e de forma contínua.

3.2 Arquitetura do Sistema

Esta secção apresenta uma visão geral da arquitetura do sistema, as tecnologias na qual está assente, uma descrição do modelo do domínio e o diagrama de instalação.

3.2.1 Arquitetura Geral

De seguida, é apresentada a arquitetura geral do sistema. Esta arquitetura é representada no diagrama 3.2. Serão apenas apontados os 7 módulos fundamentais que constituem a estrutura do sistema, sem especificar as tecnologias utilizadas nem detalhar extensivamente as suas funções.

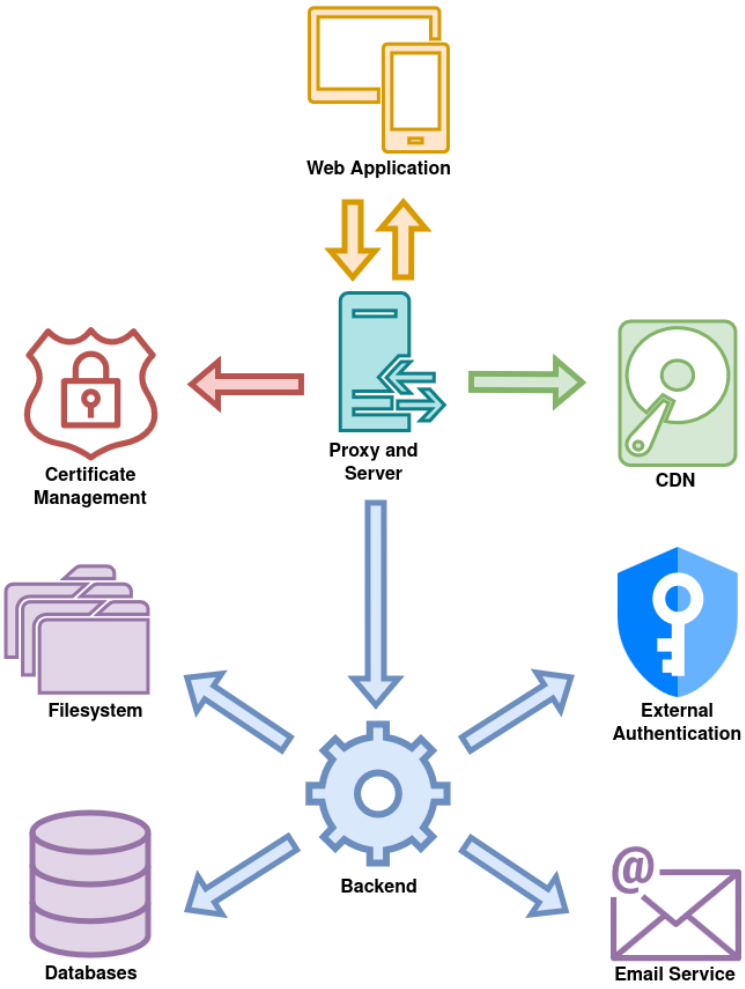


Figura 3.2: Arquitetura Geral do Sistema

- **Gestão de pedidos ao servidor**: *Proxy and Server + Certificate Management*

  Os pedidos ao servidor, quer sejam pedidos à API, páginas web ou conteúdos estáticos, são primeiro recebidos pelo *proxy and server*, que se encarrega de analisar os pedidos de modo geral.

Este componente tem a capacidade de rejeitar o mais cedo possível pedidos mal formados ou com anexos de tamanho demasiado grande, de forma a prevenir que mais esforço computacional seja gasto por outros módulos do sistema a processar o pedido.

Para além disto, este componente está encarregue de reencaminhar os pedidos aceites para os restantes componentes do sistema, de forma a simplificar e uniformizar todos os pedidos Hypertext Transfer Protocol (HTTP) necessários para o sistema executar as suas funções.

O *Proxy* tem também a capacidade de armazenar e devolver as respostas a pedidos equivalentes e repetidos, uma vez mais libertando os restantes serviços do gasto computacional de responder a um pedido que já tinha sido respondido de igual forma previamente (*caching* é o nome desta funcionalidade).

Finalmente, este módulo está também responsável por providenciar e anexar os certificados Secure Sockets Layer (SSL) do sistema em todas as respostas do mesmo. Estes certificados são gerados pelo componente de *Certificate Management* periodicamente.

- **Distribuição de conteúdo estático**: *Proxy and Server + Content Delivery Network (CDN)*

  O serviço *Content Delivery Network* permite aceder a ficheiros estáticos através de um pedido simples ao servidor, permitindo assim ao nosso serviço hospedar e utilizar conteúdo estático como, por exemplo, imagens e logótipos, sem a necessitar de depender de outros serviços Web externos para armazenar e distribuir estes ficheiros.

  Tal como seria de esperar, a função de *caching* do *Proxy and Server*, mencionada anteriormente, beneficia bastante o tempo de resposta e a disponibilidade deste componente.

- **Gestão de pedidos de páginas web**: *Web Application*

  O serviço *Web Application* providencia uma interface gráfica ao utilizador final, de forma a facilitar a utilização da plataforma. Este serviço é composto por páginas web compiladas para HyperText Markup Language (HTML) e é executado na máquina do utilizador (Client-Side).

  Estas páginas da *Web Application* estão encarregues de comunicar com os restantes serviços por via de pedidos HTTP, como por exemplo, comunicação com os serviços de autenticação externa (IdP), o CDN e a API desenvolvida, quando tal é pedido pelo utilizador final.

  Após receber as respostas destes serviços, os dados de retorno são então processados pelas páginas de forma a distribuir graficamente o estado atual da plataforma e das suas alterações através da máquina do utilizador.

- **Application Programming Interface**: *Backend*

  Este componente disponibiliza o serviço computacional interno do sistema, estando encarregue de manipular, persistir e responder a pedidos externos, ou da interface do utilizador de forma a resolver os problemas propostos pela aplicação. Para além disto, este componente está responsável pela comunicação e manutenção dos serviços de persistência de dados.

Finalmente, este componente também realiza a comunicação com os serviços externos de gestão de *logins* e utilizadores (o serviço de *External Authentication*) e a comunicação com o serviço externo de *emails*.

- **Sistema de Persistência de dados**: *Databases + FileSystem*

  Visto que o sistema opera sobre um conjunto diverso de modelos de dados, sejam ficheiros Portable Document Format (PDF) ou estruturas de utilizadores altamente variáveis, foi implementado um conjunto de bases de dados diferentes para aproveitar os benefícios e funcionalidades de cada uma, e introduzir uma boa separação de acesso aos dados, ajudando também a prevenir ocorrências de perda de informação.

  Estas bases de dados têm como objetivo principal garantir a persistência dos modelos de dados e disponibilizar uma interface que permita a manipulação destes, garantindo a consistência da informação.

- **Sistemas externos**: *Email Service + External Authentication*

  Como mencionado anteriormente na subsecção 3.1.3, dois dos requisitos propostos no desenvolvimento deste projeto foram a integração do sistema com o *login* universal da UA e também com o seu sistema interno de gestão de correio eletrónico.

  Estes sistemas externos permitem um aumento do nível de funcionalidades do sistema sem necessitar de informação adicional por parte do utilizador, e, por isso, são uma parte integral do mesmo.

- **Scripts e Logging**:

  Finalmente, foram implementados um conjunto de *scripts* e *loggers* manuais que periodicamente executam funções de manutenção e *backups* no sistema completo.

  Estes *scripts* foram desenvolvidos dentro do próprio ambiente de produção de modo a serem facilmente editáveis, e não causarem conflitos com as restantes partes do sistema, não sendo integrais para o funcionamento correto do mesmo, mas sim para rapidamente ajudarem na correção e restauro do serviço caso existam problemas com a implementação atual.

3.2.2 Modelo de Tecnologia

No que diz respeito ao modelo de tecnologia do sistema, este está representado na figura 3.3, onde foram identificadas as seguintes tecnologias:

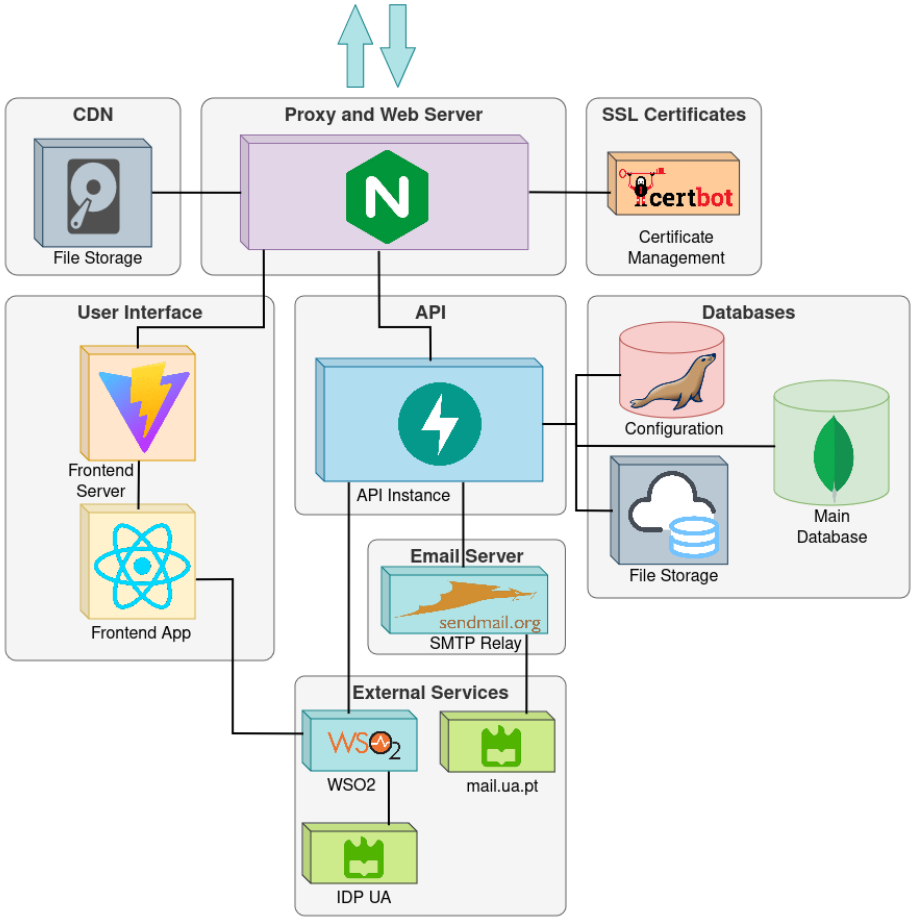


Figura 3.3: Modelo de Tecnologia

- ***Proxy and Server***: NGINX [3]

  Os serviços de *proxy*, *server*, *caching* e *parsing* de pedidos HTTP foram implementados com recurso à aplicação NGINX. Esta aplicação foi escolhida pois oferece um grande conjunto de funcionalidades com todos os serviços acima indicados e com uma configuração simples de perceber. As funcionalidades desta aplicação também recorrem a um número relativamente baixo de recursos físicos da máquina onde está implementada, libertando o resto dos recursos para uso pelos restantes serviços.

- **Certificados SSL**: Certbot [4]

  Tal como grande em parte das aplicações web, o uso de Hypertext Transfer Protocol Secure (HTTPS) e a gestão dos certificados associados é essencial para aumentar a segurança e a confiança do utilizador num serviço. Assim, foi implementado um serviço *certbot* que permite criar e atualizar as chaves SSL utilizadas com recurso à Certificate Authority (CA) *Let's Encrypt*. Este serviço foi escolhido pois permite a fácil manutenção

destas chaves de forma automática, com recurso aos *scripts* mencionados na subsecção 3.2.1.

- **CDN**: Distribuição de conteúdo estático pelo NGINX

  Para a distribuição de ficheiros e conteúdos estáticos de forma eficiente e com tempos de resposta rápidos, o serviço de *Server* do NGINX principal foi utilizado para aceder e servir os ficheiros guardados no volume estático público do sistema de ficheiros. Esta implementação pode ser considerada como um "sub-serviço"do NGINX principal, utilizando também o seu robusto sistema de *caching*.

- **Frontend Server**: Vite [5]

  Para compilar e servir as páginas web do projeto, foi usado o serviço do Vite com configurações específicas de *Server-Side Rendering*. Estas configurações permitem que as páginas web sejam geradas internamente apenas uma vez, quando a aplicação está a ser compilada, aumentando a fiabilidade da mesma e reduzindo altamente a carga computacional na máquina do utilizador, a troco de um maior tempo de *startup* do serviço e de uso de recursos locais.

- **Frontend App**: React [6]

  Para criar a plataforma web optamos por usar React por diversos motivos. Por um lado, é uma das bibliotecas de *JavaScript (JS)* mais usada na indústria e devido ao uso e popularidade que tem, existem inúmeras bibliotecas que complementam e enriquecem a plataforma. Por outro lado, ao desenvolver a plataforma com React foi possível criar uma base de código mais dinâmica, uma vez que os componentes são reutilizáveis e facilmente escaláveis. Isto permite que a base de código seja altamente manutenível e legível, um dos requisitos iniciais de maior importância do projeto proposto.

  Ao usar React, foi possível usarmos ainda recursos nativos da biblioteca, como o uso de *hooks* e *conditional rendering*, que nos permitiram criar interfaces mais dinâmicas.

- **API**: FastAPI [7]

  Para a computação e manipulação de dados no *backend*, foi necessário criar uma API Representational State Transfer (REST) que consiga integrar todos os restantes serviços, responder atempadamente a uma grande quantidade de pedidos e, acima de tudo, que seja facilmente manutenível pelos futuros administradores da plataforma.

  Um dos maiores requisitos iniciais do projeto foi a utilização de componentes facilmente alteráveis por qualquer administrador do sistema, sendo que a linguagem de programação Python foi a escolha para a implementação deste serviço. Esta linguagem permite também usufruir de uma enorme coleção de bibliotecas e recursos para reduzir ainda mais a complexidade final deste serviço.

  A *framework* FastAPI foi escolhida por estas razões. Esta *framework* permite criar APIs baseadas no sistema de tipos base de Python, permitindo que o código seja extremamente fácil de perceber, manter e alterar. O tempo de resposta de uma API que use esta *framework* é bastante baixo, devido ao uso da biblioteca *Uvicorn* [8], que permite o uso de várias *threads* assíncronas para responder a vários pedidos de forma eficaz e simultânea. Usando a biblioteca *Pydantic* [9], que permite a serialização e

validação de informação, acabamos por obter código redundante reduzido e de pouca complexidade.

- **Bases de Dados**: MongoDB, MariaDB e Volume do sistema de ficheiros

  Tal como mencionado na subsecção anterior, esta aplicação manipula uma grande quantidade de ficheiros de dados diferentes, quer sejam ficheiros PDF, imagens, modelos de utilizadores ou até configurações internas. É de notar que alguns destes dados são de caráter altamente volátil, sendo que muitos dos modelos podem conter a informação distribuída de forma diferente, dependendo da versão da aplicação usada, e que novas versões devem ser compatíveis com modelos anteriores. Para tal, foi decidido implementar vários sistemas de base de dados para aproveitar ao máximo os pontos fortes de cada sistema.

  Para guardar as configurações gerais da aplicação e *magic numbers*, como por exemplo, o número máximo da quota de um docente por ano ou a localização dos volumes onde se encontram os ficheiros PDF, foi utilizada a base de dados Structured Query Language (SQL) MariaDB [10]. Esta base de dados permite uma alta disponibilidade e performance, tal como uma boa segurança contra potenciais falhas. MariaDB também permite uma fácil migração para as bases de dados internas da Universidade de Aveiro, caso seja preciso no futuro, visto que esta é completamente compatível com as bases de dados MySQL utilizadas.

  Para persistir os dados dos modelos, como por exemplo, os campos de uma proposta de dissertação ou as informações de um utilizador, foi utilizada a base de dados MongoDB [11]. Esta base de dados implementa um modelo de execução Not Only Structured Query Language (NoSQL), na qual os dados são guardados num documento como ficheiros JavaScript Object Notation (JSON) em vez das típicas tabelas encontradas nas bases de dados SQL. Visto que os modelos de dados utilizados contêm muitos campos que apenas se encontram em alguns dos seus elementos, como por exemplo, apenas algumas propostas de dissertações tem o campo "empresa", o uso de uma base de dados SQL iria levar a um grande número de espaço gasto em valores vazios. Para além disso, a adição de novos campos ou a remoção de outros iria necessitar que os valores antigos fossem adaptados para a nova tabela, o que seria um trabalho penoso e que facilmente levaria a erros na adaptação. A base de dados MongoDB permite ainda uma fácil integração com a biblioteca Pydantic mencionada anteriormente, sendo que *queries* complexas são mais fáceis de integrar, e permite também uma rápida inserção de dados sem a necessidade da repetição dos mesmos, como aconteceria numa base de dados SQL normal. Estas funcionalidades vêm com custo de tempos de pesquisa maiores e dificuldade na integração com outras bases de dados existentes na Universidade de Aveiro.

  Finalmente, para guardar ficheiros PDF, Portable Network Graphics (PNG) e Joint Photographic Experts Group (JPEG), foi utilizado um volume do sistema de ficheiros que permite uma rápida inserção e a utilização das ferramentas pré-existentes do mesmo para a indexação e pesquisa rápida entre os ficheiros guardados, tal como a execução fácil de anti-vírus e a alteração manual de ficheiros sem ter de alterar as bases de

dados. Ao utilizar o sistema de ficheiros, também não existe a necessidade de codificar os ficheiros inseridos para poderem ser inseridos numa base de dados e, no futuro, poderá ser utilizado um sistema *Cloud* para guardar os mesmos sem a necessidade de alterar qualquer outro serviço manualmente (apenas o valor correto na base de dados de configurações).

- ***Email Server***: Sendmail [12] e mail.ua.pt

  Para enviar notificações por correio eletrónico, foi utilizado o serviço interno do Sendmail. A máquina de produção final, tal como muitas outras hospedadas internamente na Universidade de Aveiro, utilizam este serviço para comunicarem com o serviço de *emails* Simple Mail Transfer Protocol (SMTP) local. Apesar deste serviço não seguir a filosofia principal deste projeto (novas tecnologias com alta manutenibilidade), o serviço Sendmail é robusto o suficiente para funcionar corretamente sem problemas após ser implementado. Este serviço permite uma autenticação básica com o servidor de *emails* interno da UA SMTP, tal como o envio de correio eletrónico de forma rápida, com a desvantagem de ser relativamente inseguro e difícil de configurar. A insegurança deste protocolo provêm da implementação do mesmo pela própria Universidade, sendo que na sua configuração atual, não é necessário qualquer registo do nome do endereço a utilizar e qualquer conta pode receber mensagens de contas não verificadas, ou seja, é possível simular endereços de emails com nomes de remetente parecidos a, por exemplo, contas da secretaria, o que pode levar a ataques de "phishing".

- **Autenticação de Utilizadores**: Web Services Oxygenated (WSO2) e IdP UA

  Visto que esta plataforma é para o uso interno dentro da Universidade de Aveiro, um dos requisitos é a integração completa do sistema de autenticação IdP UA, utilizando o sistema de autenticação e identidade do WSO2. Com este serviço, é-nos permitido aceder a vários parâmetros internos dos utilizadores de forma a decidir quais as permissões dos mesmos. Dado esta autenticação ser altamente variável, uma vez que pedidos de novos atributos dos utilizadores podem demorar vários meses até serem aceites, todos os outros serviços têm de ser facilmente adaptados para utilizar os valores devolvidos pelo sistema de autenticação. Um exemplo é a disponibilização do número de European Credit Transfer System (ECTS) de um aluno, que está planeado para o futuro mas tal atributo ainda não foi disponibilizado.

3.2.3 Modelo do Domínio

Tal como mencionado anteriormente, os modelos do domínio são altamente variáveis, sendo que os atributos apresentados nesta secção refletem o estado final destes no momento em que o projeto final foi entregue. Sendo assim, poderão ter ocorrido alterações dos mesmos desde a criação deste relatório, mas o modelo do domínio será na generalidade parecido à figura 3.4 abaixo representada.

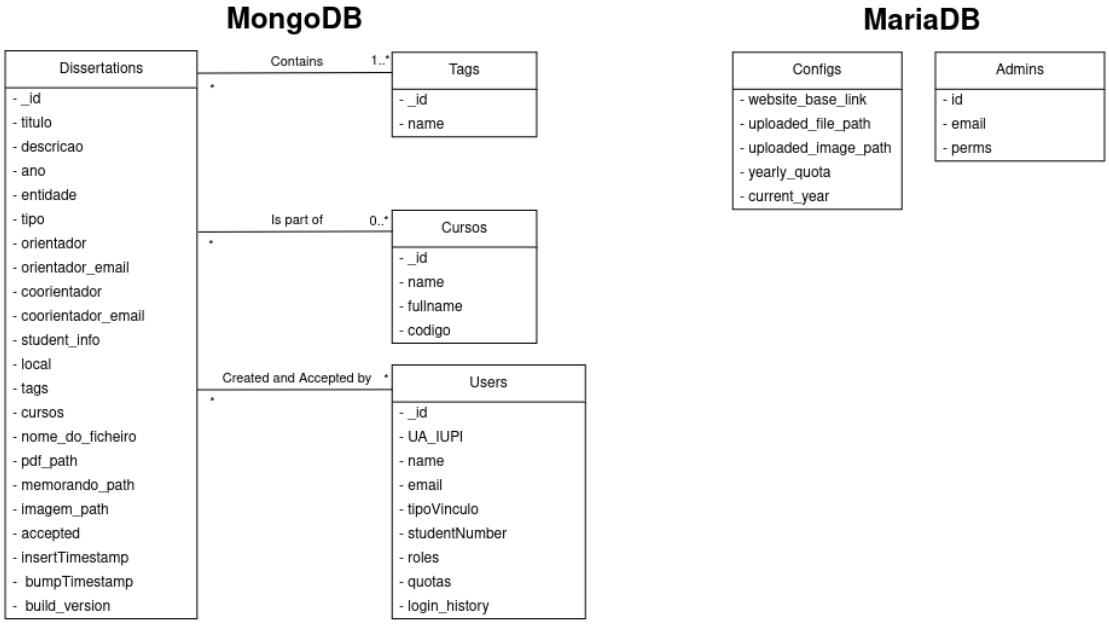


Figura 3.4: Modelo do domínio

- **MongoDB**: Dissertations, Users, Cursos e Tags (Áreas)

  Na base de dados MongoDB, são persistidos os modelos das propostas de dissertação, dos utilizadores, das tags e dos cursos. As Tags, também conhecidas como Áreas, são objetos simples compostos apenas por um nome, que servem para distinguir os conteúdos abordados em cada dissertação, como por exemplo, "Teoria da Computação"e "Análise e Processamento de Sinal".

  Os Cursos representam os diversos Mestrados existentes no DETI e são caracterizados por um nome (abreviatura), o seu nome completo e o código de curso respetivo, como por exemplo, "MEI - Mestrado em Engenharia Informática - 9263".

  Os Users são compostos pelos dados fornecidos pelo IdP UA e por uma API interna do DETI, tal como pelas ações que realizam dentro da aplicação. Nem todos os utilizadores têm todos os atributos apresentados, como por exemplo um aluno ter atributos de quotas de docentes.

  Por último, as Dissertations são compostas por todos os dados que podem existir para uma determinada proposta, incluindo os valores inseridos pelos Docentes, os cursos e Tags, os Orientadores e Coorientadores e os ficheiros associados à proposta, tal como muitos outros atributos.

- **MariaDB**: Configurações e Administradores

  Na base de dados MariaDB, são persistidos as configurações, "magic numbers"e a listagem dos administradores do sistema (com as suas permissões). Nas Configurações, persiste o link base do servidor, os caminhos para os volumes de imagens e ficheiros das propostas, a quota máxima por docente e ainda o ano académico atual.

  Nos Administradores, estão listados quais os *emails* pertencentes a cada administrador tal como uma listagem de permissões desse administrador, como por exemplo, aceitar dissertações ou atualizar o ano académico.

3.2.4 Diagrama de Instalação

Na Figura 3.5, é possível observar o diagrama de instalação desenvolvido para o projeto.

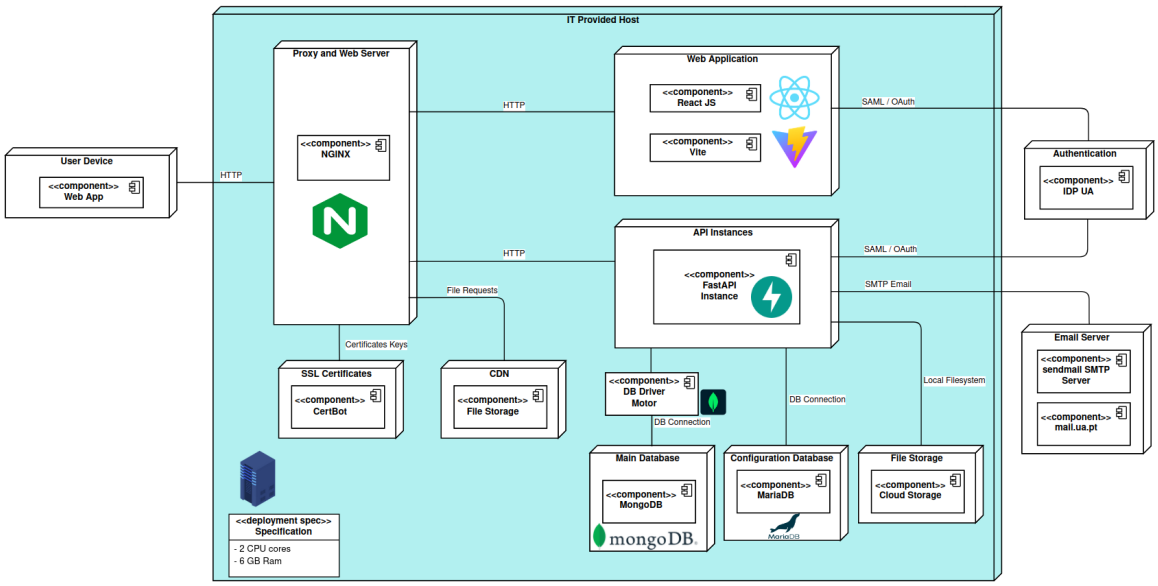


Figura 3.5: Diagrama de Instalação

- **Docker** [13]: Separação dos diversos componentes do sistema num conjunto de *containers*.

  Para a separação dos diversos componentes, foram criados diversos *containers*, sendo que cada um alberga vários componentes relacionados entre si. Na sua versão mais recente até ao dia de escrita deste relatório, a aplicação contém os seguintes *containers*:

  - **App**: Este *container* inicializa os serviços Proxy and Server, Frontend Server, Frontend App e CDN.

    Aqui, a aplicação web é compilada inicialmente para páginas web pelo Frontend Server, que depois são servidas pelo Proxy and Server. Uma execução semelhante ocorre para o CDN, em que os seus ficheiros são também servidos pelo Proxy and Server.
  - **API Instance**: Este *container* contém a instância principal da API da aplicação. Os pedidos recebidos pelo Proxy and Server relevantes são redirecionados para este *container*, onde os dados são depois processados e a resposta é devolvida.

  - **MongoDB**: Este *container* inicializa e mantém a base de dados MongoDB. Os pedidos de acesso e manipulação à base de dados por parte da API são realizados através deste *container*.
  - **MariaDB**: Este *container* inicializa e mantém a base de dados MariaDB. Os pedidos de acesso à base de dados por parte da API são realizados através deste *container*.
  - **CertBot**: Este *container* inicializa a aplicação certbot. A geração dos ficheiros SSL necessários para a validação do domínio para uso de HTTPS é realizada periodicamente, a pedido da próprio máquina (*Host*).

- **Docker Compose**: Orquestração dos vários *Docker containers* e da máquina *Host* A utilização de *docker compose* permite manipular e gerir os vários componentes de uma aplicação *multi-container*.

  Na implementação final, o *docker compose* foi usado para criar sequencialmente todos os *containers* indicados acima, realizar *health checks* periódicos a estes, gerir os volumes e ligações entre estes, e reconstruir os *containers* caso algum problema eventual ocorra. Para além disso, a utilização do *docker compose* num ambiente de produção permite a partilha de ficheiros de ambiente entre os vários *containers*, garantindo que cada serviço está a operar sobre as mesmas variáveis de ambiente.

  Finalmente, o *docker compose* permite o agrupamento de *logs*, informações e estatísticas sobre os diversos *containers* utilizados, ajudando na deteção e correção de problemas.

- **Máquina de Produção**: Servidor Linux no IT.

  A máquina de produção final é uma Virtual Machine (VM), hospedada dentro dos servidores internos do Instituto de Telecomunicações dentro da Universidade de Aveiro. Esta máquina possui uma quantidade de recursos físicos ligeiramente limitados, sendo que a sua performance tem de ser extraída da forma mais eficiente possível para evitar que a aplicação final tenha problemas a responder aos pedidos ou cause muitas falhas.

  Para além disto, foi necessário criar um conjunto de *scripts* (ou "Cron Jobs") que periodicamente obtêm e processam informação sobre a execução dos serviços e que realizem *backups* do sistema. Estes *backups* são realizados todos os dias, sendo que os últimos 7 dias de *backups* são guardados localmente e os mais antigos apagados. Também são realizados *backups* no inicio de cada mês, que são persistidos de forma a não serem apagados. Devido ao grande período de inatividade do *website*, caso ocorra um problema, o mesmo pode não ser detetado durante várias semanas, levando a que os *backups* relevantes sejam apagados e os atuais sejam inúteis (também contêm o problema). Sendo assim, um *backup* mensal que nunca é apagado resolve os problemas de "corrupção"dos *backups* mais recentes, enquanto estes *backups* diários permitem a diminuição extrema da informação perdida no caso de serem utilizados (menos de 24h de dados perdidos).
- **Email Server**: Sendmail e SMTP da Universidade de Aveiro

  Como mencionado anteriormente, esta máquina possui um serviço sendmail com a capacidade de comunicar com o serviço principal de *emails* da universidade. Este serviço sendmail está hospedado dentro da própria máquina e comunica através de uma porta dedicada para a restante rede interna da Universidade de Aveiro.

CAPÍTULO 4

Implementação

4.1 Aplicação Web

Esta secção apresenta os detalhes de implementação relativos à aplicação desenvolvida. Para maior organização, foi criada uma subsecção para o Frontend e outra para o Backend. Além disso, são apresentados ainda outros detalhes de implementação, como aspetos de autenticação, segurança e medidas para haver salvaguarda dos dados.

4.1.1 Frontend

4.1.1.1 Estrutura da pasta

A estrutura geral do pasta que contém o projeto do Frontend encontra-se dividida em várias pastas. Em primeiro lugar temos 3 pastas presentes: *public* que apresenta o favicon da aplicação, *node_modules* que contém todas as bibliotecas usadas no frontend e *src* que tem todo o código que desenvolvido, que se encontra divido em várias pastas.

Também existem alguns ficheiros com configurações importantes para o Frontend, como o **tailwind.config.js** que apresenta as cores definidas para o tema da aplicação, os ficheiros **package.json**, **package-lock.json** e **pnpm-lock.yaml** que apresentam uma lista com todas as dependências do projeto e garantem a instalação das mesmas de uma forma consistente.

Figura 4.1: Árvore de pastas do Frontend da aplicação

A pasta src/ contém as seguintes pastas e ficheiros:

- **actions**: nesta pasta estão presentes 2 ficheiros: *getActions* e *postActions*, que contêm diversas funções com o propósito de fazer chamadas à API para obter ou enviar dados. Estas funções são depois chamadas noutros ficheiros com os *hooks useQuery*, para realizar pedidos *GET*, e useMutation, para realizar pedidos *POST*, da biblioteca *@tanstack/react-query*.
- **api**: apresenta um único ficheiro, em que é criada uma instância do axios, definindo o Uniform Resource Locator (URL) base da API e também uma opção para permitir o envio de *cookies* automático em cada pedido.
- **assets**: contém todas as imagens usadas na aplicação *web*, como o nosso logótipo, o logo do DETI, entre outros.
- **constants**: apresenta constantes definidas para evitar repetição de código, como por exemplo, o tipo de ficheiros permitidos na submissão de novas propostas.
- **hooks**: constituída por 2 ficheiros, *useAxios*, que apresenta um *hook* para renovar *tokens* de autenticação expirados e atualizar *cookies* e *useMediaQuerie*, que apresenta um *hook* para verificar o tamanho do ecrã do utilizador.
- **stores**: nesta pasta encontram-se diversos ficheiros que servem como *stores* da biblioteca *Zustand*, ou seja, contêm o estado global da aplicação, que pode ser acedido e modificado por qualquer componente da aplicação. Um exemplo de um ficheiro presente é o *userStore*, que contém o estado e algumas informações do utilizador autenticado na aplicação, como o seu email, as suas roles, entre outros.
- **styles**: tem apenas um ficheiro com alguns estilos definidos para a nossa aplicação, como por exemplo, estilos para serem aplicados a uma *scroll bar*.
- **utils**: também contém apenas um ficheiro, que apresenta diversas funções que necessitam de ser usadas em vários componentes, e para evitar a repetição de código, foram

definidas neste ficheiro, como é o caso de uma função que transforma um *timestamp* em milisegundos para uma data com dia, mês, ano e hora.

- **App.jsx**: define a estrutura de navegação da aplicação, com a configuração de várias rotas protegidas por componentes de autenticação, uma rota para páginas não encontradas e uma página de erro.

Também se encontra presente na pasta src/ uma outra sub-pasta, **components/**, que subdivide-se em muitas mais pastas, pois contém todos os componentes da aplicação desenvolvidos:

- **auth**: apresenta vários ficheiros para verificarem os cargos dos *users*, para depois consoante cada uma, terem acesso apenas a páginas da aplicação que lhes são permitidas.
- **cards**: cada ficheiro representa um *card* onde são dispostas informações relativas a uma proposta, para serem apresentados nas diversas páginas existentes, com as informações apropriadas para cada um.
- **forms**: esta pasta apresenta diversos ficheiros, que representam elementos para o formulário de submissão de novas propostas. Podemos encontrar ficheiros com componentes para submissão de dados, como o *InputText* (para o input de texto) e também um ficheiro com um *schema* para ser utilizada na validação do formulário;
- **layout**: contém ficheiros que representam o *layout* da aplicação, ou seja, que são utilizados em maior parte das páginas, como o *Navbar*, que contém a barra superior de navegação da aplicação, ou o *Footer*, que contém o rodapé da aplicação.
- **modals**: constituída por muitos ficheiros que representam modais para serem utilizados também em diversas páginas da aplicação, alguns utilizados para a confirmação de uma ação, outros para indicar o estado de uma situação.
- **pages**: contém ficheiros com o código referente às páginas da aplicação web. Dentro desta pasta, ainda existe outra, com páginas destinadas apenas ao administrador.
- Também existem alguns ficheiros com componentes desenvolvidos que não estão dentro de outras pastas.

4.1.1.2 Protótipo

Em fases iniciais do projeto, foi desenvolvido um protótipo de baixa fidelidade que permitiu discutir os requisitos principais com o professor orientador, e qual seria a melhor maneira de os apresentar nas diversas páginas. Este protótipo viria a ser testado posteriormente, e os seus detalhes encontram-se discutidos na secção 5.1.

4.1.1.3 Compatibilidade com ecrãs mais pequenos

Uma das vantagens de usar tecnologias como o *React*, em conjunto com *TailWind CSS* [14], que é uma *framework* de Cascading Style Sheets (CSS) [15], é que não é necessário desenvolver toda uma nova plataforma para que a mesma funcione corretamente em ecrãs para a qual não foi otimizada (no nosso caso foi desenhada para PC). Assim, basta fazer alguns tipos de adaptações ao UI, para que o mesmo fique adaptável a outro tamanhos.

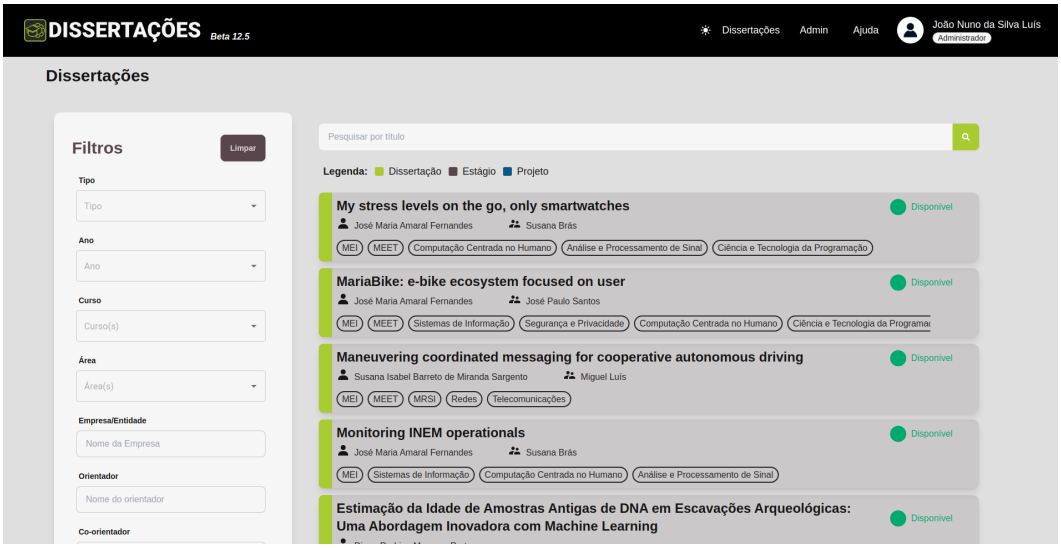


Figura 4.2: Vista da lista de dissertações nova em PC

Na figura seguinte, podemos ver como se apresenta a página principal para PC quando apresenta dissertações adicionadas:

A plataforma antiga não continha qualquer suporte para telemóveis, já que na altura em que foi desenvolvida *smartphones* nem sequer eram comuns, pelo que a página se comportava como podemos ver na figura 2.1. Podemos ver também como tratámos desta página para o caso de ser um telemóvel na figura 4.4.

Figura 4.3: Vista da lista de dissertações antiga num *smartphone*

Figura 4.4: Vista da lista de dissertações nova num *smartphone*

Depois de discussões entre o grupo e com o professor orientador, chegou-se à conclusão que quase todos os utilizadores do sistema no telemóvel seriam estudantes. Por esta razão e com o intuito de poupar algum tempo, para focar noutras funcionalidades, apenas as páginas que podem ser acedidas pelos estudantes estão otimizadas para ecrãs mais pequenos.

4.1.1.4 Adicionar dissertações

Para ter uma lista de dissertações, é necessário que estas sejam submetidas por docentes. Assim, foi desenvolvida uma página onde os docentes podem submeter as suas propostas de dissertação, como pode ser visto na figura abaixo:

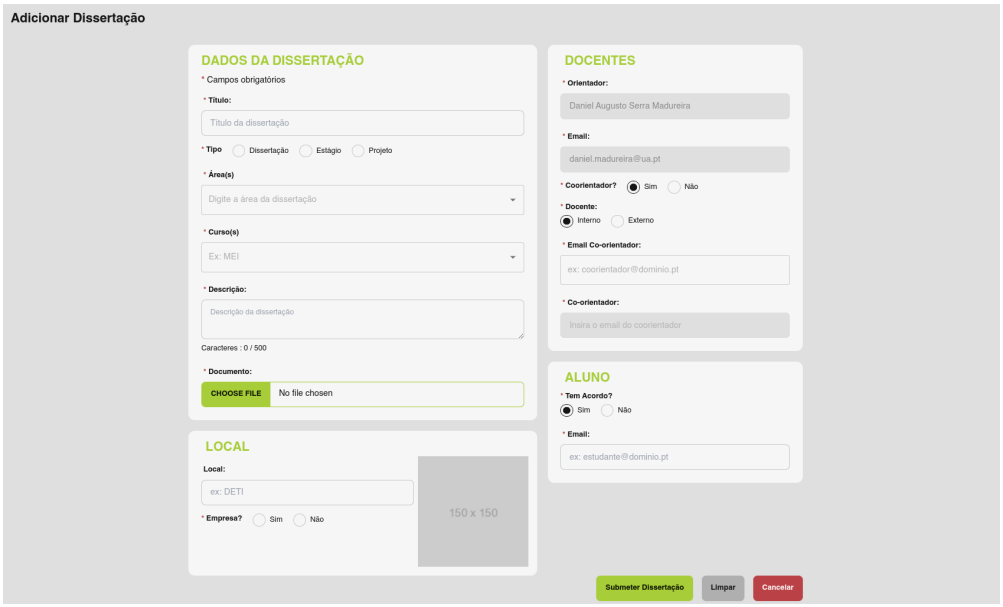


Figura 4.5: Vista do formulário para submeter dissertações

Esta página é composta por quatro grandes campos claramente divididos na UI. Assim, torna-se mais fácil para o docente perceber o que é necessário para submeter uma proposta de dissertação/estágio. Para submeter a proposta pelo formulário, é necessário preencher todos os campos obrigatórios, marcados com asterisco (*). De forma a que as dissertações submetidas não tragam problemas para a plataforma, foram definidas algumas regras para os campos, como por exemplo, o título da dissertação não pode ter mais de 130 carateres, o resumo não pode ter mais de 500 caracteres, entre outras.

Estas regras estão definidas num *schema* que é utilizado para validar o formulário antes de ser submetido. Usando a biblioteca *Yup* [16], é possível definir um *schema* com todas as regras necessárias para cada campo, e depois utilizar este *schema* para validar o formulário, neste caso usando o *YupResolver* para o formulário em *React Hook Form* [17]. Assim, através do *YupResolver*, é possível validar o formulário antes de ser submetido, e, caso existam erros, o utilizador é informado dos mesmos.

A lógica por detrás do *schema* compreende condições impostas pelos requisitos, como por exemplo, a distinção entre co-orientadores internos e externos, que dado a sua natureza, dispõem de campos diferentes. Assim, o *schema* é capaz de validar a presença de campos obrigatórios consoante a escolha do utilizador.

É de notar que no formulário são enviados ficheiros, como o documento da dissertação e o memorando, que são guardados no sistema de ficheiros do servidor. Para garantir que os ficheiros são enviados corretamente, foram tidos em conta verificações que validam os ficheiros antes de serem enviados, e que também verifica se os ficheiros são enviados corretamente para o servidor.

Para além de ficheiros são também enviadas imagens, mais especificamente imagens das empresas/locais da dissertação. Assim, foi desenvolvido um componente de galeria para as mesmas mesmas,onde os utilizadores podem escolher entre as imagens presentes na plataforma

ou dar *upload* de uma nova imagem. Este componente é composto por um carrossel onde as imagens são apresentadas, e um botão para fazer *upload* de uma nova imagem. As imagens são transformadas em base64 antes de serem enviadas para o servidor, para que possam ser guardadas no sistema de ficheiros.

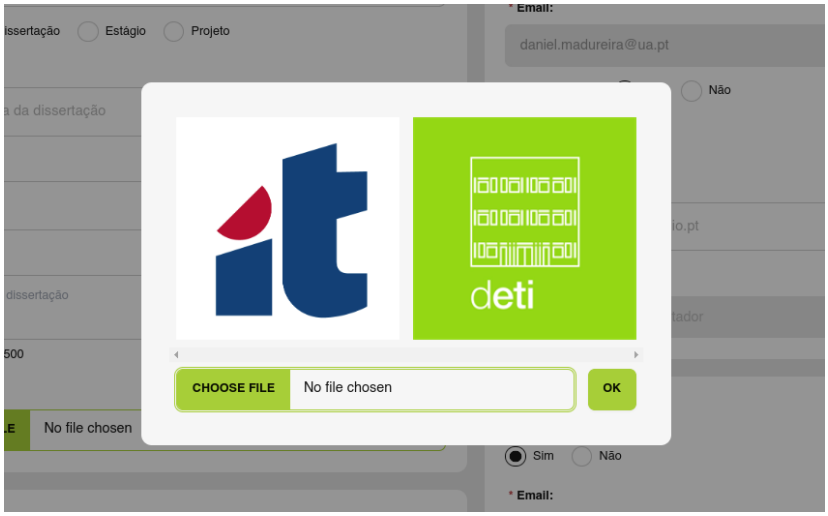


Figura 4.6: Galeria de imagens para escolher ou fazer *upload* de uma nova imagem

4.1.1.5 Editar dissertações

No caso de ser necessário editar uma proposta de dissertação, foi desenvolvida uma página onde os docentes podem editar as suas propostas submetidas. Esta página é semelhante à página de submissão de propostas, uma vez que está feita sobre o mesmo componente, mas com os campos preenchidos com os dados da proposta a ser editada.

Nesta página é necessário pedir à API os dados da proposta a ser editada, e mais importante ainda, os ficheiros como o documento da dissertação e o memorando, que são guardados no sistema de ficheiros do servidor. Assim, ao carregar a página, são imediatamente pedidos os ficheiros de modo a ser possível ver quais ficheiros existem e quais são os seus nomes.

Uma vez que não é possível editar ficheiros diretamente, é necessário encontrar uma solução alternativa para este problema. Assim, optamos por mostrar os ficheiros existentes e permitir ao utilizador fazer *upload* de novos ficheiros, que irão substituir os ficheiros existentes. Desta forma é possível verificar os ficheiros para a proposta a ser editada, e fazer, caso seja necessário, a substituição dos mesmos.

Ao submeter a proposta editada, para além dos campos da proposta validados pelo *schema*, são enviados mais três campos, que indicam se o documento da dissertação, o memorando e a imagem foram alterados. Estes campos são necessários para que a API saiba se é necessário substituir os ficheiros existentes pelos novos ficheiros enviados.

4.1.1.6 Lista de Propostas

A consulta de todas as propostas que foram submetidas por docentes e aceites por um administrador e diretores de curso é uma ação que todos os atores do nosso sistema podem realizar, pelo que foi necessário desenvolver uma página principal agradável, intuitiva e que conseguisse mostrar ao utilizador uma quantidade de informação nem exagerada nem escassa. Na figura 4.2 podemos observar que as dissertações estão dispostas numa lista, com um *card* para cada uma, que contém o título, o nome do orientador e co-orientador, os cursos e áreas a que esta se destina, o status dela e qual o seu tipo, se é uma dissertação, estágio ou projeto. Clicando num dos nomes do orientador/co-orientador, é aberto uma aplicação para se enviar um email a uma destas entidades. O tipo de cada proposta está identificado com uma cor diferente, sendo fornecido também uma legenda para ajudar o utilizador a perceber o que cada cor significa. Caso o utilizador clique num *card* de uma proposta, é redirecionado para uma página contendo todos os detalhes da mesma.

Também se encontra disponível um componente com filtros, para permitir ao utilizador poder fazer uma filtragem rápida por propostas com base nos critérios, status, tipo, ano, curso, área, empresa, nome do orientador e nome do co-orientador, sendo possível a filtragem pelo preenchimento de vários campos. Além disto também é possível clicar nas áreas ou cursos das propostas para as filtrar por esses critérios. Para utilizadores mais experientes na utilização de sistemas semelhantes ao nosso, desenvolvemos uma barra de pesquisa baseada em *tags*, ou seja, o utilizador insere uma palavra-chave começada com o carácter "@", como por exemplo @ano, @status e depois com um espaço, insere o valor pelo qual quer pesquisar entre aspas, ou seja, ficaria algo como @status "Disponível". Caso o utilizador não introduza uma palavra chave válida, a pesquisa é feita com base no título de uma proposta. Na implementação dos filtros, criámos uma *store* usando o *Zustand* [18], para se poder guardar o estado dos filtros existentes de uma forma eficiente e para caso seja necessário aceder à *store* e adicionar filtros noutros componentes, em que é possível aceder ao estado global dos filtros e alterá-lo.

Adicionámos o número total de propostas que se encontram na lista e as que estão a ser visualizadas, isto porque à medida que o utilizador vai dando *scroll* para baixo na página, vão sendo carregadas mais propostas. Para conseguirmos atingir esta funcionalidade usámos o *hook* useInfiniteQuery da biblioteca *@tanstack/react* que permite fazer pedidos à API de forma paginada, ou seja, a cada pedido feito à API, são carregadas mais propostas, e assim sucessivamente até que todas as propostas sejam carregadas. De forma a chegar ao *infinite scroll*, foi necessário manter um registo da referencia do fim da lista de propostas, para que quando o utilizador chegasse a essa referência, fosse feito um novo pedido de uma nova página de propostas à API.

4.1.1.7 Detalhes de uma Proposta

Os detalhes de uma proposta são apresentados numa página à parte acessível através dos *cards* das propostas, onde se encontram todas as informações relativas a uma proposta, que incluem:

- título;
- tipo;
- status;
- áreas e cursos;
- imagem associada;
- orientador e co-orientador;
- ano e data de submissão;
- local;
- descrição;
- documento PDF;

Esta página apresenta vários estados, em que, dependendo do papel do utilizador autenticado, e também do *status* da proposta, são apresentados diferentes botões, como por exemplo, quando uma dissertação ainda não foi aceite ou recusada por um administrador, aparece a opção para o puder realizar como se pode visualizar na página 4.7

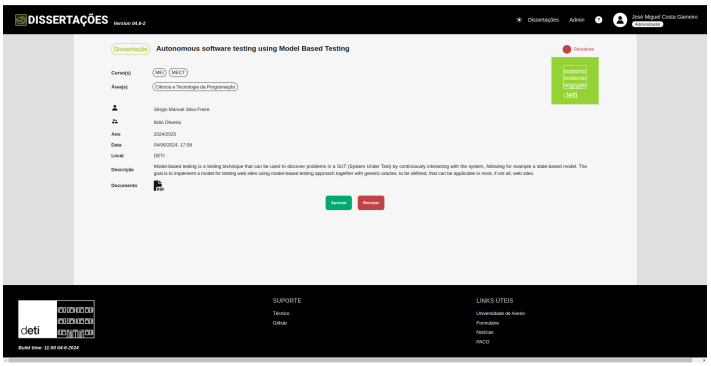


Figura 4.7: Detalhes de uma dissertação por aprovar para um administrador

Ou como o caso de quando uma proposta é atribuída a um estudante, ou seja, o "casamento"é concluído entre um estudante e um orientador, então são disponibilizadas informações sobre o estudante e data de atribuição, como podemos ver na figura 4.8. Também podemos observar que uma opção para revogar uma proposta surge, no entanto, esta opção apenas se encontra disponível para administradores.

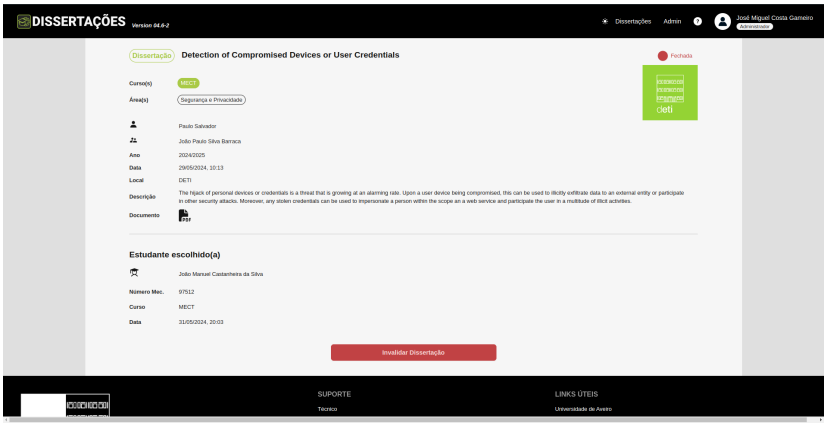


Figura 4.8: Detalhes de uma dissertação atribuída a um estudante

4.1.1.8 Perfil e Perfil Público

Para cada utilizador autenticado, são disponibilizadas no seu perfil informações sobre o mesmo, como o seu nome, número mecanográfico, email, curso e cargos. Existe também uma designação, em que o seu valor inicial é o nome do utilizador normal, no entanto, é dada a possibilidade de o alterar para que este seja identificado na aplicação por um nome que seja mais comum para todas as pessoas que utilizam a aplicação. No caso do cargo de estudante existe ainda um outro campo, que é a média, que é algo que o estudante tem de preencher ou no perfil, ou na primeira vez que efetua o *login*. Este dado é fornecido aos professores, no processo de escolha de estudantes para uma proposta. Para além disto existe uma outra secção no perfil que mostra, no caso de um estudante, as propostas em que mostrou interesse, onde é disponibilizado o estado em que esta se encontra, ou seja, se o orientador mostrou também interesse no mesmo ou se já está atribuída. No caso de um docente, mostra os acordos, ou seja, as propostas que já foram atribuídas a estudantes.

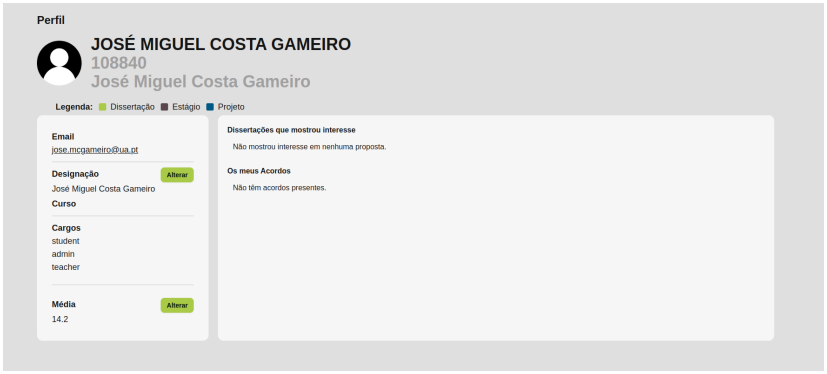


Figura 4.9: Perfil de um utilizador com todos os cargos

O perfil público é algo semelhante ao perfil normal mas com algumas diferenças, pois, este é algo virado para os docentes, em que estão presentes todas as suas propostas que foram submetidas e devidamente aceites por um administrador e pelo menos um diretor de curso. Este perfil é acessível por qualquer pessoa através do URL */professor/[email]* em que o email é o do docente que se pretende pesquisar. Esta ferramenta é útil, para de forma rápida, oferece aos estudantes a capacidade de puderem analisar as propostas de um determinado docente.

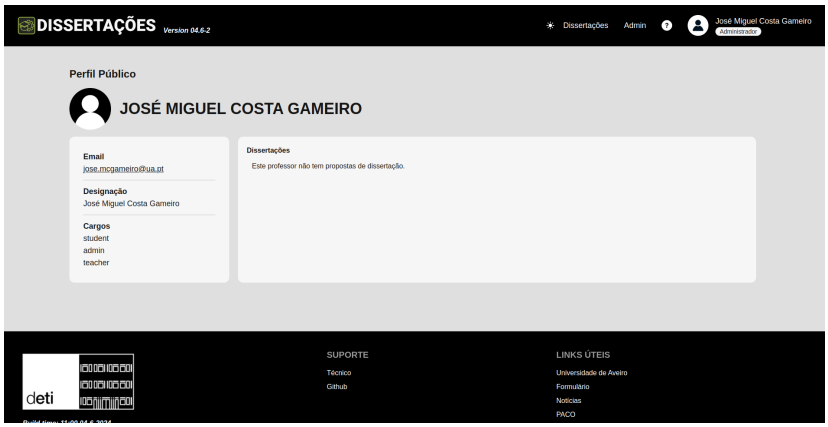


Figura 4.10: Perfil Público de um docente

4.1.1.9 Propostas de um docente

Para cada docente, é disponibilizada uma página onde este pode ver todas as propostas que submeteu, e que foram aceites pelo administrador e pelo menos um diretor de curso. Esta página é semelhante à página de lista de propostas, no entanto, apenas são apresentadas as propostas submetidas pelo docente autenticado. Em cada proposta, é possível verificar os estudantes que mostraram interesse na mesma e assinaram o acordo final, caso existam, e realizar também as ações de escolher um estudante e assinar o acordo final.

À medida que vão existindo alterações no estado da proposta, quer da parte do estudante, quer da parte do docente, são enviados *emails* a informar das alterações do estado da dissertação, nos detalhes de uma proposta e no perfil do docente/estudante o estado também é alterado para informar os utilizadores das alterações que ocorreram.

4.1.1.10 Propostas para um diretor de curso

Um diretor de curso tem uma página para poder validar/revogar um curso numa proposta submetida por um docente. Esta ação é possível de se fazer quer nesta página, quer nos detalhes da mesma, desde que o utilizador em questão tenha o cargo de diretor de um curso associado à proposta.

Para além disto, é possível também ver as suas últimas ações, ou seja, as últimas propostas que foram aprovadas ou rejeitadas por si.

4.1.1.11 Funcionalidades de um administrador

Uma das funcionalidades mais importantes para o administrador da plataforma é a capacidade de aprovar ou rejeitar propostas de dissertações submetidas por professores. Para tal, o administrador tem acesso a uma página onde pode ver todas as propostas submetidas que ainda não foram aceites, que apresenta uma estrutura semelhante à da figura 4.2. Nesta página, o administrador pode ver a informação de cada proposta e proceder à sua aprovação ou decisão. No caso da proposta ser rejeitada também é necessário indicar o motivo da rejeição, que é enviado no email ao professor.

Devido ser expectável uma grande quantidade de propostas submetidas, foi desenvolvida a funcionalidade de aceitar várias dissertações de uma só vez, para facilitar o processo de aprovação por parte do administrador.

Outra funcionalidade desenvolvida consiste na possibilidade de o administrador poder ver todas as dissertações que foram aprovadas pelo mesmo, mas que ainda não foram aprovadas por um diretor.

Ver Quotas atuais dos Docentes

O administrador tem também a possibilidade de ver as quotas atuais de cada docente, ou seja, o número de propostas de dissertações que cada docente pode submeter num determinado ano letivo. Esta funcionalidade é importante para garantir que as propostas são distribuídas de forma equitativa pelos docentes, tendo cada docente uma quota máxima de 4. Com cada acordo feito, a quota do docente do ano respetivo é atualizada, aumentando em 1 caso seja de orientação única ou em 0.5 caso haja a presença de um coorientador.

Monitorização

Nesta página, o administrador pode verificar o estado atual da aplicação, tendo acesso ao estado da API, incluindo o seu tempo de resposta. Também pode verificar os últimos logins realizados na aplicação, podendo filtrar entre os 10 e os 100 últimos logins, para garantir que não existem acessos indevidos ou para dar *debug* a problemas de autenticação.

Gerir Cursos e Áreas

Nesta página, o administrador pode adicionar e editar cursos, áreas de especialização e diretores de curso. À data presente, os cursos presentes são os registados na tabela 3.1. As funcionalidades de editar e remover áreas não foram implementadas, visto que a alteração de uma área teria de implicar a alteração de todas as propostas que a referenciam, o que poderia levar a problemas de integridade alização.

Para cada curso, o administrador pode ainda gerir o seu diretor de curso, adicionando um novo diretor ou removendo-o.

Exportar Dados para Ficheiro Excel

O administrador tem ainda a possibilidade de exportar todos os dados da aplicação para um ficheiro Excel, para que possam ser guardados e consultados posteriormente.

Atualizar Ano Académico

O administrador tem a possibilidade de atualizar o ano académico de forma a simular as mudanças de ano letivo. Após atualizar o ano letivo, todas as propostas submetidas passam a ser referentes ao novo ano letivo, e as quotas dos docentes são também atualizadas para o novo ano.

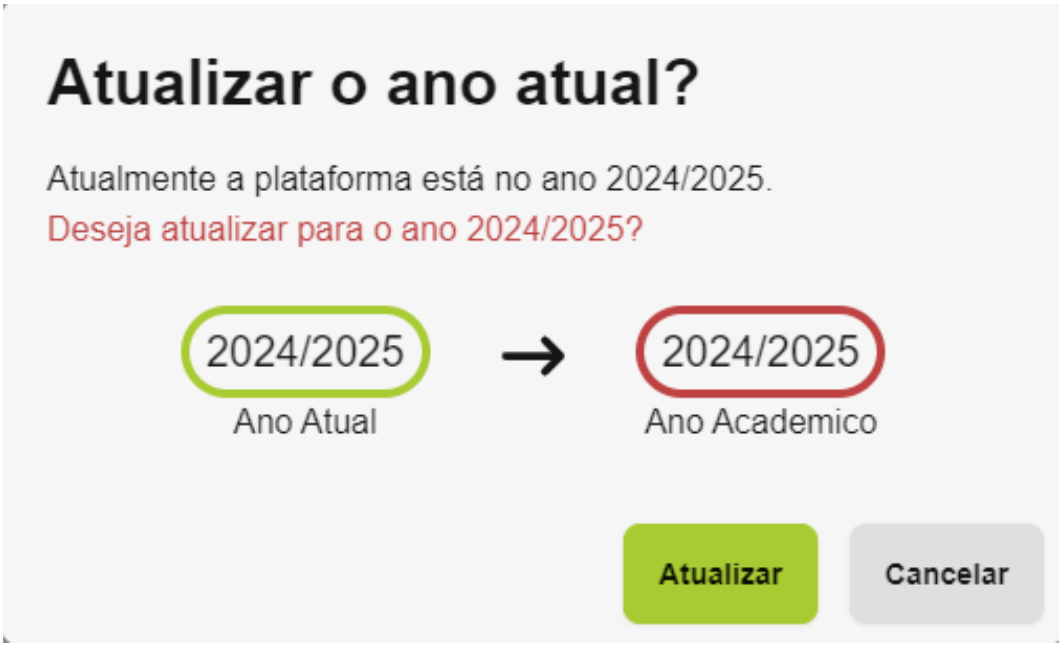


Figura 4.11: Menu de Atualização do ano letivo

Como se pode ver na figura acima, dado que a atualização do ano letivo só pode ser feita no inicio de um novo ano, não é possível ainda a atualização do mesmo.

4.1.2 Backend

4.1.2.1 Estrutura da pasta

A estrutura geral da implementação do Backend da aplicação pode ser separada em quatro pastas: a API em si, as duas bases de dados e ainda os ficheiros submetidos pelos docentes.

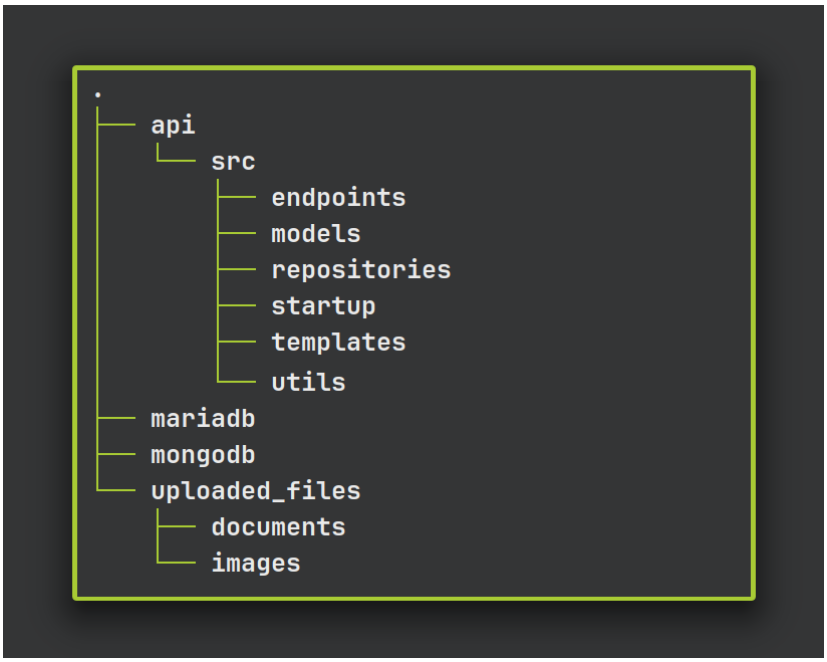


Figura 4.12: Árvore de pastas do Backend da aplicação

Começando pela API, esta pasta contém todos os ficheiros de código e de configurações necessários para a implementação do serviço final. Esta pasta está subdividida de forma a suportar uma divisão mais clara das funcionalidades de cada ficheiro:

- *Endpoints*: Funções que processam os dados recebidos pelo serviço e que utilizam as restantes funções desenvolvidas noutras pastas para manipular dados persistidos, calcular as respostas aos pedidos e ainda garantir a segurança e autenticação de cada pedido;
- *Models*: Utilizando a biblioteca *Pydantic*, os modelos representam os Objetos mais relevantes do sistema pelos seus atributos, como por exemplo, os vários campos de dados que constituem uma proposta de dissertação ou a informação de um utilizador;
- *Repositories*: Estas funções permitem a integração com a base de dados principal (MongoDB), interagindo com a mesma. Esta separação permite acessos controlados à base de dados e promove a reutilização de código para a comunicação com a mesma;
- *Startup*: Os ficheiros de *startup* são sempre executados ao iniciar o serviço, garantindo um estado inicial do mesmo consistente e esperado. Alguns exemplos das funcionalidades deste ficheiros são o *setup* das ligações às bases de dados e a geração de dados estáticos nas mesmas, como por exemplo, a listagem de mestrados;
- Templates: Estas ficheiros são os *templates* HTML consumidos pela implementação de Jinja2 [19], o que permite gerar páginas estáticas em HTML para, por exemplo,

criar *emails* personalizados e listas de propostas para a secretaria da Universidade, sem precisar de duplicar ou exportar as mesmas manualmente;

- Utils: Por fim, esta pasta contém diversas funções úteis para o resto da aplicação, como por exemplo, verificações de consistência de dados (*parsers*), serviços de autenticação, *loggers*, verificações de ficheiros e ainda funções de integração com o serviço de *emails* da Universidade.

As pastas "MariaDB"e "MongoDB"são utilizadas para persistir as configurações da base de dados em questão, permitindo uma edição fácil das mesmas sem necessitar de variáveis de ambiente complexas ou do uso de serviços externos.

Finalmente, a pasta de ficheiros submetidos permite guardar preservar os seguintes ficheiros:

- Documentos: Qualquer documento de uma dissertação, seja o documento base da mesma ou o memorando, é preservado nesta pasta para permitir o acesso fácil através do código ou através do sistema de ficheiros da própria máquina;
- Imagens: Similarmente, todas as imagens submetidas pela plataforma são preservadas nesta pasta. Estas imagens são, geralmente, os logótipos das diversas empresas incluídas para as dissertações inseridas;
- Logs: Como mencionado anteriormente, foram utilizados *loggers* através da aplicação. De forma a que as informações de problemas e mudanças na plataforma possam ser analisadas posteriormente, qualquer *log* gerado pela aplicação é persistido por ficheiros localizados nesta pasta;

4.1.2.2 API

Para o processamento de dados e integração entre os diversos componentes do sistema desenvolvido, foi necessário criar um serviço primário que consiga obter pedidos de clientes, calcular uma resposta apropriada, responder aos pedidos, manipular as diversas bases de dados, integrar os serviços externos e acima de tudo manter uma alta qualidade de serviço e disponibilidade.

Existem várias *frameworks* que se conseguem encaixar nestes critérios e realizar todos os requisitos necessários por isso foi necessário criar um conjunto de regras para realizar uma escolha sobre qual *framework* seria utilizada neste projeto.

Após várias reuniões com o nosso orientador, foi alcançado um consenso nestas regras. A API é, acima de tudo, altamente manutenível pelos futuros administradores da plataforma, e para isso usaram-se apenas componentes com uma alta garantia de "sobrevivência"durante todo o ciclo de vida e uso da aplicação, e ainda fazer uma gestão eficiente nos recursos da máquina de produção, visto que as características da mesma não eram conhecidas durante a parte inicial do desenvolvimento. Com estes requisitos em mente, algumas opções mais comuns foram rapidamente eliminadas até se chegar à solução final.

Uma caraterística fundamental a escolher era a API ser escrita na linguagem de programação Python, visto que esta é altamente popular e o número de bibliotecas suportadas por esta é imensa, o que permite um acrescento rápido de funcionalidades no serviço, caso sejam necessárias no futuro.

Assim, a opção tomada foi usar a *framework* FastAPI [19] já que seguia todos os critérios necessários. O uso da FastAPI permite o rápido desenvolvimento de uma API RESTful [20] sem grande *overhead*, como é comum num sistema baseado em, por exemplo, Spring [21] ou NodeJS [22].

Esta *framework*, como o nome indica, foi também desenvolvida para ser o mais eficiente possível, disponibilizando diversas integrações com, por exemplo, a biblioteca Uvicorn, que permite que a API seja executada por diversas *threads* individuais de forma assíncrona e com configuração manual mínima, algo que normalmente requer uma modificação extensa da linguagem de Python, visto que o conceito principal da mesma normalmente não permite este nível de computação assíncrona.

4.1.2.3 Base de Dados

Para a separação de dados e o proveito das vantagens de cada serviço, foram implementadas múltiplas formas de persistência de informação.

A primeira destas é o serviço de base de dados SQL MariaDB. Esta base de dados foi escolhida prioritariamente pelo seu alto nível de segurança e integridade, que são requisitos necessários visto que este serviço irá persistir as informações administrativas da plataforma e os chamados "números mágicos"da mesma, como por exemplo, a listagem dos administradores e as suas permissões, o número máximo de quota de um docente para o dado ano académico, a listagem dos Mestrados suportados pela plataforma e a localização das restantes bases de dados. O uso de SQL garante ainda a formatação e a compatibilidade de informação ao longo do tempo de vida da plataforma, para que não seja preciso modificar quaisquer serviços adjacentes. Este base de dados é ainda altamente compacta e a sua compatibilidade com um serviço MySQL permite a que, no futuro, o serviço seja migrado para uma implementação da universidade mais segura ou até na *cloud*, sem quaisquer alterações ao modelo de dados utilizado.

Para a base de dados geral do sistema, foi escolhido o serviço NoSQL proveniente do MongoDB. O uso deste serviço trás diversas vantagens para o desenvolvimento e manutenção da plataforma. Ao contrário de serviços SQL (que usam formatação de dados por tabelas), o MongoDB não tem restrições sobre o formato de cada pedaço de dados, permitindo que os modelos dos mesmos evoluam sem perder a compatibilidade com versões anteriores dos dados. Para além disto, esta base de dados opera sobre dados em formato JSON, o que remove a necessidade de subdividir modelos por tabelas separadas ou de realizar processamentos extensos cada vez que é necessário operar sobre um modelo guardado em tabela. Finalmente, o MongoDB contêm ferramentas de organização interna que permitem otimizar cada coleção de modelos dependendo da forma de como são acedidos, e ainda realizar complexas operações de pesquisa por filtros sem a necessidade de delegar esse processo ao serviço da API.

Por fim, o sistema de ficheiros do UNIX [23] foi utilizado para complementar a persistência de ficheiros, sem a necessidade de processar os mesmos cada vez que estes são recebidos ou enviados. Visto que os ficheiros são primeiro analisados por integridade dos seus conteúdos em múltiplos outros pontos do sistema (na API, no *reverse proxy* e no frontend), não faz sentido

persistir os mesmos dentro de uma das bases de dados mencionadas anteriormente, visto que a transformação dos mesmos em informação processável pelas mesmas e vice-versa iria introduzir processos desnecessários. Outra vantagem da utilização do sistema de ficheiros vem do ponto de vista da monitorização, uma vez que um administrador com acesso à máquina pode facilmente gerir o tamanho dos ficheiros, alterar os mesmos em caso de corrupção ou qualquer problema no processamento inicial ou até remover deste sistema os ficheiros de dissertações mais antigos, algo que é realizado na plataforma anterior cerca de uma vez a cada dois anos para libertar espaço dentro da máquina e arquivar os ficheiros das dissertações mais antigas na secretaria da Universidade. Por fim, o uso do sistema de ficheiros permite diminuir extremamente o tamanho das restantes bases de dados, visto que estes ocupam a maior parte do tamanho (em disco) do modelo de uma dissertação, o que revela ainda um grande aumento na performance de pesquisas de propostas, visto que ao contrário de outros campos do modelo, não é necessário obter o ficheiro em sí cada vez que se precisa de aceder a uma proposta, mas apenas quando o ficheiro da mesma é pedido pelos restantes serviços da plataforma. Isto reduz o tamanho da informação de resposta da ordem das dezenas de *megabytes* para apenas poucos *kilobytes*.

4.2.1 Autenticação

Constituía um requisito desde o início do projeto que a autenticação de utilizadores na aplicação fosse feita através do IdP da Universidade de Aveiro. O IdP da UA permite um serviço de *Single Sign-On (SSO)* que dá permissão a que um utilizador se autentique uma única vez e tenha acesso a vários serviços sem ter de se autenticar novamente. Este serviço é baseado no protocolo Security Assertion Markup Language (SAML). Numa primeira abordagem foi-nos pedido que fizéssemos a autenticação através do protocolo SAML. Porém esta abordagem requereria uma maior complexidade de implementação, pois este protocolo caiu em desuso e perdeu suporte por parte da comunidade de desenvolvimento. A autenticação através do protocolo SAML também iria requerer que os STIC fizessem a configuração do SAML para a nossa aplicação, o que poderia ser um processo moroso e complexo.

Sendo confrontados com vários problemas em relação à implementação e uso deste protocolo, decidimos mudar a nossa abordagem e optar por uma mais moderna e mais simples, que é a autenticação através do protocolo OpenID Connect (OIDC). A Universidade de Aveiro disponibiliza um serviço de autenticação através do protocolo OIDC, que é mais simples de implementar e que é mais seguro que o protocolo SAML, para além de ser mais moderno e ter mais suporte por parte da comunidade de desenvolvimento. Este serviço consiste numa autenticação federada na Universidade de Aveiro através do uso do WSO2 *Identity Server*, que é um serviço de autenticação e autorização que implementa o protocolo OIDC, que comunica diretamente com o IdP da Universidade.

A autenticação na aplicação apresenta os seguintes passos:

- Redirecionamento para o IdP da UA;
- Autenticação do utilizador;
- Redirecionamento para a aplicação com o *token* de autenticação;
- Validação do *token* de autenticação;
- Obtenção de um *token* de acesso;
- Obtenção dos dados do utilizador através do *token* de acesso;
- Armazenamento dos dados do utilizador na base de dados;
- Envio do *token* de acesso para o utilizador;

Quando carrega no botão de "Entrar"o utilizador é redirecionado para o IdP da Universidade de Aveiro, onde é autenticado. Após a autenticação, o utilizador é redirecionado para a aplicação com um *token* de autenticação. Internamente, o IdP valida os dados fornecidos pelo utilizador (email e password), e devolve um *token* de autenticação que é enviado para a aplicação. A aplicação envia então o *token* para o Backend através de um pedido HTTP que é validado pelo mesmo. Após a validação do *token* de autenticação, é então requisitado um *token* de acesso para o utilizador. Com o *token* de acesso é possível obter os dados do utilizador, como o nome, email, número de aluno, etc. Estas informações são então armazenadas na base de dados da aplicação, para posterior uso. Finalmente, o *token* de acesso é enviado para

o utilizador, que é armazenado no seu *browser* em *Cookies* e é utilizado para autenticar o utilizador nas próximas vezes que este aceder à aplicação.

O protocolo OIDC também introduz a noção de um *refresh token*, que é um *token* com uma maior duração que é utilizado para obter um novo *access token* quando este expira. Este *refresh token* é utilizado na aplicação para garantir que um utilizador não tem de se autenticar novamente após o *access token* expirar.

Para facilitar a autenticação e torná-la também mais segura, foi desenvolvida uma função que verifica se um utilizador está autenticado. Esta função recebe o *access token* do utilizador e tenta obter as informações do utilizador perante o IdP da Universidade, considerando-o como autenticado caso este pedido tenha sucesso. Esta função é utilizada em todas as rotas da aplicação que requerem autenticação, garantindo que apenas utilizadores autenticados têm acesso a estas rotas.

Com recurso à biblioteca *Axios* de React, foi implementado um *interceptor* que captura todas as respostas de pedidos HTTP feitos pela aplicação ao Backend antes que estas possam ser processadas pela aplicação. No caso de um pedido ao Backend ser rejeitado por falta de autenticação, o *interceptor* faz um novo pedido, desta vez requisitando um novo *access token* com o *refresh token* armazenado no *cookie* do utilizador. Este novo *access token* é então armazenado no *cookie* do utilizador e o pedido original é reenviado com o novo *access token*. Caso o *interceptor* não detete um *refresh token* no *cookie* do utilizador, ou o pedido por um novo *access token*, este redireciona o utilizador para a página de autenticação.

Desta forma, o utilizador não tem de se autenticar novamente após o *access token* expirar, garantindo uma melhor experiência de utilização. Também é garantido que o utilizador se mantém autenticado durante a sua sessão na aplicação, mesmo que esta seja prolongada.

4.2.2 Autorização

A autorização de utilizadores na aplicação é feita através de um sistema de permissões baseado em *roles*. Cada utilizador poderá ter uma ou várias *roles* associadas, sendo estas extraídas a partir do seu vínculo com a universidade (aluno, docente, etc), o qual é obtido através do IdP. Estas roles são armazenadas na base de dados da aplicação e são utilizadas para determinar que ações é um utilizador pode realizar na aplicação.

Atualmente, a aplicação suporta as seguintes funções:

- *Student*: Utilizador que está inscrito num curso da Universidade de Aveiro; Tipo de vinculo "Aluno";
- *Teacher*: Utilizador que é docente na Universidade de Aveiro; Tipo de vinculo "Docente";
- *Diretor de Curso*: Utilizador que é diretor de um curso na Universidade de Aveiro; Atribuídos pelo administrador
- *Admin*: Utilizador que é administrador da aplicação; Atribuídos através da base de dados de configurações;
- *Staff*: Utilizador que é da secretaria da Universidade de Aveiro; Tipo de Vinculo "Não Docente";

Tanto no Frontend como no Backend, as funcionalidades são limitadas consoante o papel do utilizador, que é verificada antes de realizar qualquer ação. Por exemplo, exportar o estado das dissertações atuais para um ficheiro Excel é uma funcionalidade apenas disponível para o administrador da plataforma e para membros da secretaria (roles de *Administrador* e *Staff* respetivamente).

A colaboração entre orientadores e co-orientadores em propostas de dissertação assume que o co-orientador poderá ser uma pessoa externa à Universidade de Aveiro, como um docente de outra instituição de ensino superior ou um profissional de uma empresa. Para garantir suporte para estes co-orientadores externos, os mesmos são criados quando é adicionada uma proposta que os referencia, assumindo também a role de *Teacher*, porém os mesmos nunca poderão interagir com o website pois não se conseguem autenticar perante o IdP da Universidade de Aveiro.

4.3 Segurança

Uma das vantagens de hospedar o sistema dentro da rede interna da UA é a disponibilidade de mecanismos de segurança de *networking*. Apesar de ser preciso superar algumas restrições impostas sobre o uso da rede interna da Universidade, esta implementação permite utilizar grande parte das defesas a ataques disponíveis como, por exemplo, ataques de Denial of Service (DoS). Estas defesas também incluem várias *firewalls* e restrições nas comunicações, o que nos permite utilizar outros serviços na máquina de produção sem a preocupação de um atacante aceder à mesma por meios não previstos. Para além disto, a monitorização automática de pedidos suspeitos por parte da rede em si permite a nossa plataforma estar protegida de ataques mais rudimentares.

No entanto, é necessário desenvolver mais camadas de proteção no nosso sistema para maior segurança, diferenciando os pedidos legítimos dos pedidos maliciosos.

Assim, a primeira barreira implementada é o *reverse proxy*. Este serviço analisa os cabeçalhos, as origens e os meta-dados de cada pedido, filtrando qualquer comunicação que não esteja dentro dos parâmetros definidos. Por exemplo, qualquer pedido não proveniente do frontend da plataforma, com cabeçalhos em falta, ou com mais de 50 *megabytes* de tamanho total é automaticamente rejeitado. Adicionalmente, estes filtros ajudam a remover o processamento destes pedidos pela API, diminuindo os danos causados por estes. Após completar estas verificações, a mensagem é enviada para a API. Esta encarrega-se de verificar a coerência dos dados submetidos no corpo e nos parâmetros da mensagem, tal como o *token* de autenticação da mesma. Qualquer valor submetido tem de primeiro ser verificado para garantir que faz sentido no fluxo pretendido da aplicação, por exemplo, um aluno não pode colocar interesse numa proposta de uma dissertação caso esta ainda não tenha sido aprovada por um administrador, isto é, não aparece na listagem pública. Em pedidos que submetem dados para serem persistidos, os valores inseridos também têm de ser verificados, como por exemplo, o tamanho máximo do título de uma dissertação (que é de 130 carateres). Adicionalmente, visto que apenas são aceites ficheiros do tipo PDF (ficheiros das propostas

ou memorandos) ou PNG (logótipos de empresas), ambos têm de ser verificados pela sua consistência (se o PDF é válido) e características (resolução máxima da imagem).

Em segundo lugar, a performance da aplicação é monitorizada a cada hora, o que ajuda a detetar em tempo útil casos em que seja introduzido *malware*, por exemplo.

Para garantir que todas estas etapas realmente conseguem realmente impedir atacantes foi realizando um teste de resistência à API pelo Professor João Barraca, em que foram simulados pedidos formados corretamente e autenticados, mas com valores fictícios e ficheiros inconsistentes. Estes testes não revelaram fraquezas com a implementação do serviço, sendo que nenhum afetou o sistema em si, tendo sido a maior parte dos pedidos recusados pela API, e os restantes não criaram quaisquer problemas visíveis publicamente, pois apesar de terem sido introduzidas propostas com valores estranhos, estas teriam de ser aprovadas pelo administrador, validando assim também o fluxo definido para a plataforma. É de notar que deste teste foram descobertos alguns problemas com a verificação do nome do co-Orientador pois estava apenas a ser feita no frontend, pelo que após este teste, implementámos essa correção verificando essa questão também na API. Os ficheiros submetidos nos testes não resultaram em qualquer ameaça para a API, visto que são extensamente analisados pela mesma em memória e apenas são persistidos no sistema após completar estas verificações. Neste caso houve rejeição dos pedidos realizados.

O uso da framework *React* ajuda também a proteger a aplicação de ataques de Cross-site Scripting (XSS), visto que a mesma previne a injeção de código malicioso no HTML da aplicação.

O uso de uma base de dados NoSQL (*MongoDB*) protege a aplicação de ataques de injeção de código SQL, visto que este tipo de base de dados não utiliza SQL para realizar operações. No entanto, ainda pode ser vulnerável a ataques de NoSQL *injection*, pelo que é necessário garantir que os dados persistidos e acedidos são validos. Desta forma, a API realiza verificações dos dados submetidos pelos utilizadores antes de os persistir na base de dados, garantindo que estes são válidos e consistentes.

Foi tomada a decisão de guardar o token de acesso no *cookie* do utilizador, visto que este é mais seguro que guardá-lo no *local storage* ou no *session storage*, fornecendo alguns mecanismos de segurança adicionais, através do uso de *flags* como *samesite* e *secure*. No nosso caso, o token de acesso é guardado com a *flags samesite* e *secure*, o que garante que o token de acesso só é enviado para o servidor quando o pedido é feito a partir do mesmo domínio, não podendo ser acedido por terceiros. O uso do *cookie* também previne ataques de *Cross-site Request Forgery (CSRF)*, visto que o token de acesso é enviado automaticamente com todos os pedidos feitos para o servidor, garantindo que o pedido é autenticado. O uso da *flag httpOnly* também foi tida em consideração, visto que esta previne que o *token* de acesso seja acedido por *scripts* do lado do cliente, porém a sua implementação foi descartada visto ter trazido vários problemas.

4.4 *Backups* dos Dados

Backups recorrentes e extensos do estado atual do sistema são absolutamente críticos numa aplicação deste tipo. Para evitar situações em que, um docente e um aluno discordam sobre os termos de uma proposta acordada na plataforma, a garantia da integridade dos dados e a disponibilização de acesso a versões dos mesmos da data onde o acordo aconteceu são funcionalidades críticas que foram implementadas. Para resolver estas situações e garantir que caso ocorra alguma mudança não esperada no sistema, esta possa ser revertida com o mínimo de danos possíveis, a máquina implementa um serviço interno criado por nós que realiza um backup completo da plataforma diariamente. Este backup inclui todo o código, configurações e ficheiros das bases de dados. Observámos que estes *backups* deveriam ser realizados em horas de pouca afluência à plataforma, pelo que, após monitorizarmos durante algum tempo os períodos regulares em que é feito *login*, decidimos realizar os *backups* três e meia da manhã.

No entanto, esta solução aloca ao disco da máquina uma maior utilização do mesmo apenas para *backups*, pelo que decidimos apenas persistir os *backups* dos últimos oito dias e apagar os antigos restantes. Desta forma surge um problema associado, visto que a época da escolha das dissertações é limitada apenas a certos meses do ano, a plataforma irá ter algum período de inatividade. Caso um erro ocorre-se durante este período de inatividade, e apenas fosse detetado alguns meses depois, o *backup* mais antigo seria apenas o realizado oito dias antes do problema ser detetado e teria também o mesmo problema. Assim, para evitar este problema, criámos outro serviço que no primeiro dia de cada mês copia o *backup* desse dia para uma outra pasta garantindo assim que existe pelo menos um *backup* que nunca é apagado correspondente a cada mês de operação da plataforma.

Resumindo, existem dois tipos de serviços de *backups* que ocorrem durante a execução contínua da base de dados: um serviço de *backups* diários que apenas persiste os últimos oito *backups*, permitindo reduzir ao máximo a perda de informação seguida por um problema no sistema que tenha de ser revertido, e um sistema de backups mensais que permite obter *backups* "a longo prazo"que nunca são apagados, garantindo que durante períodos de alta inatividade o sistema continua a persistir *backups* relevantes.

É de notar que, no momento de escrita deste relatório, os backups são realizados e guardados dentro da própria máquina de produção. Isto pode vir a trazer consequências graves caso exista algum problema com esta máquina. Assim, está neste momento a cargo do administrador realizar uma cópia dos backups para um local seguro. É no entanto de ressalvar que os processos de backups foram desenvolvidos com a integração de um sistema de armazenamento *cloud* automático em vista, sendo apenas necessário fazer algumas adaptações para incluir esta funcionalidade.

4.5.1 Ligação à API

Inicialmente, a ligação entre o Frontend e o Backend foi feita através de pedidos HTTP feitos diretamente pelo Frontend ao Backend, usando a biblioteca *Axios* de React. Para garantir que os dados da API chegavam antes que o componente fosse renderizado, foi necessário utilizar *hooks* de React, como o *useEffect*, e só então guardar os dados no estado do componente.

Desta forma foi possível garantir que os dados da API eram carregados antes que o componente fosse renderizado, garantindo que o utilizador não via um ecrã vazio enquanto os dados eram carregados. No entanto, este método não era o mais eficiente, visto que o componente era renderizado duas vezes, uma vez sem os dados e outra com os dados, o que poderia levar a problemas de performance. Para agravar a situação, este método requeria que os dados necessários a cada componente fossem carregados em cada um dos mesmos, o que poderia levar a uma duplicação de pedidos à API.

Para resolver este problema, optámos por migrar a gestão dos pedidos à API para a biblioteca *@tanstack/react-query*. Esta biblioteca disponibiliza *hooks* que permitem fazer pedidos à API de forma inteligente e acompanhar o estado dos pedidos a cada momento, assim como configurar parâmetros como intervalo de *refetching*, entre outros.

Assim, o *@tanstack/react-query* permitiu que as chamadas à API fossem feitas de forma inteligente. Ao usar os *hooks* fornecidos por esta biblioteca é possível fazer a chamada num componente e identificar a mesma através de uma *queryKey*. Desta forma, se a mesma chamada for feita noutro componente, a biblioteca verifica se os dados já foram carregados e, caso tenham sido, não faz a chamada novamente, garantindo que os dados são carregados apenas uma vez. Isto também permite que as *queries* posssam ser invalidadas ou mudadas, garantindo que os dados são atualizados quando necessário.

Isto foi um *deal breaker* importantíssimo, visto que a aplicação é altamente dependente de dados carregados da API e que a performance da aplicação é crítica para a experiência do utilizador. Através da migração para o *@tanstack/react-query*, foi possível reduzir o número de pedidos à API e gerir a aplicação de acordo com o estado dos pedidos, fazendo com que a renderização condicional dos dados girasse em volta do estado dos pedidos e não da renderização dos componentes.

Desta forma foi possível garantir que os dados da API eram carregados antes que o componente fosse renderizado, garantindo que o utilizador não via um ecrã vazio enquanto os dados eram carregados. No entanto, este método não era o mais eficiente, visto que o componente era renderizado duas vezes, uma vez sem os dados e outra com os dados, o que poderia levar a problemas de performance. Para agravar a situação, este método requiria que os dados necessários a cada componente fossem carregados em cada componente, o que poderia levar a uma duplicação de pedidos à API.

4.5.2 Gestão do estado na aplicação

Outro desafio encontrado foi a gestão do estado da aplicação. Inicialmente, o estado da aplicação era gerido através de *props* e *states* de componentes, o que levava a uma complexidade desnecessária e a uma dificuldade em gerir o estado da aplicação de forma global. Em muitos casos, era necessário passar *flags* e estados de um componente para outro, o que levava a uma complexidade desnecessária e a uma dificuldade em gerir o estado da aplicação de forma global e por cair no problema de *property drilling*.

A decisão de introduzir o *zustand* na aplicação foi tomada para resolver este problema. O *zustand* é uma biblioteca de gestão de estado que permite gerir o estado da aplicação de forma global, sem a necessidade de passar *props* e *states* entre componentes. Esta biblioteca permite criar *stores* que contêm o estado da aplicação e disponibilizar este estado a todos os componentes da aplicação. Desta forma, é possível aceder ao estado da aplicação em qualquer componente, sem a necessidade de passar *props* e *states* entre componentes.

No entanto, o *zustand* introduziu um novo problema: a complexidade de gerir o estado da aplicação de forma global. Tivemos de chegar a um equilíbrio na medida em que foi necessário decidir para cada componente e caso se o estado do mesmo deveria ser gerido localmente ou globalmente. De forma geral, o *zustand* é usado para guardar dados que são necessários em vários componentes da aplicação, como por exemplo, o estado de autenticação do utilizador, os dados dos utilizadores, listas de áreas e cursos, etc. No entanto, o estado de componentes que são usados apenas num componente específico é gerido localmente, visto que não é necessário aceder a estes dados noutros componentes. Apesar de em muitos casos ser necessário passar *props* e *flags* entre componentes, o estado de um modo geral é gerido de forma global.

O caso de uso mais crucial do *zustand* foi na implementação dos filtros, visto que estes são geridos por diversos componentes e requerem que o estado seja atualizado em todos os componentes que usam os filtros. O *zustand* permitiu que os filtros fossem geridos de forma global, passando a função para que estes fossem atualizados de forma consistente em todos os componentes que usam os filtros. Para além disso, os próprios filtros são guardados no *zustand*, permitindo que o estado dos filtros seja acedido em qualquer componente da aplicação.

A combinação do *zustand* com o *@tanstack/react-query* permitiu que a gestão do estado da aplicação fosse feita de forma eficiente e consistente, garantindo que o estado da aplicação seja atualizado de forma consistente em todos os componentes da aplicação.

4.5.3 Problemas com o IdP da Universidade de Aveiro

Existiram vários problemas na implementação de autenticação através do IdP da Universidade de Aveiro. Inicialmente foi-nos proposto que a autenticação fosse feita através do protocolo SAML, requerendo estudo do mesmo pela nossa parte. No entanto, houve vários problemas com a implementação do SAML WSO2 devido à sua complexidade e a necessidade de configuração manual por parte dos STIC para a nossa aplicação.

Sendo assim, foi necessária a mudança para o protocolo UA através do WSO2 Identity Server, que é um serviço de autenticação e autorização que implementa o protocolo OIDC, como mencionado anteriormente.

Isto levou a várias reescritas do sistema de autenticação da aplicação até chegar a um ponto estável. Dado que a aplicação toda dependeria de autenticação federada na UA, este problema foi um dos maiores problemas encontrados durante o desenvolvimento da aplicação, que atrasou significativamente o desenvolvimento do projeto.

Para além disto, foi-nos forçado o uso de HTTPS por parte do IdP, o que levou a várias mudanças na configuração do servidor de produção, com a introdução de certificados SSL e a configuração do servidor para aceitar pedidos HTTPS.

4.5.4 IUPI vs Emails

Por vezes, o IdP da Universidade de Aveiro não fornece o email do utilizador autenticado no formato habitual, mas sim com *live* antes de ua.pt, por exemplo, email@live.ua.pt em vez de email@ua.pt.

Estando nós a indexar cada utilizador através do seu email na base de dados, isto levou a problemas de duplicação de utilizadores, visto que agora poderia existir um utilizador que estava registado na base de dados com dois emails diferentes.

Para resolver este problema, foi necessário passar a indexar o utilizador tanto pelo seu email como pelo seu IUPI, que é um identificador único de cada utilizador na Universidade de Aveiro. Desta forma, um utilizador é primeiro procurado pelo seu email, e caso não seja encontrado, é procurado pelo seu IUPI. Tendo a certeza que o IUPI é único para cada utilizador, garantimos que encontraremos sempre o utilizador em questão caso este esteja registado no sistema, e atualizamos então o seu email para o fornecido pelo IdP na altura da autenticação.

CAPÍTULO 5

Testes e Resultados

5.1 Testes de Usabilidade

Uma vez especificados os requisitos, foi desenvolvido um protótipo de baixa fidelidade, para descobrir erros e áreas a melhorar, assim como obter algumas opiniões sobre a visão geral do sistema. Um protótipo de baixa fidelidade apresenta uma interface aproximada à interface que vai ser desenvolvida e tem a vantagem de poupar tempo aos desenvolvedores, já que os mesmos conseguem testar uma UI antes de ser escrita uma única linha de código e alterações ao protótipo são fáceis, rápidas e baratas de fazer.

5.1.1 Amostra

Com a ajuda dos professores Beatriz Sousa Santos, Paulo Dias e Samuel Silva da Unidade Curricular de Interação Humano-Computador (IHC), no dia vinte e um de março de dois mil e vinte e quatro, conduzimos um teste de Usabilidade com dezanove participantes. Estes participantes, dos quais dezasseis do sexo masculino e três do sexo feminino, eram todos do curso de Licenciatura em Engenharia Informática (LEI) e frequentadores da UC de IHC. Assim, em relação aos estudantes alvo da nossa plataforma, que são aqueles que estão interessados em escolher uma dissertação/estágio, os estudantes envolvidos no teste são mais novos, e por isso foi necessário dar mais algum contexto e esclarecer alguns termos mais específicos relacionados com a própria temática. No entanto, é expectável que parte destes alunos venha a usar o sistema no futuro, por isso o seu *feedback* foi bastante importante já que se estes mesmos alunos tivessem sucesso a usar o sistema, também os estudantes alvo serão capazes de o usar.

5.1.2 Método

O protótipo foi desenhado usando a aplicação web Figma [24] e é possível consultar o protótipo desenvolvido no seguinte *link*.

Os testes começaram com uma descrição daquilo que era o sistema, o que é que estava a ser feito com os testes e quais eram os seus objetivos. Foi dito aos participantes o que iriam fazer, bem como explicado que, se encontrassem alguma dificuldade ao longo do percurso, seria uma falha na interface do utilizador e não culpa deles, para que o participante se sentisse confortável tanto com a equipa de teste como com o próprio teste.

Enquanto cada um dos participantes realizava as cinco tarefas propostas, que podem ser consultadas no Apêndice B, encontrava-se junto dos mesmos a equipa de teste que foi tirando notas usando um Guia de Observador, que pode ser consultado no Apêndice C. Desta forma, o observador estava ao dispôr do participante para qualquer tipo de questão. O observador conseguia assim também perceber se o participante ia ficando preso em algum momento e ia registando tudo o que os utilizadores iam dizendo, sem qualquer interpretação dos atos e/ou palavras do participante.

Depois das tarefas estarem concluídas, foi apresentado um questionário pós-tarefa, disponível no Apêndice D, no qual o participante podia comentar a experiência que o sistema lhe proporcionou. Foi ainda pedido que respondesse a algumas questões de escolha múltipla, que permitem calcular o System Usability Scale [25] e assim, através dessas dez questões, obter uma avaliação quantitativa da usabilidade do sistema.

Por último, já em contexto mais informal, foi também mostrado aos participantes o *frontend* desenvolvido até à data do teste, que contava com boa parte da vista do aluno já desenvolvida. Questionou-se os mesmos se achavam que a página desenvolvida estava consistente com o previamente testado no protótipo Figma e também sugestões do que é que os participantes gostariam que estivesse diferente em relação ao que estava feito.

5.1.3 Resultados

Depois de realizarmos os testes de usabilidade, procedemos ao tratamento dos dados obtidos, de forma a podermos tirar algumas conclusões a partir desses mesmos resultados, para além de notas tomadas pelos observadores assim como comentários dados pelos participantes.

Na figura 5.1, podemos verificar se, para cada tarefa, a mesma foi concluída pelos participantes ou não. Assim, registámos 96.8% de rácio de conclusão de tarefas.

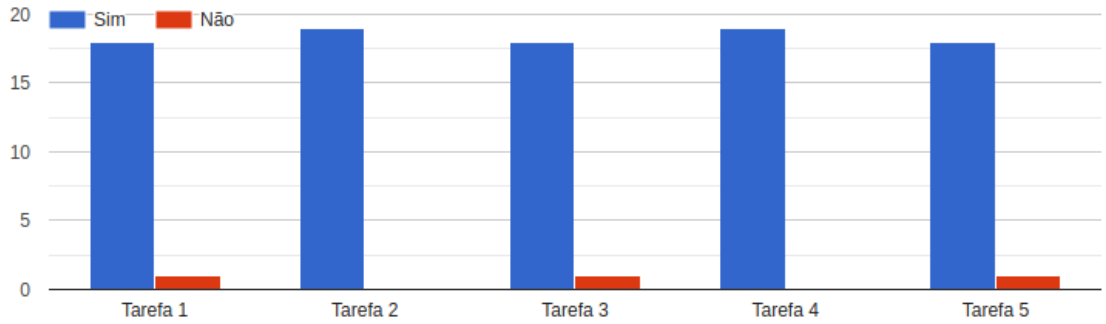


Figura 5.1: Conclusão das tarefas no teste de Usabilidade

Relacionado também com estes dados estão as tarefas às quais os participantes chegaram à resposta certa, que se encontram na figura 5.2, tendo sido essa relação de 78.9%.

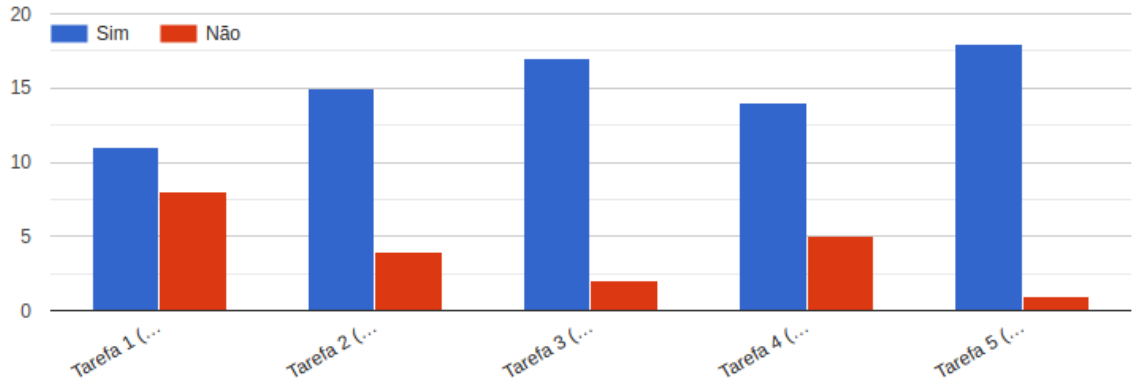


Figura 5.2: Conclusão das tarefas com sucesso no teste de Usabilidade

Por último, como referido na subsecção anterior, foi fornecido aos participantes um questionário pós-tarefa, a partir do qual se pode calcular o valor do SUS. Na figura 5.3 podemos encontrar a mediana da pontuação por questão. É usada a mediana e não a média, para suprimir eventuais erros dos participantes aquando da resposta ao questionário, visto que, para interpretação da figura seguinte, deve ter-se em consideração que em questões de número par, a melhor pontuação a obter é 5, enquanto que nas de número ímpar é 1. Este procedimento é realizado de forma a evitar que o inquirido tenha a tentação de preencher a coluna toda de alto a baixo com o mesmo valor.

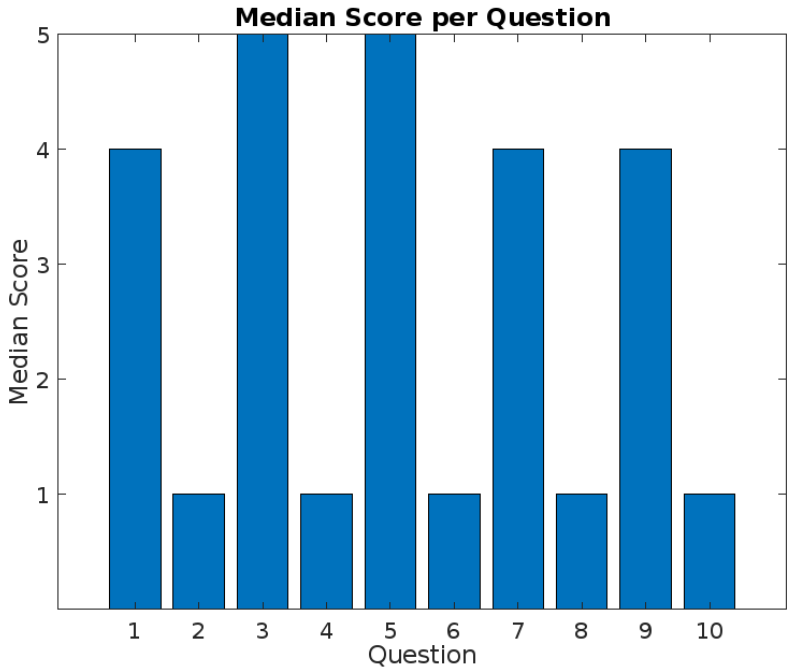


Figura 5.3: Pontuação mediana por questão

Com estes dados, calculámos o nosso SUS, obtendo uma pontuação de 91. Como esse valor é maior do que 68, isto significa que podemos considerar o sistema usável.

No que toca a apreciações/anotações acerca do sistema, retirámos das mesmas algumas considerações:

- A maior parte dos participantes, apesar de ser a primeira vez que usavam este sistema, acharam-no intuitivo e fácil de entender;
- A tarefa 2 foi, na maior partes das vezes, completada usando um clique extra, já que toda a informação estava visível, mas alguns dos participantes optavam na mesma por carregar no botão "Ver mais";
- Devia ser adicionado ao sistema algum tipo de legendas, que ajudasse a clarificar os termos e expressões usadas;
- O uso do termo "Escolhida"estava a ser confundido, pois quando uma dissertação/estágio a tinha, alguns participantes achavam que essa mesma proposta tinha ficado com acordo fechado e escolhida por eles;
- Notificações deviam ser mais visíveis, através do uso de alertas no *frontend*. Esta funcionalidade veio a ser substituída em detrimento do sistema de e-mails que se encontra até ao momento implementado.

5.2 Testes de Carga

De modo a medir a performance a API, foram realizados testes de carga utilizando a ferramenta [26].

Este tipo de teste é importante para verificar se a aplicação é capaz de suportar um determinado número de utilizadores, ou seja, se a aplicação é escalável. É de notar que atualmente existe apenas 1 instância da API a correr, pelo que os resultados obtidos podem ser melhorados com a adição de mais instâncias (escalabilidade horizontal).

Os resultados obtidos foram os seguintes:

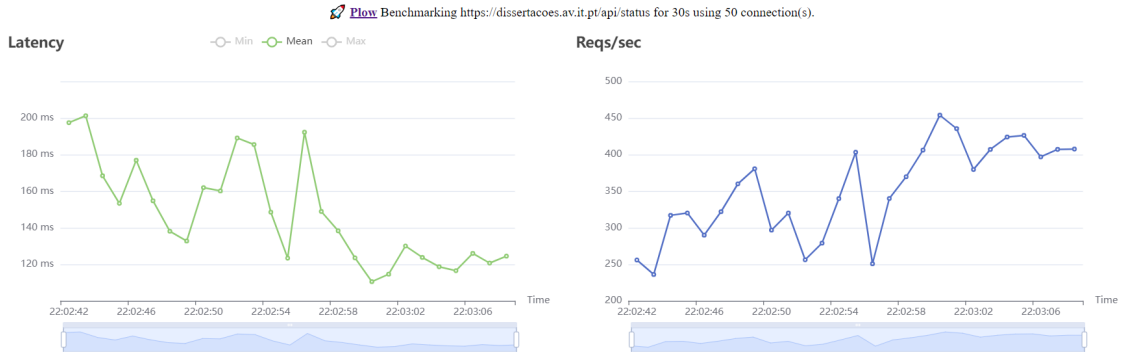


Figura 5.4: Latência de resposta média e número de pedidos por segundo

```
Benchmarking https://dissertacoes.av.it.pt/api/status for 30s using 50 connection(s).
@ Real-time charts is listening on http://[::]:18888

Summary:
  Elapsed        30s
  Count        10221
    2xx        10221
  RPS        340.697
  Reads    0.068MB/s
  Writes   0.032MB/s

Statistics     Min       Mean      StdDev      Max
  Latency   86.519ms  146.355ms  42.234ms  564.298ms
  RPS         191.37    340.64     71.17     453.89

Latency Percentile:
  P50           P75        P90        P95        P99       P99.9     P99.99
  132.432ms  160.615ms  206.862ms  229.619ms  280.459ms  489.839ms  564.298ms

Latency Histogram:
  113.515ms  2592  25.36%
  130.397ms  3869  37.85%
  156.738ms  2009  19.66%
  196.265ms  1005   9.83%
  231.453ms   572   5.60%
  278.574ms   138   1.35%
  367.269ms    29   0.28%
  505.39ms      7   0.07%
```

Figura 5.5: Todas as estatísticas do teste de carga

Neste teste foram simuladas 50 conexões a enviar pedidos à API durante 30 segundos. A latência média de resposta foi de 146 ms e o número médio de pedidos por segundo foi de 340. O pico de latência de resposta, 564 ms, ocorreu quando o número de pedidos por segundo foi de 453. O mínimo valor de latência registado foi de 86 ms, quando o número de pedidos por segundo foi de 191.

Analisando estes dados, podemos concluir que a API é capaz de suportar um número elevado de pedidos por segundo, muito maior que o esperado e a realidade atual, mantendo uma latência de resposta aceitável.

CAPÍTULO 6

Conclusão

6.1 Resumo do Projeto

Este projeto começou com a compreensão de como funcionava o processo de publicitação e atribuição de dissertações/estágios no DETI, e alguma linguagem específica associada. Além disso, houve também o estudo da plataforma antiga, descrita no capítulo 2. Concluiu-se que vulnerabilidades de segurança, bugs inexplorados, funcionalidades abandonadas e UI desatualizada, associados a tecnologias antigas, fariam com que o sistema a desenvolver teria de ser totalmente novo, sem qualquer ligação a esse código *legacy*.

Através de reuniões com o professor orientador e o administrador anterior da plataforma, foram definidos objetivos e funcionalidades que o novo sistema deveria ter. Esses requisitos do sistema e arquitetura foram especificados em detalhe no capítulo 3.

Quando o sistema já se encontrava devidamente planeado e prototipado, foram realizados testes de usabilidade com alunos, testando o protótipo desenvolvido naquela fase. Esta fase, documentada no capítulo 5, resultou em algumas alterações que vieram depois a ser implementadas na parte seguinte do Projeto. A aplicação web desenvolvida em React e que conta também com bases de dados MariaDB, MongoDB e armazenamento de ficheiros, implementou todos os casos de uso previamente especificados no capítulo 3, e conta ainda com suporte para ecrãs mais pequenos como telemóveis. A implementação encontra-se descrita no capítulo 4.

6.2 Resultados Principais

Devido a termos conseguido cumprir com as datas originalmente propostas, à data de escrita deste relatório temos a nossa aplicação a ser usada pela comunidade do DETI há praticamente dois meses, entre uma fase beta e fase final, que foi *deployed* no início do mês de maio. Assim, destacamos também algumas estatísticas registadas até ao momento, como o facto de 320 utilizadores terem já efetuado *login* na nossa plataforma, 94 dissertações foram submetidas e ainda 6 acordos que já se encontram fechados para dissertações no próximo ano.

Além disso, destacamos os requisitos inicialmente propostos:

- Integração com o IdP da UA;
- Listar dissertações;
- Casamento (acordo entre o docente e o estudante);
- Submeter dissertações;
- Mostrar interesse em dissertações;
- Filtrar a lista por campos chave;
- Pesquisa avançada com *tags*;
- Exportar o estado das propostas para a secretaria;

E todas as funcionalidades desenvolvidas com sucesso:

- Integração com o IdP da UA;
- Listar dissertações;
- Casamento (acordo entre o docente e o estudante);
- Submeter dissertações;
- Mostrar interesse em dissertações;
- Filtrar a lista por campos chave;
- Pesquisa avançada com *tags*;
- Exportar o estado das propostas para a secretaria;
- Novo ator - Diretores de curso;
- Diretor pode aceitar ou recusar o uso do seu curso em propostas;
- Diretor pode cancelar acordos;
- Admins podem gerir áreas e cursos;
- Editar dissertações;
- Trazer dissertações abertas para o ano corrente;
- Verificar as quotas dos docentes por parte dos admins;
- Página de monitorização do sistema;
- Página de ajuda;
- Manual de utilização para docentes e estudantes;
- Perfil público;
- Introdução de *alias* para docentes;
- Revogar curso de uma dissertação por parte do diretor;
- Invalidar dissertação por parte do administrador;
- Novo ator - Funcionários da secretaria;

- Páginas responsivas para estudantes;

Destacamos que apesar de termos uma *deadline* de lançamento para o início de maio, o grupo cumpriu com sucesso todos os requisitos e milestones propostos, tendo a aplicação sido lançada no início de maio, como planeado. Dado que todos os requisitos iniciais foram cumpridos, e ainda mais desenvolvidos, consideramos que o projeto foi um sucesso.

6.3 Trabalho Futuro

Apesar de todos os objetivos iniciais terem sido mais do que cumpridos, alguns aspetos que surgiram mais à frente podem ser refinados e outros mudados, apesar de estarem dependentes de entidades externas:

- O código do *frontend* necessita de ser revisto e a sua documentação melhorada;
- Tornar o processo de *deploy* mais automático, usando *workflows* do GitHub;
- À data de escrita deste relatório, o sistema encontra-se hospedado num servidor do IT. Sendo um produto direcionado ao DETI, é expectável que o mesmo venha a ser transferido para uma máquina dos STIC no futuro, e algumas adaptações vão ter de ocorrer para essa transferência ser concluída com sucesso;
- O atributo dos ECTS ainda não está a ser disponibilizado pelos STIC. No entanto, é expectável que o mesmo venha a acontecer no futuro;
- Migrar para um sistema de orquestração de *containers* baseado em Kubernetes. Apesar de esta alteração não proporcionar grande vantagem face à implementação no ambiente de produção atual, em caso de migração para os servidores dos STIC, Kubernetes permitem gerir recursos de forma mais eficiente e contam ainda com ferramentas de monitorização;

Por último, existe a possibilidade de o sistema poder vir a ser usado por mais departamentos ou até pela UA toda em geral. Em ambos os casos, terá de haver um novo levantamento de requisitos visto que as regras variam de departamento para departamento, para que depois possam ser feitas as adaptações necessárias.

Referências

[1] *Microsoft*. URL: https://dotnet.microsoft.com/pt-br/apps/aspnet.

[2] K. E. Wiegers, *Software Requirements*, Segunda Edição. Redmond, WA, USA: Microsoft Press, 2003.

[3] *Nginx*. URL: https://nginx.org/en/.

[4] *CertBot*. URL: https://certbot.eff.org/.

[5] *ViteJS*. URL: https://vitejs.dev/.

[6] *React*. URL: https://react.dev/.

[7] *FastAPI*. URL: https://fastapi.tiangolo.com/.

[8] *Uvicorn*. URL: https://www.uvicorn.org/.

[9] *Pydantic*. URL: https://docs.pydantic.dev/latest/.

[10] *MariaDB*, nov. de 2019. URL: https://mariadb.org/.

[11] *MongoDB*. URL: https://www.mongodb.com/.

[12] *SendMail*, mai. de 2024. URL: https://www.proofpoint.com/us/products/email-protection/open-source-email-solution.

[13] *Docker*. URL: https://www.docker.com/.

[14] *TailwindCSS*. URL: https://tailwindcss.com/.

[15] *CSS*. URL: https://www.w3.org/TR/CSS/#css.

[16] *yup*. URL: https://github.com/jquense/yup.

[17] *ReactHookForm*. URL: https://www.react-hook-form.com/.

[18] *zustand*. URL: https://zustand-demo.pmnd.rs/.

[19] *Jinja*. URL: https://jinja.palletsprojects.com/en/3.1.x/.

[20] *APIRestFul*. URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[21] *Spring*. URL: https://spring.io/.

[22] *NodeJS*. URL: https://nodejs.org/en.

[23] *Unix*. URL: https://www.opengroup.org/membership/forums/platform/unix.

[24] *Figma*. URL: https://www.figma.com/.

[25] A. Bhat, *System usability scale: What it is, calculation + usage*, ago. de 2023. URL: https://www.questionpro.com/blog/system-usability-scale/.

[26] *plow*. URL: https://github.com/six-ddc/plow.

APÊNDICE A

Apêndice A - Documentação da API

Dissertacoes API

Overview

The Dissertacoes API provides an easy way to access and manage the dissertation season for the DETI department at the University of Aveiro.

It allows users to search for dissertations, teachers, and courses, as well as manage their own dissertations.

It has built-in authentication and authorization mechanisms, allowing teachers to manage their own dissertations and students to show interest in them.

Check out our links below:

- [Website](https://dissertacoes.av.it.pt)
- [Documentation Microsite](https://pi-dsd.github.io/microsite/)
- [GitHub Repository](https://github.com/detiuaveiro/dsd)

Apache 2.0

Paths

GET `/docs` Swagger Ui

Swagger UI for API documentation

Responses

Code	Description	Links
200	Swagger UI *Content* **application/json**	No Links

GET `/user/my_dissertations` List Dissertations

Retrieve all the dissertations of the user

Responses

Code	Description	Links
200	Dissertations retrieved *Content* **application/json**	No Links

GET `/user/list/latest/{number}` List Latest Logins

Retrieve the latest logins

Parameters

Type	Name	Description	Schema
path	**number** *required*		integer

Responses

Code	Description	Links
200	Latest logins retrieved *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

POST `/user/media` Insert Media

Parameters

Type	Name	Description	Schema
query	**media** *required*		number

Responses

Code	Description	Links
200	Successful Response *Content* **application/json**	No Links

Code	Description	Links
422	Validation Error *Content* **application/json**	No Links

GET `/teacher/my_dissertations` List Teacher Dissertations

Retrieve all the dissertations of the teacher

Responses

Code	Description	Links
200	Dissertations retrieved *Content* **application/json**	No Links

GET `/teacher/my_dissertations/years` List Teacher Dissertations Years

Retrieve all the years of the dissertations of the teacher

Responses

Code	Description	Links
200	Years retrieved *Content* **application/json**	No Links

GET `/teacher/my_dissertations/acordos` List Teacher Agreements

Retrieve all the agreements of the teacher

Parameters

Type	Name	Description	Schema
query	**year** *required*		string

Responses

Code	Description	Links
200	Agreements retrieved *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/teacher/my_dissertations/acordos/{dissertation_id}` Get Teacher Dissertations Interest Status

Retrieve interest status of the given dissertation of the teacher

Parameters

Type	Name	Description	Schema
path	**dissertation_id** *required*		string

Responses

Code	Description	Links
200	Interests retrieved *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/teacher/emails` List Teacher Emails

Retrieve all the emails of the teachers

Responses

Code	Description	Links
200	Emails retrieved *Content* **application/json**	No Links

GET `/teacher/my_acordos` List Teacher Agreements

Retrieve all the agreements of the teacher

Responses

Code	Description	Links
200	Agreements retrieved *Content* **application/json**	No Links

POST `/teacher/alias` Update Teacher Alias

Update the alias of the teacher

Parameters

Type	Name	Description	Schema
query	**alias** *required*		string

Responses

Code	Description	Links
200	Alias updated *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/teacher/info/{email}` Get Teacher Public Info

Retrieve the public info of the teacher by email

Parameters

Type	Name	Description	Schema
path	**email** *required*		string

Responses

Code	Description	Links
200	Teacher public info retrieved *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/teacher/dissertations/{email}` Get Teacher Public Dissertations

Retrieve the public dissertations of the teacher by email

Parameters

Type	Name	Description	Schema
path	**email** *required*		string

Responses

Code	Description	Links
200	Teacher public dissertations retrieved *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/teacher/alias/{email}` Get Teacher Alias

Retrieve the alias of the teacher by email

Parameters

Type	Name	Description	Schema
path	**email** *required*		string

Responses

Code	Description	Links
200	Teacher alias retrieved *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

POST `/dissertation/add` Add Dissertation

Add a new dissertation to the database.

Responses

Code	Description	Links
200	Add a new dissertation *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

PUT `/dissertation/edit/{dissertationID}` Edit Dissertation

Edit a dissertation in the database.

Parameters

Type	Name	Description	Schema
path	**dissertationID** *required*		string

Responses

Code	Description	Links
200	Edit a dissertation *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/dissertation/list` Get Dissertations

Get a list of dissertations from the database.

Responses

Code	Description	Links
200	Get a list of dissertations *Content* **application/json**	No Links

GET `/dissertation/get/{id}` Get Dissertation

Get a dissertation from the database.

Parameters

Type	Name	Description	Schema
path	**id** *required*		string

Responses

Code	Description	Links
200	Get a dissertation *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/dissertation/{id}/document` Get Dissertation File

Get a dissertation file from the database.

Parameters

Type	Name	Description	Schema
path	**id** *required*		string

Responses

Code	Description	Links
200	Get a dissertation file *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/dissertation/{id}/image` Get Dissertation Image

Get a dissertation image from the database.

Parameters

Type	Name	Description	Schema
path	**id** *required*		string

Responses

Code	Description	Links
200	Get a dissertation image *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/dissertation/{id}/memorando` Get Dissertation Memorando

Get a dissertation memorando from the database.

Parameters

Type	Name	Description	Schema
path	**id** *required*		string

Responses

Code	Description	Links
200	Get a dissertation memorando *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/dissertation/images` Get Default Images

Get a list of default images from the database.

Responses

Code	Description	Links
200	Get a list of default images *Content* **application/json**	No Links

GET `/dissertation/years` Get Dissertation Years

Get a list of dissertation years from the database.

Responses

Code	Description	Links
200	Get a list of dissertation years *Content* **application/json**	No Links

GET `/dissertation/current_year` Get Current Year

Get the current year from the database.

Responses

Code	Description	Links
200	Get the current year *Content* **application/json**	No Links

PUT `/dissertation/{dissertationID}/update_year` Update Dissertation Year

Update a dissertation year in the database.

Parameters

Type	Name	Description	Schema
path	**dissertationID** *required*		string
query	**year** *required*		string

Responses

Code	Description	Links
200	Update a dissertation year *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/tags/list` List Tags

Retrieve all the tags

Responses

Code	Description	Links
200	Tag retrieved *Content* **application/json**	No Links

POST `/tags/add` Add Tag

Add a tag

Parameters

Type	Name	Description	Schema
query	**name** *required*		string

Responses

Code	Description	Links
200	Tag added *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/courses/list` List Courses

Retrieve all the courses

Responses

Code	Description	Links
200	Courses retrieved *Content* **application/json**	No Links

GET `/courses/list/diretores` List Courses Directors

Retrieve all the courses with their directors

Responses

Code	Description	Links
200	Courses with directors retrieved *Content* **application/json**	No Links

POST `/courses/add` Add Course

Add a new course

Parameters

Type	Name	Description	Schema
query	**codigo** *required*		string
query	**name** *required*		string
query	**fullname** *required*		string

Responses

Code	Description	Links
200	Course added *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/list/getFileName` Get Excel File Name

Get the current year incorporated into a filename

Responses

Code	Description	Links
200	Successful Response *Content* **application/json**	No Links

GET `/list/generate` Generate Dissertations Report

Generate a file with the list of dissertations

Responses

Code	Description	Links
200	Successful Response *Content* **application/json**	No Links

GET `/statusAPI` Api Status

Check if the API is alive

Responses

Code	Description	Links
200	API Status *Content* **application/json**	No Links

GET `/login_callback` Login Callback

Backend Login Callback for WSO2 response

Parameters

Type	Name	Description	Schema
query	**code** *optional*		string

Responses

Code	Description	Links
200	Backend Login Callback *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/user_info` Get User Info

Retrieve user info from the database

Responses

Code	Description	Links
200	Get logged in user info *Content* **application/json**	No Links

GET `/refresh_token` Refresh Token

Refresh the access token when it expires

Responses

Code	Description	Links
200	Refresh access token *Content* **application/json**	No Links

POST `/logout` Logout

Logout the user

Responses

Code	Description	Links
200	Logout *Content* **application/json**	No Links

GET `/checklogin` Check Login

Check if the user is logged in based on the access token

Responses

Code	Description	Links
200	Check if user is logged in *Content* **application/json**	No Links

GET `/checkteacher` Check Teacher

Check if the user is a teacher

Responses

Code	Description	Links
200	Check if user is a teacher *Content* **application/json**	No Links

GET `/checkadmin` Check Admin

Check if the user is an admin

Responses

Code	Description	Links
200	Check if user is an admin *Content* **application/json**	No Links

GET `/checkdirector` Check Director

Check if the user is a director of any course

Responses

Code	Description	Links
200	Check if user is a director *Content* **application/json**	No Links

POST `/admin/accept/{id}` Accept Dissertation

Accept a dissertation

Parameters

Type	Name	Description	Schema
path	**id** *required*		string

Responses

Code	Description	Links
200	Accept a dissertation *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

POST `/admin/deny/{id}` Deny Dissertation

Parameters

Type	Name	Description	Schema
path	**id** *required*		string

Responses

Code	Description	Links
200	Successful Response *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/admin/list` Get Unaccepted Dissertations

Retrieve dissertations that are not accepted yet

Responses

Code	Description	Links
200	Get Unaccepted Dissertations *Content* **application/json**	No Links

GET `/admin/list/limbo` Get Dissertations In Limbo

Retrieve dissertations that are in limbo state, not accepted by any director yet.

Responses

Code	Description	Links
200	Get Limbo Dissertations *Content* **application/json**	No Links

GET `/admin/listTeachers/{number}` Get Teachers Quotas

Retrieve teachers quotas

Parameters

Type	Name	Description	Schema
path	**number** *required*		integer

Responses

Code	Description	Links
200	Get Teachers Quotas *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

POST `/admin/update_current_year` Update Current Year

Update the current year

Responses

Code	Description	Links
200	Update the current year *Content* **application/json**	No Links

GET `/admin/get_current_and_calculated_year` Get The Current System Year And Current Academic Year

Get the current system year and current academic year

Responses

Code	Description	Links
200	Get the current system year and current academic year *Content* **application/json**	No Links

PUT `/admin/invalidate/{dissertation_id}` Invalidate Dissertation

Invalidate a dissertation

Parameters

Type	Name	Description	Schema
path	**dissertation_id** *required*		string

Responses

Code	Description	Links
200	Invalidate a dissertation *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/search/list` Get Search Attributes

Retrieve dissertation search attributes

Responses

Code	Description	Links
200	Search attributes retrieved *Content* **application/json**	No Links

POST `/interest/register_interest_student` Register Interest Student

This endpoint is used by students to register their interest in a dissertation. The student must provide the dissertation ID in the query parameters. The student must be authenticated and have the role of student. The student can only register interest in a dissertation if the dissertation is available, has been accepted by an admin and the student has not already registered interest in a dissertation for the current year. If the student is successful in registering interest, an email is sent to the teacher of the dissertation.

Parameters

Type	Name	Description	Schema
query	**dissertationID** *required*		string

Responses

Code	Description	Links
200	Register interest of a student in a dissertation *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

POST `/interest/register_interest_teacher` Register Interest Teacher

This endpoint is used by teachers to register their interest in a student's application for a dissertation. The teacher must provide the dissertation ID and the student ID in the query parameters. The teacher must be authenticated and have the role of teacher. The teacher can only register interest in a student's application if the student has already registered interest in the dissertation and the teacher is the orientador of the dissertation. If the teacher is successful in registering interest, an email is sent to the student.

Parameters

Type	Name	Description	Schema
query	**studentID** *required*		string
query	**dissertationID** *required*		string

Responses

Code	Description	Links
200	Register interest of a teacher in a student's application for a dissertation *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

POST `/interest/confirm_interest_student` Confirm Interest Student

This endpoint is used by students to confirm their interest in a dissertation. The student must provide the dissertation ID in the query parameters. The student must be authenticated and have the role of student. The student can only confirm interest in a dissertation if the teacher has shown interest in the student's application for the dissertation. If the student is successful in confirming interest, an email is sent to the teacher of the dissertation.

Parameters

Type	Name	Description	Schema
query	**dissertationID** *required*		string

Responses

Code	Description	Links
200	Confirm interest of a student in a dissertation *Content* **application/json**	No Links

Code	Description	Links
422	Validation Error *Content* **application/json**	No Links

POST `/interest/confirm_interest_teacher` Confirm Interest Teacher

This endpoint is used by teachers to confirm their interest in a student's application for a dissertation. The teacher must provide the dissertation ID and the student ID in the query parameters. The teacher must be authenticated and have the role of teacher. The teacher can only confirm interest in a student's application if the student has already confirmed interest in the dissertation and the teacher is the orientador of the dissertation. If the teacher is successful in confirming interest, an email is sent to the student.

Parameters

Type	Name	Description	Schema
query	**studentID** *required*		string
query	**dissertationID** *required*		string

Responses

Code	Description	Links
200	Confirm interest of a teacher in a student's application for a dissertation *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

POST `/interest/remove_interest_student` Remove Interest Student

This endpoint is used by students to remove their interest in a dissertation. The student must provide the dissertation ID in the query parameters. The student must be authenticated and have the role of student. The student can only remove interest in a dissertation if the student has already registered interest in the dissertation. If the student is successful in removing interest, an email is

sent to the teacher of the dissertation.

Parameters

Type	Name	Description	Schema
query	**dissertationID** *required*		string

Responses

Code	Description	Links
200	Remove interest of a student in a dissertation *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

POST `/diretor/assign` Assign Director

Assign a teacher as a director of a course.

Parameters

Type	Name	Description	Schema
query	**curso** *required*		
query	**email** *required*		

Responses

Code	Description	Links
200	Director assigned successfully *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

POST `/diretor/remove` Remove Director

Remove a teacher as a director of a course.

Parameters

Type	Name	Description	Schema
query	**curso** *required*		

Responses

Code	Description	Links
200	Director removed successfully *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/diretor/get` Get Director

Retrieve the director of a course.

Parameters

Type	Name	Description	Schema
query	**curso** *required*		

Responses

Code	Description	Links
200	Director retrieved successfully *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/diretor/list` List Directors

Retrieve all the directors.

Responses

Code	Description	Links
200	Directors retrieved successfully *Content* **application/json**	No Links

POST `/diretor/accept_curso` Accept Course

Accept the course in the dissertation.

Parameters

Type	Name	Description	Schema
query	**dissertationID** *required*		string
query	**curso** *required*		string

Responses

Code	Description	Links
200	Course accepted successfully *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

POST `/diretor/deny_curso` Deny Course

Deny the course in the dissertation.

Parameters

Type	Name	Description	Schema
query	**dissertationID** *required*		string

Type	Name	Description	Schema
query	**curso** *required*		string

Responses

Code	Description	Links
200	Course denied successfully *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

POST `/diretor/reset_curso` Reset Course

Reset the course state in the dissertation.

Parameters

Type	Name	Description	Schema
query	**dissertationID** *required*		string
query	**curso** *required*		string

Responses

Code	Description	Links
200	Course state reset successfully *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/diretor/dissertations` Get Director Dissertations

Retrieve all the dissertations that the user is director of.

Responses

Code	Description	Links
200	Dissertations retrieved successfully *Content* **application/json**	No Links

POST `/diretor/cancel_agreement` Cancel Agreement

Cancel the agreement of a dissertation.

Parameters

Type	Name	Description	Schema
query	**dissertationID** *required*		

Responses

Code	Description	Links
200	Agreement canceled successfully *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/diretor/dissertation_logs/{number}` Get Director Logs

Retrieve all the logs of the dissertations that the user is director of.

Parameters

Type	Name	Description	Schema
path	**number** *required*		integer

Responses

Code	Description	Links
200	Logs retrieved successfully *Content* **application/json**	No Links
422	Validation Error *Content* **application/json**	No Links

GET `/status` Alive

Responses

Code	Description	Links
200	Successful Response *Content* **application/json**	No Links

Components

Schemas

Body_Add_Dissertation_dissertation_add_post

Title: Body_Add_Dissertation_dissertation_add_post

Properties

Name	Description	Schema
dissertation_pdf *required*	**Title**: Dissertation Pdf	string (binary)
memorando *optional*	**Title**: Memorando	string (binary)
dissertation_data *required*	dissertation data in JSON Format. **Title**: Dissertation Data	(json)

Body_Edit_Dissertation_dissertation_edit*dissertationID*put

Title: Body_Edit_Dissertation_dissertation_edit*dissertationID*put

Properties

Name	Description	Schema
dissertation_data *required*	dissertation data in JSON Format. **Title**: Dissertation Data	(json)
dissertation_pdf *optional*	**Title**: Dissertation Pdf	string (binary)
memorando *optional*	**Title**: Memorando	string (binary)

HTTPValidationError

Title: HTTPValidationError

Properties

Name	Description	Schema
detail *optional*	**Title**: Detail	< ValidationError > array

ValidationError

Title: ValidationError

Properties

Name	Description	Schema
loc *required*	**Title**: Location	< > array
msg *required*	**Title**: Message	string
type *required*	**Title**: Error Type	string

APÊNDICE B

Apêndice B - Guião do Teste de Usabilidade

Teste de Usabilidade

* Indicates required question

1. **Tarefa 1 -** Fazer login e indicar quantas dissertações de mestrado estão presentes no website e quanta(s) é que estão disponíveis. *

2. Dificuldade da **Tarefa 1** *

   Mark only one oval.

   - 1 (Muito difícil)
   - 2 (Difícil)
   - 3 (Média)
   - 4 (Fácil)
   - 5 (Muito Fácil)

3. **Tarefa 2 -** Analisar a dissertação UI Design principles and their effects in network
   traffic e indicar o nome do orientador, co-orientador e os cursos a que é destinada. *

4. Dificuldade da **Tarefa 2** *

Mark only one oval.

- 1 (Muito difícil)
- 2 (Difícil)
- 3 (Média)
- 4 (Fácil)
- 5 (Muito Fácil)

5. **Tarefa 3 -** Mostrar interesse a dissertação AI-based River Discharge Forecasting e indicar qual a mensagem que apareceu para a confirmação da escolha. *

_____

6. Dificuldade da **Tarefa 3** *

Mark only one oval.

- 1 (Muito difícil)
- 2 (Difícil)
- 3 (Média)
- 4 (Fácil)
- 5 (Muito Fácil)

7. **Tarefa 4 -** Remover o interesse na dissertação Porcupine: Visualization and Analysis of Multimodal Interaction e indicar o número de teses que estão presentes no perfil e o código do curso do aluno. *

_____

8. Dificuldade da **Tarefa 4** *

Mark only one oval.

- 1 (Muito difícil)
- 2 (Difícil)
- 3 (Média)
- 4 (Fácil)
- 5 (Muito Fácil)

9. **Tarefa 5 -** Verificar a nova notificação recebida e assinar o acordo já assinado pelo professor. Indicar o local da dissertação atribuída e a mensagem de confirmação. *

_____

10. Dificuldade da **Tarefa 5** *

Mark only one oval.

- 1 (Muito difícil)
- 2 (Difícil)
- 3 (Média)
- 4 (Fácil)
- 5 (Muito Fácil)

11. Comentários adicionais

_____

_____

_____

_____

_____

APÊNDICE C

Apêndice C - Guião do Observador

Guião do Observador

Testes de usabilidade

* Indicates required question

1. Escolher o nome do observador *

   Mark only one oval.

   - ◯ Daniel
   - ◯ João
   - ◯ José
   - ◯ Pedro
   - ◯ Rodrigo

2. O utilizador conclui a tarefa: *

   Mark only one oval per row.

	Sim	Não
Tarefa 1	◯	◯
Tarefa 2	◯	◯
Tarefa 3	◯	◯
Tarefa 4	◯	◯
Tarefa 5	◯	◯

3. O utilizador completou a task com os resultados corretos: *

Mark only one oval per row.

	Sim	Não
Tarefa 1 (Resposta - 3 dissertações e 2 disponíveis)	◯	◯
Tarefa 2 (Resposta - orientador:Josefino Calças co-orientador: Joana Saia cursos: MEI e MCTW)	◯	◯
Tarefa 3 (Resposta - mensagem: Adicionado interesse com sucesso)	◯	◯
Tarefa 4 (Resposta - nº dissertações perfil: 2 nºcurso:4321)	◯	◯
Tarefa 5 (Resposta - local: DETI mensagem: Acordo Assinado com sucesso e Dissertação atribuída	◯	◯

4. Tempo estimado para a completação de cada uma das tarefas *

Mark only one oval per row.

	30s-40s	20s-30s	10s-30s	40s-50s	50s-1min	1min-2min	2min-4min
Tarefa 1	◯	◯	◯	◯	◯	◯	◯
Tarefa 2	◯	◯	◯	◯	◯	◯	◯
Tarefa 3	◯	◯	◯	◯	◯	◯	◯
Tarefa 4	◯	◯	◯	◯	◯	◯	◯
Tarefa 5	◯	◯	◯	◯	◯	◯	◯

5. Número de erros cometidos *

Mark only one oval per row.

	1	2	3	4	5+	0
Tarefa 1	◯	◯	◯	◯	◯	◯
Tarefa 2	◯	◯	◯	◯	◯	◯
Tarefa 3	◯	◯	◯	◯	◯	◯
Tarefa 4	◯	◯	◯	◯	◯	◯
Tarefa 5	◯	◯	◯	◯	◯	◯

6. Severidade dos erros cometidos *

Mark only one oval per row.

	Nenhuma	Pouca	Alguma	Muita	Extrema
Tarefa 1					
Tarefa 2					
Tarefa 3					
Tarefa 4					
Tarefa 5					

7. O utilizador ficou perdido em algum momento: *

Mark only one oval per row.

	Sim	Não
Tarefa 1		
Tarefa 2		
Tarefa 3		
Tarefa 4		
Tarefa 5		

8. O utilizador precisou de ajuda: *

Mark only one oval per row.

	Sim	Não
Tarefa 1		
Tarefa 2		
Tarefa 3		
Tarefa 4		
Tarefa 5		

9. Facilidade/Dificuldade obervada na completação da task: *

Mark only one oval per row.

	Muito Complexo	Complexo	Razoável	Simples	Muito fácil
Tarefa 1					
Tarefa 2					
Tarefa 3					
Tarefa 4					
Tarefa 5					

10. Observações adicionais

APÊNDICE D

Apêndice D - Questionário Pós-Tarefas

Post Task Questionnaire

Instructions: Thank you for your cooperation with this study, which aims to evaluate the User Interface of the application/system and, try to improve it following the Usability criteria.
Your collaboration is important for the success of this evaluation, so we ask you to complete this questionnaire, the data of which will be used in total anonymity for scientific purposes only.

1. Demographic data

Gender: ☐ Female ☐ Male

Previous experience with this type of application/system: ☐ None ☐ Some ☐ A lot

Observations (fill in any relevant facts for this test, e.g. vision, handiness):

2. Overall opinion on the application/system (SUS)

After using the application/system and taking into account your final assessment, check the circle that best reflects your opinion regarding its usage. If you believe that these quantifications are not applicable, choose NA.

I think that I would like to use this system frequently.	Totally agree	○○○○○	Totally disagree	NA
I found the system unnecessarily complex.	Totally agree	○○○○○	Totally disagree	NA
I thought the system was easy to use.	Totally agree	○○○○○	Totally disagree	NA
I think that I would need the support of a technical person to be able to use this system.	Totally agree	○○○○○	Totally disagree	NA
I found the various functions in this system were well integrated.	Totally agree	○○○○○	Totally disagree	NA
I thought there was too much inconsistency in this system.	Totally agree	○○○○○	Totally disagree	NA
I would imagine that most people would learn to use this system very quickly.	Totally agree	○○○○○	Totally disagree	NA
I found the system very cumbersome to use.	Totally agree	○○○○○	Totally disagree	NA
I felt very confident using the system.	Totally agree	○○○○○	Totally disagree	NA
I needed to learn a lot of things before I could get going with this system.	Totally agree	○○○○○	Totally disagree	NA

Please leave any comments about the user experience provided by the application/system:

_____
_____
_____
_____

Thank you very much for your collaboration!

APÊNDICE E

Apêndice E - Manual de Utilizador do Aluno

Universidade de Aveiro

DETI

Guião de apoio para a plataforma Dissertações

Introdução

Este documento serve de apoio à nova plataforma **Dissertações**, que se encontra disponível no link: https://dissertacoes.av.it.pt

Nesta plataforma, como aluno, poderá escolher um tema de dissertação/estágio para os vários mestrados lecionados pelo DETI.

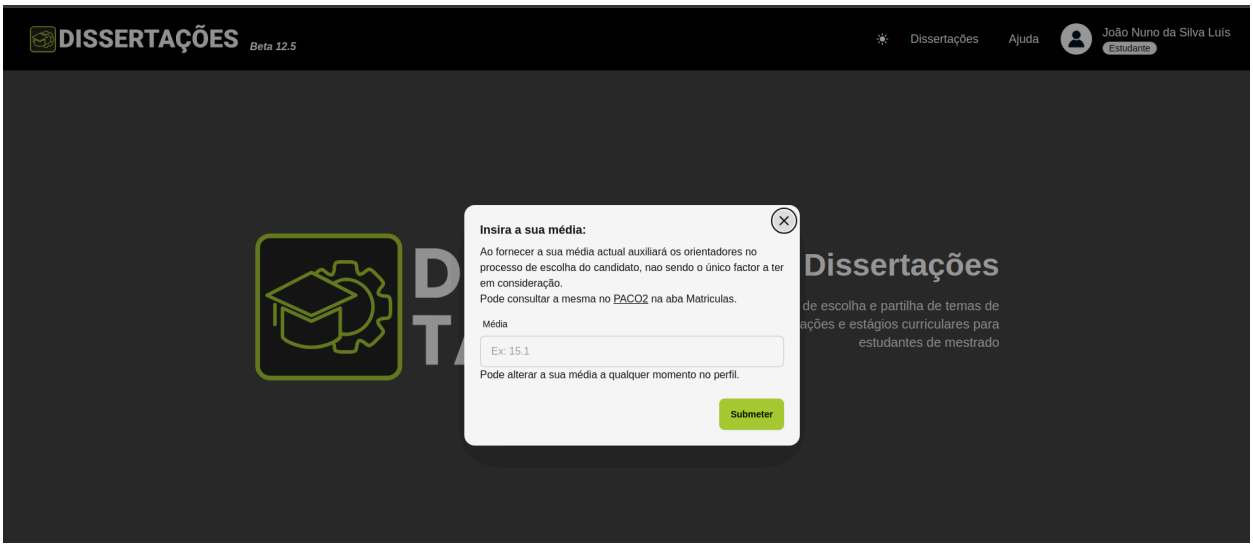
Em caso de problemas com a plataforma, poderá comunicar com o administrador do mesmo da seguinte forma:

- Email: dgomes@ua.pt ;

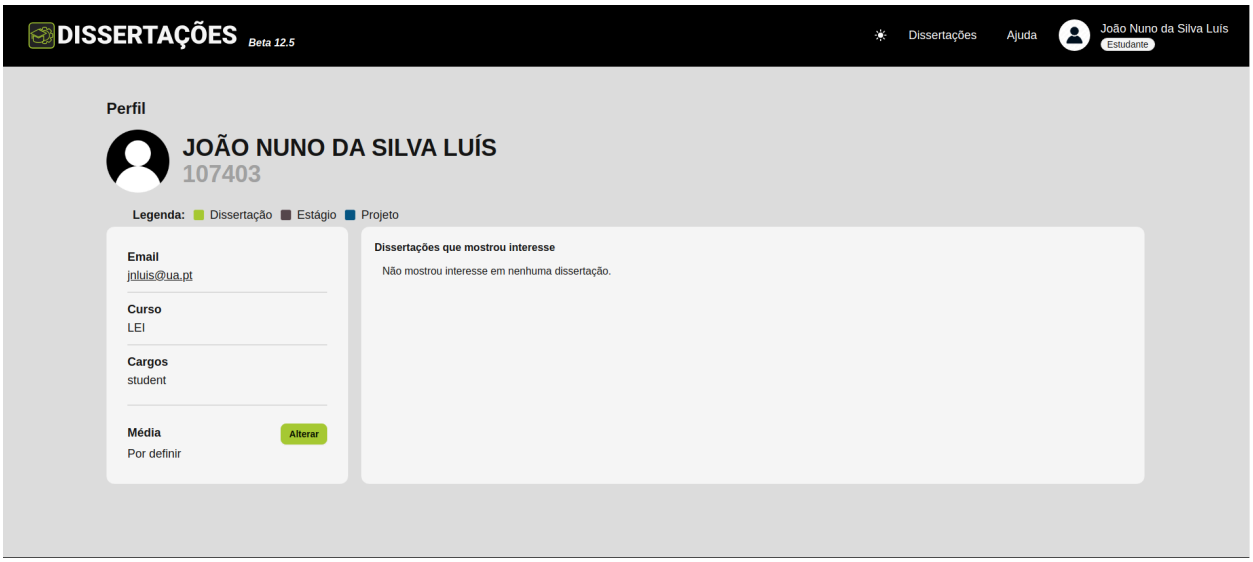
Alunos

Primeiro Login

Depois de efetuar login com o IdP.UA na plataforma, será pedido que insira a sua média. Este valor, conjuntamente com o número de ECTS aprovados, **pode vir** a ser uma métrica de avaliação usada pelos docentes orientadores de uma proposta. Assim, **não deve mentir** ou colocar um valor errado na sua média.



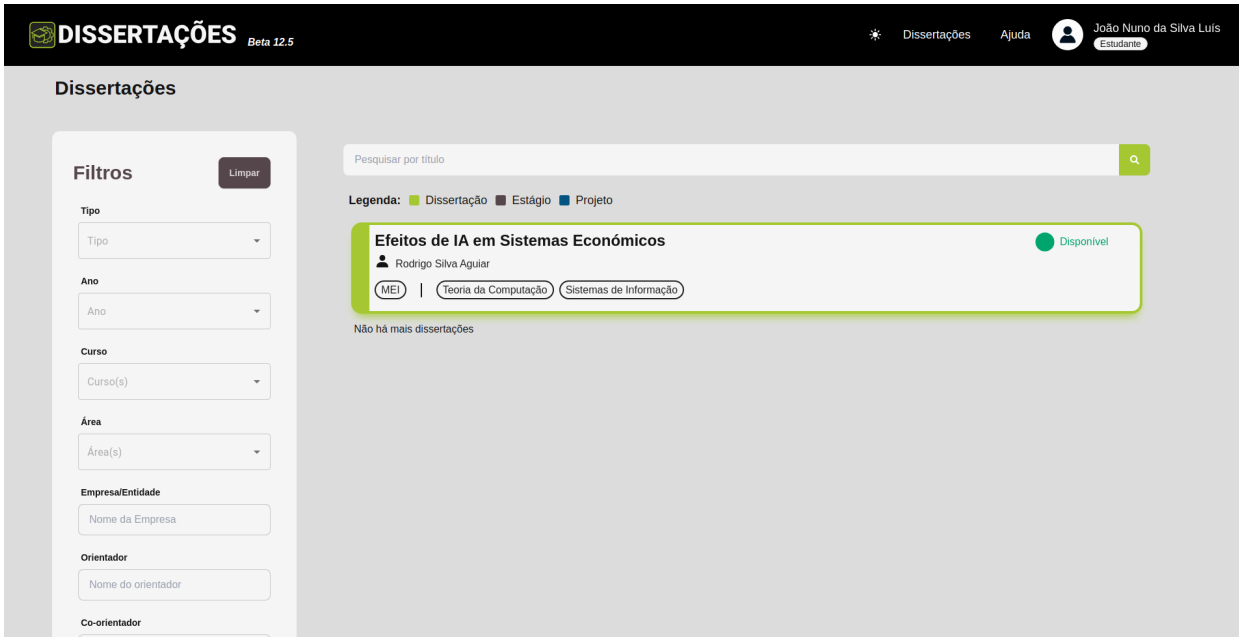
No entanto, se optar por inserir a sua média mais tarde, poderá fazê-lo na página do **"Perfil"**, clicando no canto superior direito e depois clicar no botão **"Alterar"**.



Para além disso, nesta página conseguirá ver mais algumas informações pessoais e ainda acompanhar o estado de todas as propostas na qual mostrar interesse.

Lista de Dissertações

Na aba **Dissertações ou no Logo da página**, poderá encontrar a página principal da aplicação. Nesta, encontrará todas as propostas registadas no sistema.



Os filtros, aplicados automaticamente, isto é, **sem necessidade** de clicar num botão de Aplicar, permitem filtrar as propostas por diferentes campos das mesmas.

Nesta página, são mostradas as **informações gerais** de cada proposta, ou seja, **título, orientador e co-orientador** (a proposta da imagem acima não possui um co-orientador), **curso(s)** a que se aplica e ainda **área(s) científicas aplicáveis.**

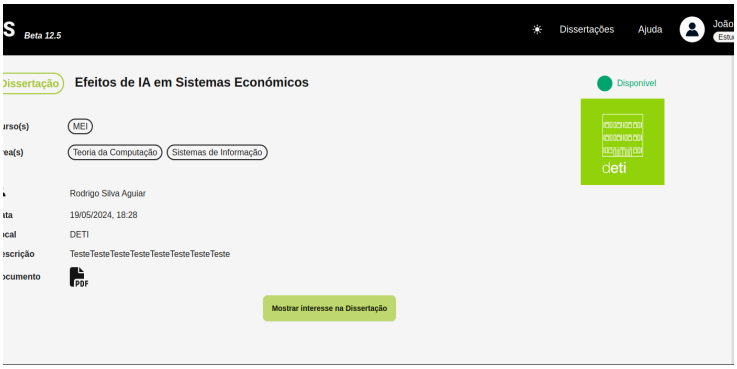
Nesta página, as propostas podem apresentar 2 estados: Aberta ou Fechada (ou seja, proposta já atribuída a um outro aluno).

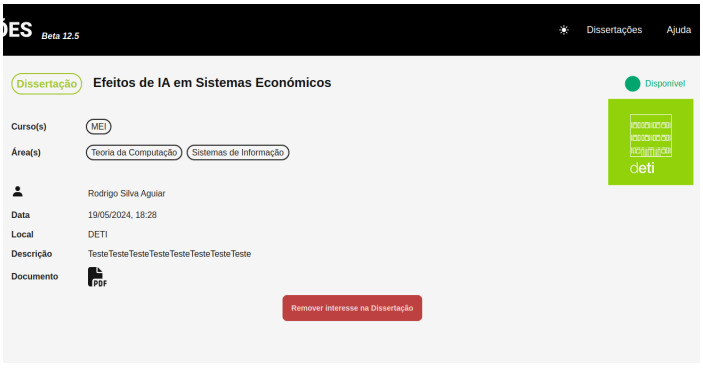
Por último, notar ainda que as propostas adicionadas à plataforma desde o último login terão um fundo branco (como a da imagem acima), enquanto as restantes, na teoria já vistas, terão um fundo mais acinzentado.

Interesse em Propostas

Carregando numa proposta para ver os seus detalhes, é possível mostrar/remover interesse na mesma.

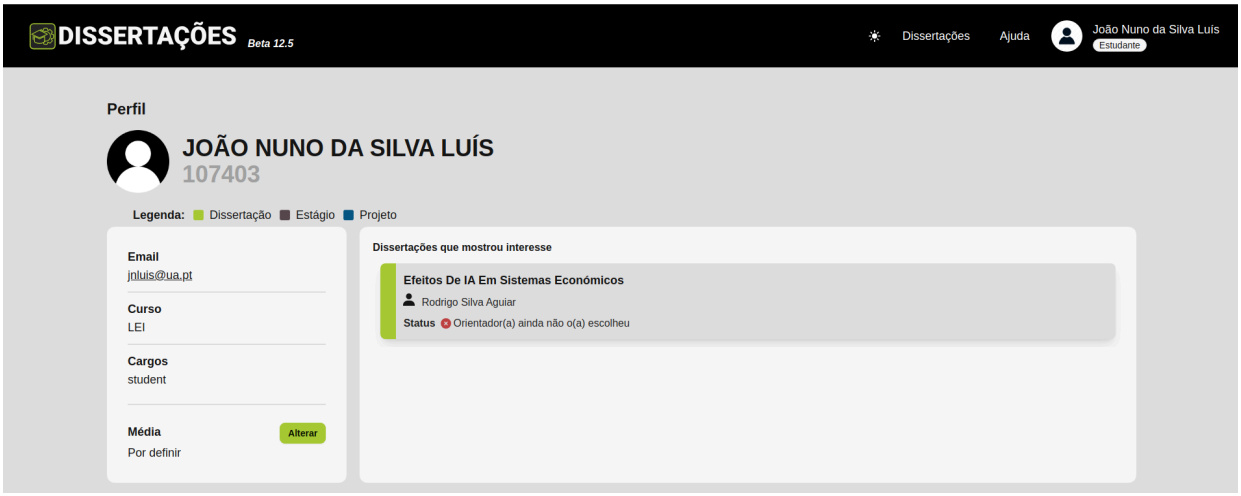
IMPORTANTE: pode mostrar interesse no número de propostas que quiser. Esta ação não é vinculativa!





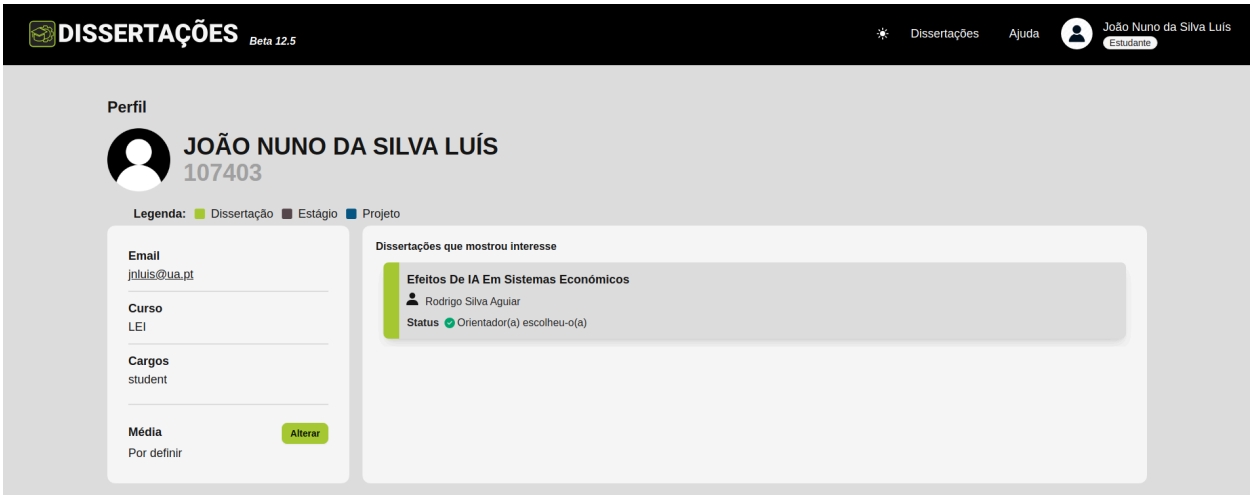
Depois de mostrar interesse numa proposta, poderá acompanhar o estado da mesma na página de perfil.

Neste momento, o professor foi notificado que há um aluno interessado naquela proposta. Escolher um aluno é uma ação que **poderá levar** algum **tempo**, pelo que se existir alguma evolução no processo, o aluno **será notificado** por **e-mail**.



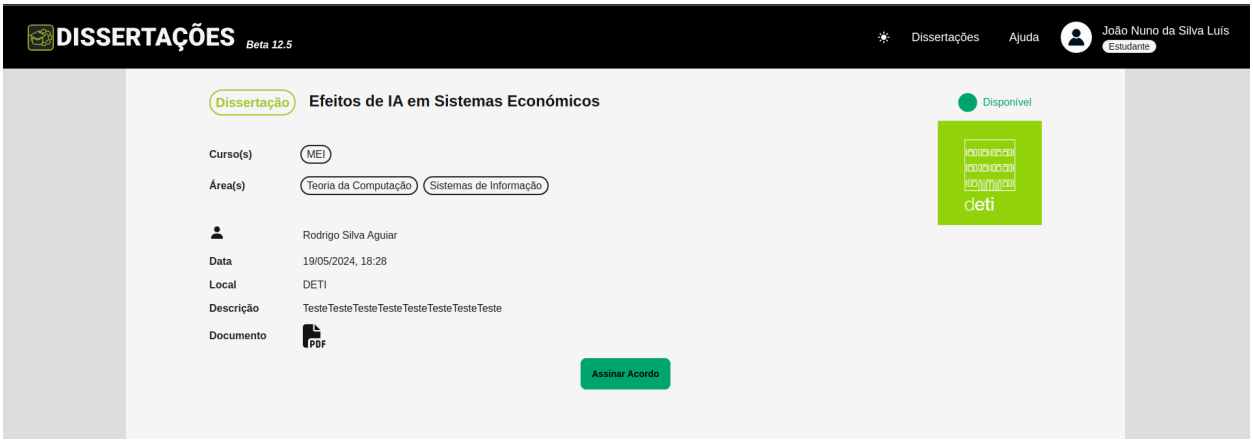
Assinar Acordo

Se o/a professor o/a escolher, a sua página de perfil será atualizada:



Já nos detalhes da proposta, terá agora de assinar o acordo para avançar para a fase seguinte.

IMPORTANTE: Esta ação não pode ser desfeita, pelo que deve apenas fazê-la se tiver a certeza que quer esta proposta.



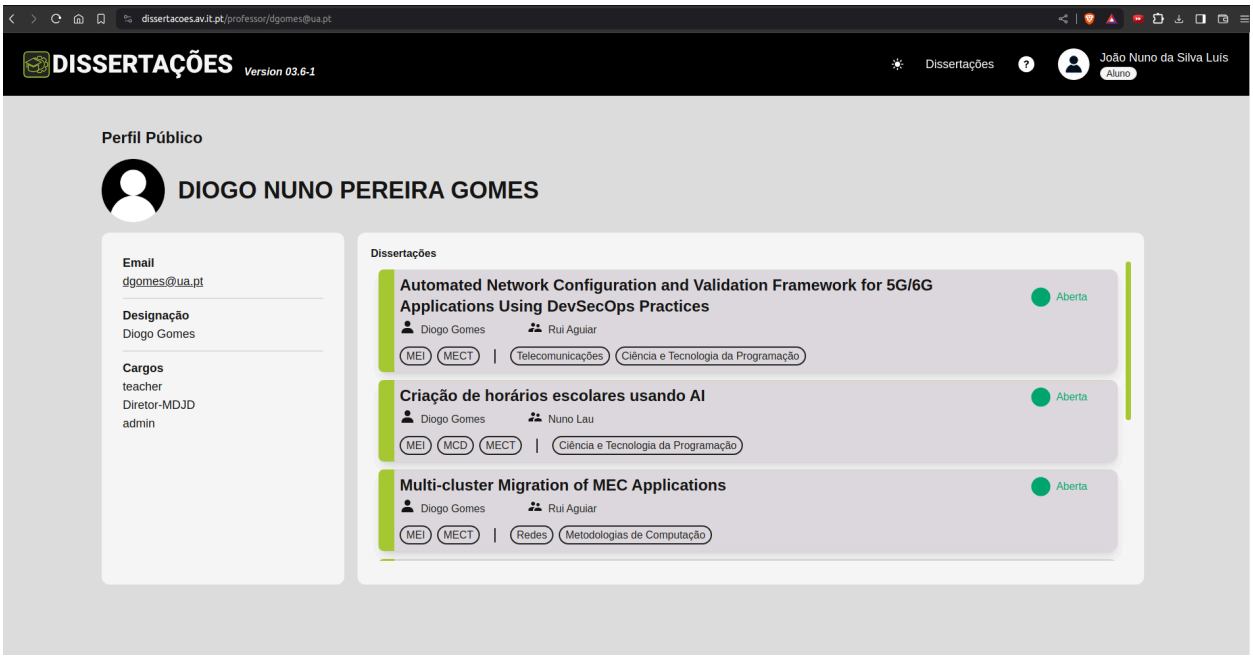
Esta ação atualizará a sua página de perfil novamente:

Resta agora a confirmação final por parte do professor orientador. Se a mesma for dada, receberá um **e-mail** com a atribuição do tema e sua página de perfil ficará da seguinte forma:

Nesta fase, ficará impossibilitado de mostrar interesse noutras propostas de dissertação/estágio para o ano letivo corrente.

Perfil Público

É possível procurar por todas as propostas que um docente tenha disponíveis na plataforma. Para isto, basta acrescentar /professor/[*e-mail do professor*] ao URL base, como se pode ver na figura abaixo. Esta página é particularmente **útil** para mostrar de forma rápida **todas** as propostas orientadas **ou** co-orientadas pelo docente.



APÊNDICE F

Apêndice F - Manual de Utilizador do Docente e Diretor de Curso

Universidade de Aveiro

DETI

Guião de apoio para a plataforma Dissertações

Introdução

Este documento serve de apoio à nova plataforma **Dissertações**, que se encontra disponível no link: https://dissertacoes.av.it.pt

Nesta plataforma, como docente, poderá submeter dissertações/estágios para os vários mestrados do DETI e gerir os estudantes interessados em cada uma das suas propostas até que cheguem a um acordo.

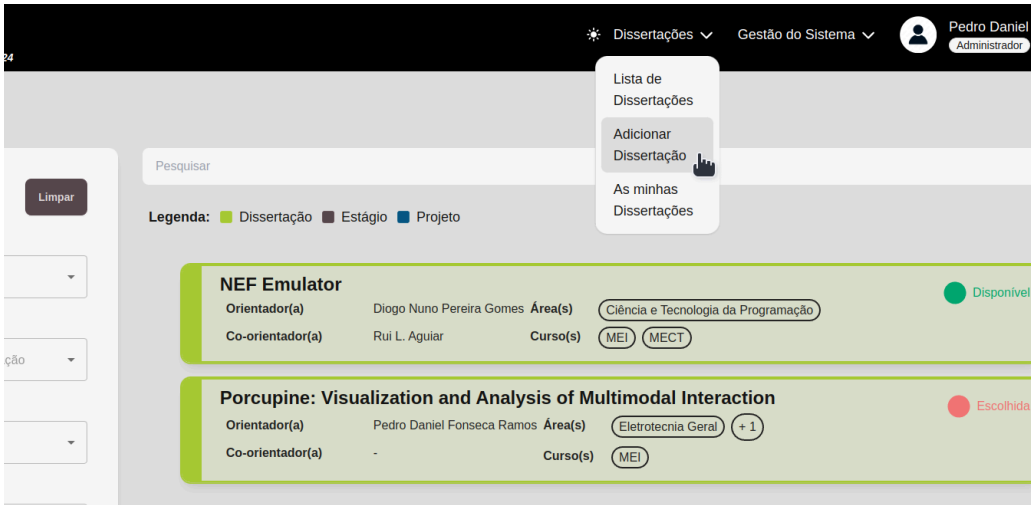
Em caso de problemas com a plataforma ou sugestões de como melhorar a mesma, poderá comunicar connosco pelas seguintes plataformas:

- Email: dgomes@ua.pt ;
- Issues no GitHub (acessível apenas a membros da organização do DETI): https://github.com/detiuaveiro/dsd/issues ;

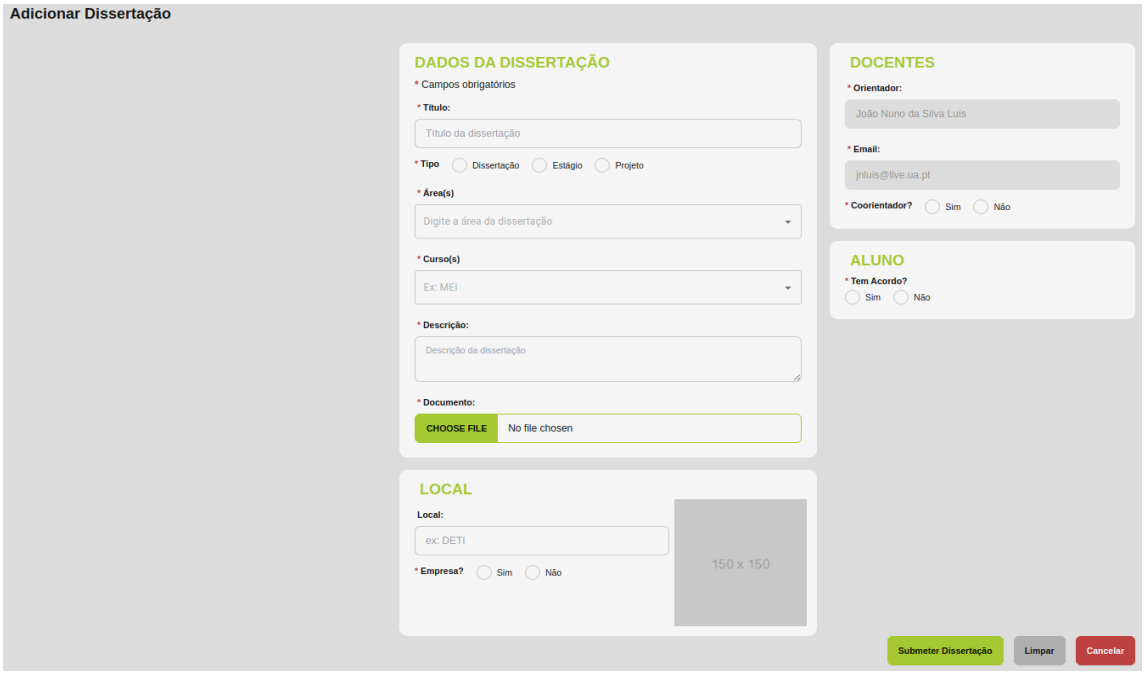
Docentes

Inserção de propostas

Depois de efetuar login com o IdP.UA na plataforma, deve dirigir-se à aba **Dissertações**, clicando na opção **Adicionar Dissertação**.



Nesta página irá encontrar um **formulário** com todos os atributos necessários para adicionar a sua dissertação à plataforma.



No campo título, o mesmo deve ter no **máximo 130 caracteres.**

O campo das **áreas** está de acordo com o mapa das Áreas Científicas da UA, devendo estas apenas serem escolhidas caso sejam **aplicáveis**.

O campo do curso será sujeito a **revisão** por parte dos **diretores de curso**, que **validarão** se a dissertação proposta se enquadra no mestrado por eles tutelado em fase posterior à submissão da dissertação.

Enquanto **pelo menos um diretor de curso** não **validar** o enquadramento de uma dissertação, a dissertação **não aparece** na listagem das **dissertações**. Caso sejam escolhidos vários cursos, a dissertação é **atualizada** à medida que os cursos escolhidos são validados pelos diretores de curso.

O campo da descrição deve conter, no mínimo, **10** caracteres, e no máximo **500** e deverá ser um pequeno *abstract* da dissertação para consulta rápida pelos candidatos.

O campo do documento deve ser um ficheiro em **formato PDF** válido com, no máximo, **50mb** de tamanho.

Na secção Local, é possível colocar uma **imagem** associada ao local/empresa em formato **.png ou .jpeg**, e com um tamanho **recomendado** de 150x150 pixels. Caso não seja colocada uma imagem, será utilizada uma *default* (imagem do DETI).

Caso seja escolhida a opção de **Empresa**, terá de ser fornecido o **nome da empresa** e, se aplicável, o **ficheiro do Memorando**, com um tamanho máximo de 50mb.

O memorando é **opcional** para **Dissertações** e **Projetos**, mas é **obrigatório** para **Estágios** que envolvam uma empresa.

Na secção de **docentes**, é possível indicar **coorientadores**.

Se este for do tipo interno, é **necessário** que o mesmo efetue login na plataforma **pelo menos uma vez**, antes de poder ser indicado.

Será fácil perceber se este já o fez ou não, pois o seu email **aparecerá como sugestão** de *autocomplete*.

Se o coorientador for **externo**, como não tem (ou pode não ter) conta institucional da UA, **não é necessário login** prévio.

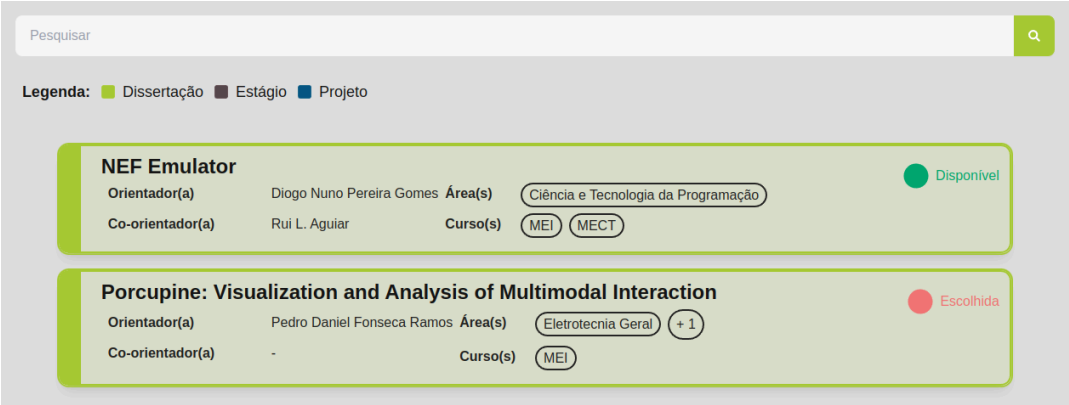
O campo de aluno pode ser preenchido caso a dissertação **já tenha um aluno atribuído**. Este campo **não precisa** que o aluno se registe primeiro na plataforma.

Esta opção **não bloqueará a dissertação** para os restantes alunos, pelo que qualquer outro aluno(a) pode registar interesse e um acordo ser feito com o mesmo(a) segundo o processo normal.

Caso seja atribuído um aluno, a dissertação t**erá na mesma que ser aprovada** por um administrador e por pelo menos um diretor de curso antes de o acordo final poder ser fechado.

Caso a dissertação seja registada com um aluno atribuído, ainda será necessário posteriormente a confirmação por ambas as partes (aluno 1º e orientador 2º).

Lista de Dissertações



Após a submissão de uma dissertação, esta **só será apresentada** aos restantes utilizadores após os **seguintes critérios** serem cumpridos:

- Um dos **administradores** da plataforma **aceita** a sua dissertação

- **Pelo menos um** Diretor de Curso escolhido validar que a dissertação é adequada ao seu curso;

Nas propostas, **apenas** serão mostrados os cursos que já se encontram **aceites** pelo respetivo diretor de curso.

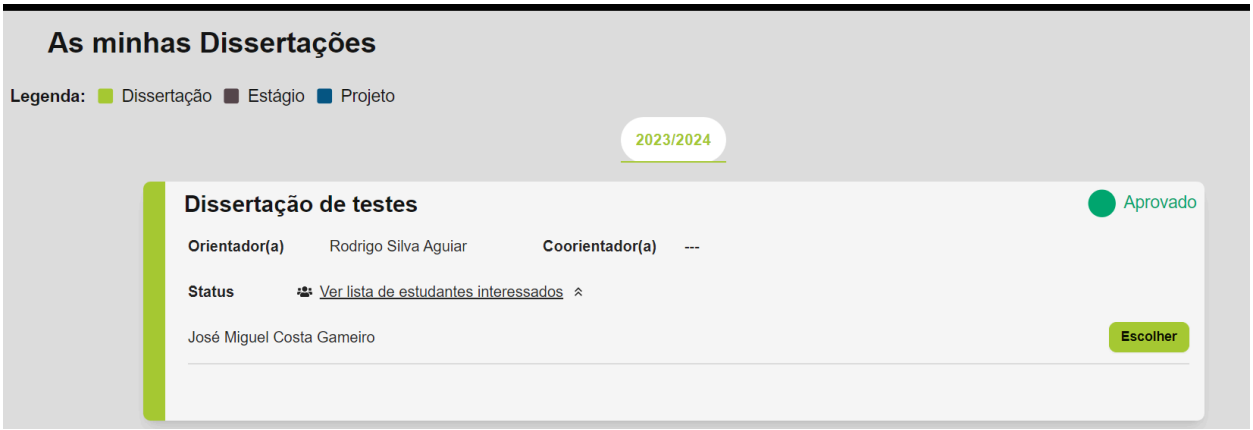
Sendo assim, por exemplo, será **possível** que uma dissertação submetida com MEI e MECT como cursos possa aparecer apenas com o curso MEI na lista caso o diretor do MECT **ainda não tenha validado** o seu curso nesta dissertação.

Ao **clicar em cima** do nome do orientador ou coorientador será **possível enviar** diretamente um **email** para o mesmo, sendo este um **padrão aplicado** em toda a plataforma quando são referidos tanto nomes de docentes como de alunos.

Interesse em Dissertações

Após a validação da dissertação por um administrador da plataforma e de pelo menos um diretor de um curso escolhido, qualquer estudante pode mostrar interesse na mesma, na aba **Dissertações -> Lista de Dissertações**.

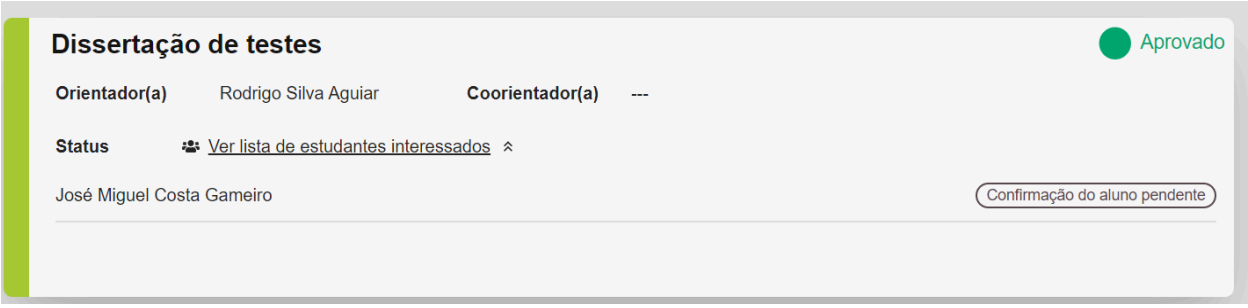
Ao estudante é possível mostrar interesse em dissertações, e quando essa ação ocorrer, essa informação está disponível na aba **Dissertações -> As minhas Dissertações**, perto da dissertação em questão.



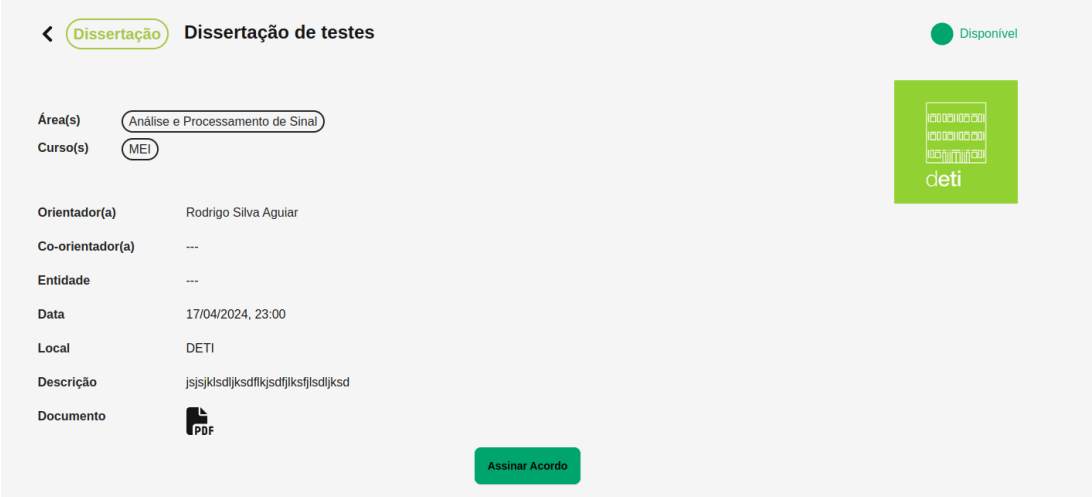
Nesta página, pode escolher quais dos alunos que mostraram interesse nesta dissertação deseja avançar para a próxima fase de escolha.

Após escolher um aluno, este será notificado e vai-lhe ser pedido que assine o acordo final.

Nesta fase, deverá ver o seguinte:

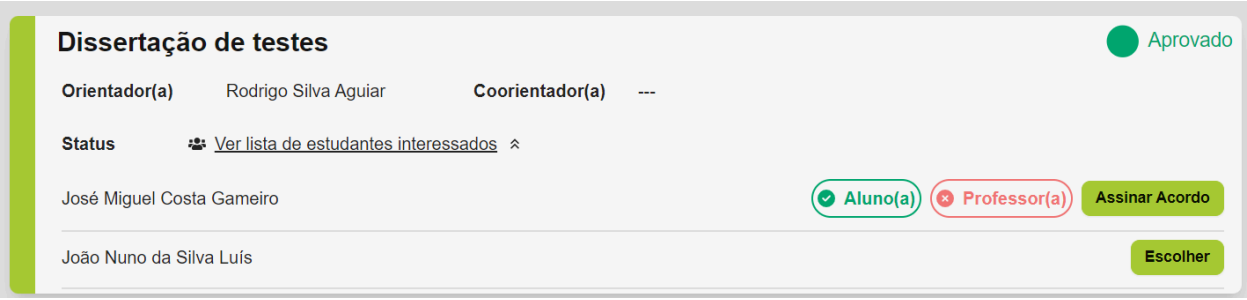


O aluno deverá ver o seguinte:

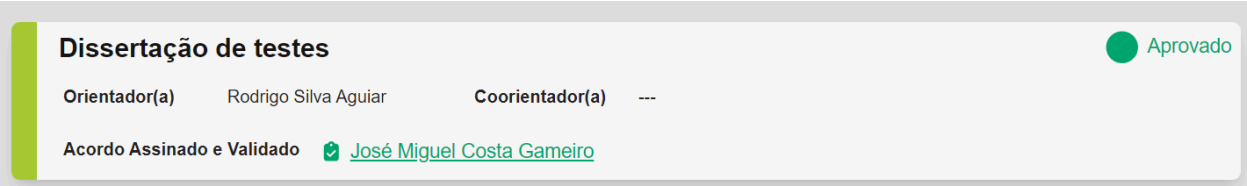


Após um destes alunos assinar, será possível aceitar o aluno final e assinar um acordo com o mesmo na página **As minhas Dissertações.**

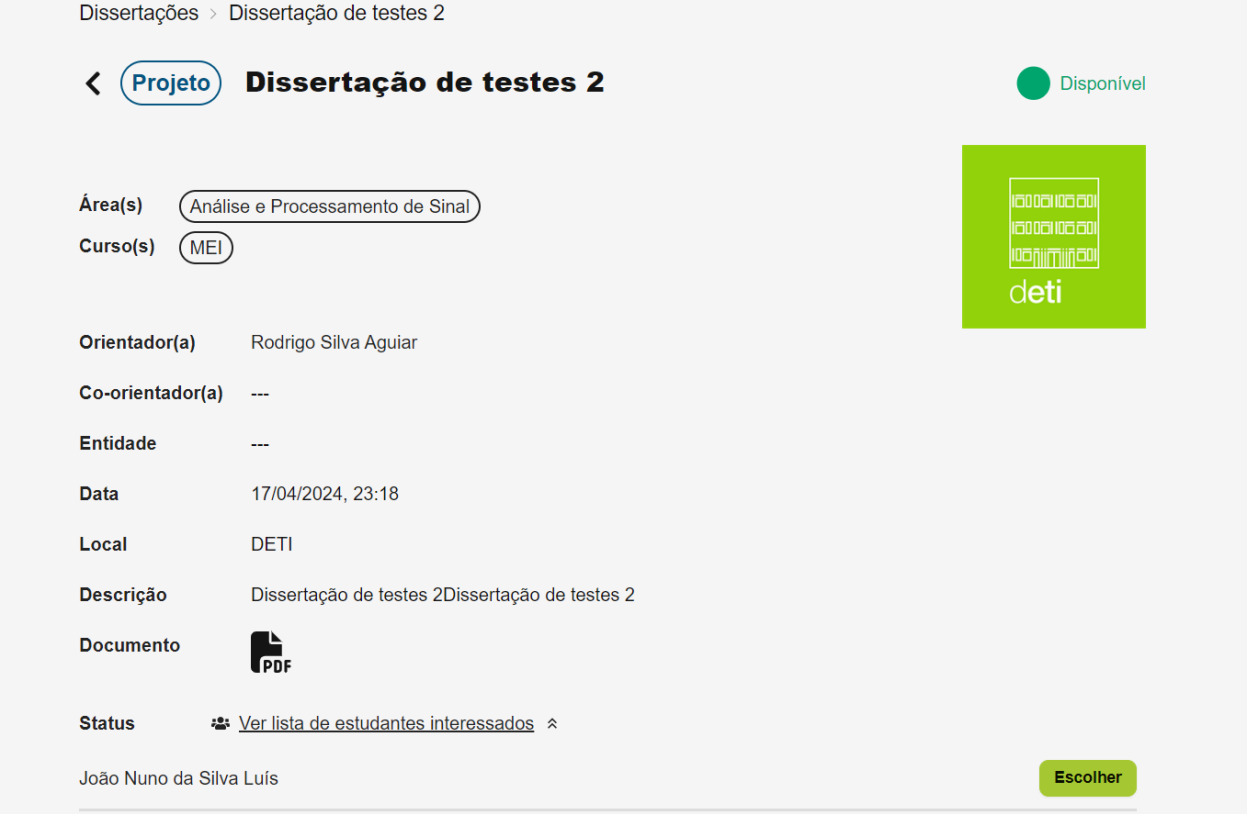
Esta ação atribuiu a dissertação ao aluno, sendo que a mesma passa para o Estado de Escolhida.



Ao clicar no botão de Assinar Acordo, esta dissertação será finalmente atribuída ao aluno, aparecendo o seguinte na interface.



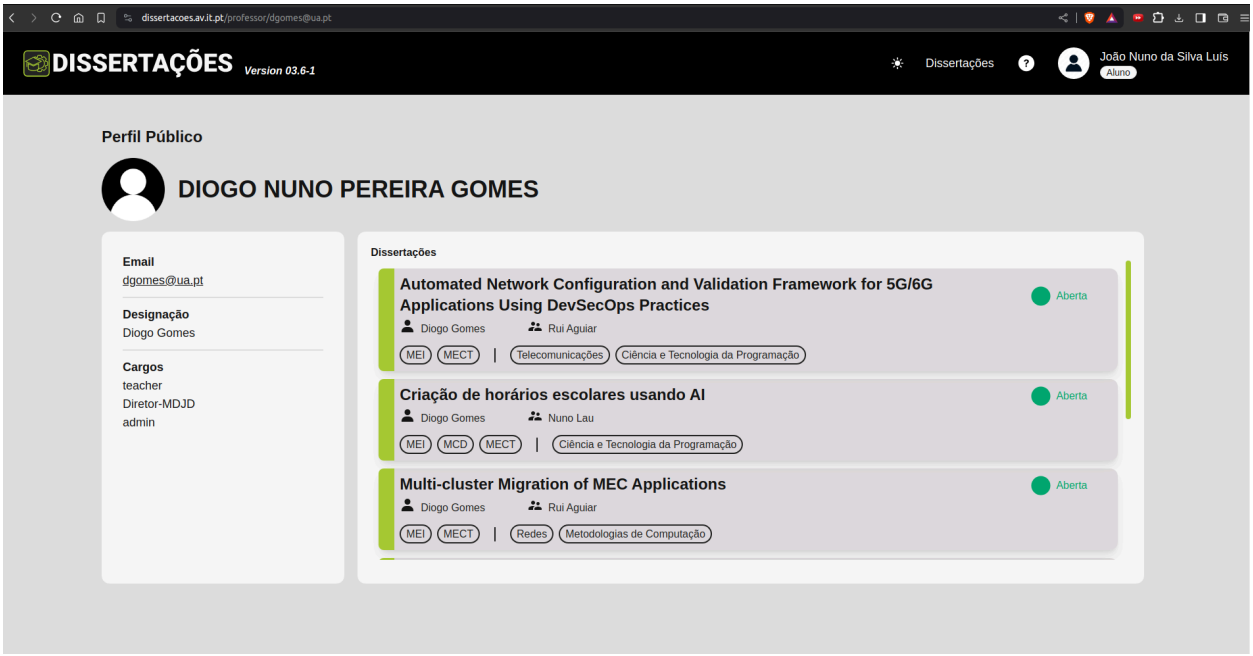
Em alternativa ao procedimento anterior, é também possível acompanhar os interessados de uma determinada dissertação nos detalhes da mesma.



Em ambos os procedimentos, é possível contactar qualquer dos alunos interessados via e-mail e para isso basta clicar no seu nome, que ativará com *popup* onde será levado para a sua aplicação de e-mails.

Perfil Público

É possível mostrar todas as propostas que um docente tenha disponíveis na plataforma. Para isto, basta acrescentar /professor/[*e-mail do professor*] ao URL base, como se pode ver na figura abaixo. Esta página é particularmente **útil** para mostrar de forma rápida **todas** as propostas orientadas **ou** co-orientadas pelo docente.



Diretores de Curso

Validação de dissertações

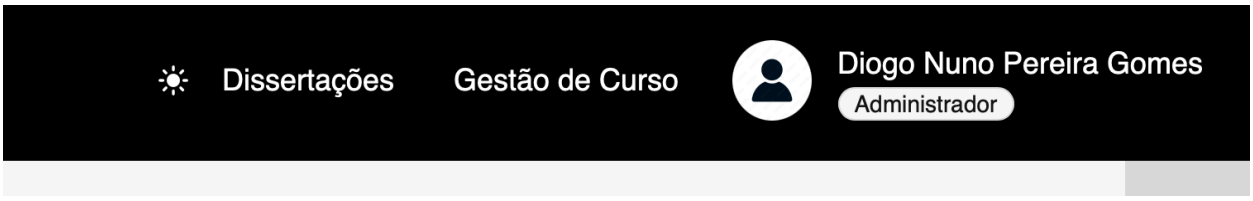
Ao contrário do website antigo, no novo dsd existe o **cargo** de **diretor de curso** com funções na aplicação.

Diretores de curso são responsáveis por **validar** que uma dissertação é adequada ao **seu curso**.

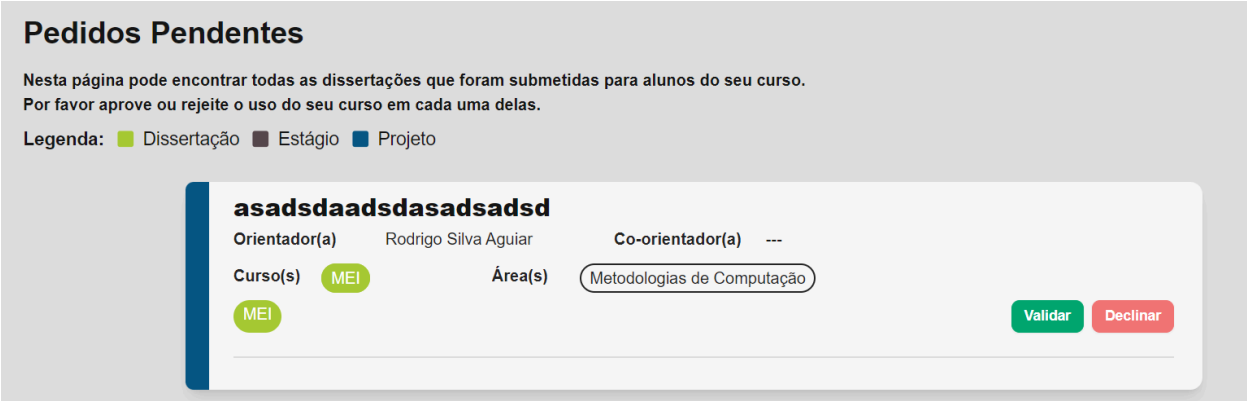
A atribuição deste cargo é realizada pelos **administradores da plataforma**, que atualizam quem é o diretor de um determinado curso.

Utilizadores que são simultaneamente diretores de um curso, terão na **barra de navegação** o respectivo **curso do qual são diretores**.

Aparecerá também na barra de navegação a opção **"Gestão de Curso"**.



Através deste dropdown, se selecionar a opção "Gerir Curso", será possível um diretor de curso **validar ou declinar** a dissertação tendo em conta a adequação da mesma ao seu **curso**.



Últimas ações

Dado que no pico de utilização da plataforma podem ocorrer um número elevado de submissões de propostas, abrindo o dropdown "**Gestão de Curso**" e selecionando "**Últimas Ações**", é possível acompanhar o estado de **todas** as dissertações/estágios que tenham referenciado o curso do qual é diretor. Se clicar no botão "Página da dissertação", irá ser levado para os detalhes da mesma.