A picture containing text

Description automatically generated



INSTITUTO POLITÉCNICO DE COIMBRA

INSTITUTO SUPERIOR

DE ENGENHARIA

DE COIMBRA

DEPARTAMENTO DE INFORMÁTICA E SISTEMAS

**Sistema de Informação para Gestão de Parcerias de Estágio e Projetos do ISEC**

Projeto Final de Licenciatura em Engenharia Informática

Autor

**Bruno Gabriel Tavares Pinto**

Orientador

**César Paulo das Dores Páris**

Coimbra, julho e 2025

Índice

[Índice i](#_Toc207059099)

[Índice de figuras iii](#_Toc207059100)

[Lista de siglas e acrónimos iv](#_Toc207059101)

[Agradecimentos 1](#_Toc207059102)

[Resumo 2](#_Toc207059103)

[1 Introdução 3](#_Toc207059104)

[1.1 Estrutura do relatório 3](#_Toc207059105)

[2 Estado pré existente do projeto 5](#_Toc207059106)

[2.1 Arquitetura e tecnologias 5](#_Toc207059107)

[3 Conceção do Sistema 6](#_Toc207059108)

[3.1 Objetivos 6](#_Toc207059109)

[3.2 Arquitetura e tecnologias 7](#_Toc207059110)

[3.2.1 Arquitetura do sitema 7](#_Toc207059111)

[3.2.2 Tecnologias usadas 8](#_Toc207059112)

[4 Implementação da API 10](#_Toc207059113)

[4.1 Modelo de Dados 10](#_Toc207059114)

[4.2 Implementação da API 10](#_Toc207059115)

[4.2.1 Organização do projeto 11](#_Toc207059116)

[4.2.2 Funcionamento da API 11](#_Toc207059117)

[5 Interface e design visual 13](#_Toc207059118)

[5.1 Conceção visual do sistema 13](#_Toc207059119)

[5.1.1 Cores 13](#_Toc207059120)

[5.1.2 Tipografia 13](#_Toc207059121)

[5.1.3 Espaçamentos 14](#_Toc207059122)

[5.1.4 Componentes 14](#_Toc207059123)

[5.2 *Design* da Aplicação 14](#_Toc207059124)

[5.3 Implementação da Interface 15](#_Toc207059125)

[5.3.1 Organização do projeto 15](#_Toc207059126)

[5.3.2 Arquitetura da Interface 16](#_Toc207059127)

[6 Casos de uso 18](#_Toc207059128)

[6.1 Autenticação de um utilizador (*login*) 18](#_Toc207059129)

[6.2 Filtragem de Listas 19](#_Toc207059130)

[6.3 Gestão de Docentes 20](#_Toc207059131)

[6.4 Gestão de Calendários 23](#_Toc207059132)

[6.5 Submissão de uma Proposta 23](#_Toc207059133)

[7 Desenvolvimento Futuro 24](#_Toc207059134)

[8 Conclusão 26](#_Toc207059135)

[Apêndices I](#_Toc207059136)

Índice de figuras

[Figura 1 - Arquitetura do Sistema 8](#_Toc207058143)

[Figura 2 - Diagrama Entidade-Relação 10](#_Toc207058144)

[Figura 3 – Estrutura padrão dos endpoints da API 12](#_Toc207058145)

[Figura 4 - Representação de uma cor com as suas nove variações 13](#_Toc207058146)

[Figura 5 - Página de *Login* 18](#_Toc207058147)

[Figura 6 - Página inicial com as permissões padrão 19](#_Toc207058148)

[Figura 7 - Página inicial com todas as permissões 19](#_Toc207058149)

[Figura 8 - filtragem de listas 20](#_Toc207058150)

[Figura 9 - Listagem de docentes 20](#_Toc207058151)

[Figura 10 - Visualização de um docente 21](#_Toc207058152)

[Figura 11 - Edição de um docente 22](#_Toc207058153)

[Figura 12 - Listagem de com um docente removido 22](#_Toc207058154)

[Figura 13 - Estado das fucnionalidades 24](#_Toc207058155)

Lista de siglas e acrónimos

|  |  |
| --- | --- |
| API | *Application Programming Interface* |
| HTML | *HyperText Markup Language* |
| HTTP | *Hypertext Transfer Protocol* |
| IDE | *Integrated Development Environmen* |
| ISEC | Instituto Superior de Engenharia de Coimbra |
| JSON | *JavaScript Object Notation* |
| JWT | JSON *Web Token* |
| MVT | *Model-View-Template* |
| ORM | *Object Relational Mapping* |
| SQL | *Structured Query Language* |
| UI | *User Interface* |
| VSCode | Visual Studio Code |

Agradecimentos

Agradeço ao professor César Páris, orientador do projeto, e ao professor João Costa, responsável pela unidade curricular, pela orientação, disponibilidade e apoio prestados ao longo do semestre, que foram essenciais para a concretização deste trabalho.

Resumo

Palavras Chave: Estágios, …

# Introdução

Este relatório descreve o trabalho desenvolvido no âmbito da Licenciatura em Engenharia Informática no ISEC (Instituto Superior de Engenharia de Coimbra). O projeto realizado intitula-se "Desenvolvimento de um Sistema de Informação para Gestão de Parcerias de Estágio e Projetos do ISEC".

O projeto teve como objetivo desenvolver uma plataforma que centralizasse toda a informação relativa às colaborações entre o ISEC e empresas externas, no âmbito de estágios e projetos de investigação. Através desta solução, pretendeu-se facilitar a gestão, comunicação e acompanhamento destas parcerias, promovendo maior eficiência administrativa e transparência nas interações entre os vários intervenientes: alunos, docentes e entidades parceiras.

Este projeto surge da necessidade de colmatar uma lacuna no sistema do ISEC, onde não existe atualmente uma ferramenta centralizada para gerir e monitorizar as colaborações com empresas no âmbito de estágios e projetos de investigação. Com este sistema pretende-se organizar e documentar essas parcerias, facilitando a comunicação, o acompanhamento e a avaliação das oportunidades, promovendo uma gestão mais eficiente e transparente entre alunos, docentes e entidades parceiras.

O desenvolvimento do projeto foi realizado ao longo do semestre com reuniões quinzenais, que serviram para apresentar o trabalho desenvolvido e receber *feedback* dos docentes envolvidos. Estas reuniões permitiram assegurar que o desenvolvimento seguia na direção correta, possibilitando corrigir erros e melhorar certos aspetos da aplicação.

O projeto contou com a orientação do professor César Páris e com o acompanhamento do professor João Costa, que atuou como cliente do sistema.

## Estrutura do relatório

Este relatório está organizado em seis capítulos, que abordam as diferentes fases e componentes do projeto:

* **Estado pré existente do projeto:** apresenta a análise do sistema antes do desenvolvimento, identificando funcionalidades implementadas, limitações e problemas encontrados.
* **Conceção do Sistema:** apresenta o planeamento e as decisões de conceção do sistema, incluindo a definição dos objetivos do projeto, a arquitetura adotada, as tecnologias utilizadas e os critérios que orientaram o desenvolvimento da aplicação.
* **Implementação da API:** detalha o processo de conceção e desenvolvimento da API (*Application Programming Interface*), explicando a modelação dos dados, a estrutura dos *endpoints*, os métodos utilizados e as questões relacionadas com segurança e autenticação.
* **Interface e design visual:** apresenta as decisões tomadas no planeamento e conceção da interface da aplicação, abordando aspetos de usabilidade, acessibilidade, organização visual e a implementação prática do design no sistema.
* **Casos de uso:** apresenta alguns fluxos do sistema, ilustrando o seu funcionamento através de cenários de interação complementados com a visualização dos ecrãs correspondentes.
* **Desenvolvimento Futuro:** detalha as funcionalidades pendentes, potencias melhorias e o planeamento de evoluções futuras do sistema, destacando oportunidades para expansão e otimização.

# Estado pré existente do projeto

No início do projeto, a aplicação encontrava-se parcialmente desenvolvida, pelo que foi necessário realizar testes práticos para compreender o seu funcionamento e identificar as funcionalidades implementadas. A partir desta análise, foi possível identificar as seguintes funcionalidades:

* **Administrador:**
  + Gerir Empresas, Cursos, Calendários, Alunos e Propostas.
  + Importar uma lista de alunos com recurso a um ficheiro Excel.
* **Empresa:**
  + Criação de propostas.
  + Consulta do estado das propostas.
* **Aluno:** 
  + Edição do seu perfil, incluindo currículo, médias e outros dados pessoais e curriculares.
  + Consulta das propostas disponíveis.
  + Submissão de uma candidatura e acompanhamento do estado da mesma**.**

Durante a avaliação, foram identificadas algumas limitações, nomeadamente: a necessidade de atualizar manualmente a página para visualizar novas propostas, a impossibilidade por parte das Empresas editarem as suas próprias propostas, restrições de acesso do administrador a candidaturas e propostas aprovadas, e validação insuficiente de alguns campos.

Posteriormente, foi realizada uma análise da arquitetura e da estrutura do código, que revelou uma organização pouco estruturada, baixa modularidade e falta de documentação. Estes fatores dificultavam a manutenção e evolução da aplicação, tornando necessário reformular o sistema para acomodar de forma adequada as funcionalidades existentes e os novos objetivos definidos.

## Arquitetura e tecnologias

O sistema encontrava-se dividido em duas aplicações: o *backend*, desenvolvido em Django [1], que funcionava como uma API responsável pela lógica de negócio, gestão de dados e comunicação com a base de dados; e o *frontend*, desenvolvido em React [2], que constituía a aplicação cliente e era responsável pela interface e interação com o utilizador.

Apesar da separação entre frontend e backend, a análise da estrutura do código revelou baixa modularidade e falta de documentação, o que dificultava a manutenção e evolução do sistema. A organização das pastas e componentes não seguia padrões consistentes, tornando complexa a compreensão do fluxo da aplicação e a implementação de alterações ou novas funcionalidades.

# Conceção do Sistema

Após a análise do sistema pré-existente, verificou-se que, apesar de algumas funcionalidades estarem implementadas, a arquitetura apresentava limitações em termos de modularidade, organização do código e documentação, o que dificultava a manutenção e a expansão do sistema.

Face a este cenário, tornou-se necessário refazer o sistema do zero, aproveitando apenas os componentes e estruturas que se mostrassem estáveis e reutilizáveis. Esta abordagem permitiu criar uma base mais organizada, modular e documentada, mantendo a ideia geral da arquitetura original, que já respeitava boas práticas de separação de responsabilidades entre cliente e servidor. Além disso, algumas das tecnologias utilizadas foram mantidas, por serem soluções consolidadas, escaláveis e compatíveis com futuras evoluções. Desta forma, as novas funcionalidades puderam ser implementadas de forma eficiente e sustentável, garantindo que o sistema se mantivesse escalável e fácil de manter a longo prazo.

## Objetivos

Com base na análise do sistema existente e em reuniões realizadas com os docentes envolvidos no projeto, foram definidos os objetivos que orientaram o desenvolvimento da nova versão da aplicação. Estes requisitos visam responder às necessidades identificadas, com foco na criação de uma arquitetura escalável, modular e bem organizada, de forma a facilitar a evolução e manutenção do sistema a longo prazo.

A formalização completa dos requisitos e funcionalidades, incluindo a respetiva categorização de prioridade segundo a notação MoSCoW [3], encontra-se detalhada no Anexo X.

Entre as funcionalidades planeadas destacam-se:

* **Exportação de atribuições:** Gerar ficheiro Excel com a listagem final de alunos, empresas e estágios/projetos atribuídos.
* **Criação automática do documento da proposta:** Preenchimento automático do PDF da proposta, utilizando um *template* predefinido.
* **Preenchimento automático de protocolos:** Gerar o protocolo com dados preenchidos automaticamente, prontos para assinatura pelas partes envolvidas.
* **Sistema de comunicação centralizado:** Implementação de um espaço único no sistema para troca de mensagens entre alunos, empresas, docentes e administradores, centralizando a comunicação e tornando a gestão mais prática, organizada e eficiente.
* **Automatização das atribuições:** Desenvolvimento de um algoritmo que realiza automaticamente a distribuição dos alunos pelas propostas, garantindo maior rapidez no processo.
* **Redesign da interface:** Reformulação completa da UI para acomodar a nova estrutura do sistema e garantir uma melhor acessibilidade, responsividade e usabilidade.
* **Notificação de atribuições às empresas:** Envio automático de informação sobre as atribuições por email ou através da aplicação.

Os requisitos apresentados implicam, direta ou indiretamente, uma reestruturação da arquitetura atual do sistema – seja pela modificação de funcionalidades existentes, seja pela introdução de novos papéis funcionais. Assim, será necessário evoluir para um modelo com quatro tipos de utilizadores:

* Administrador
* Docente
* Representante
* Aluno

Com esta expansão e a consequente adição de novas páginas e funcionalidades, a complexidade do sistema aumenta. Para garantir que a aplicação se mantenha eficaz e de fácil utilização, é essencial reorganizar visualmente a informação e implementar um redesign da interface. Esta reformulação permitirá preservar uma experiência limpa, intuitiva e eficiente, mesmo perante o crescimento das funcionalidades e das interações no sistema.

## Arquitetura e tecnologias

### Arquitetura do sitema

A aplicação segue uma arquitetura de cliente-servidor, onde a interface (cliente) comunica com uma API que funciona como camada intermediária entre a interface do utilizador e a base de dados (servidor). Toda a lógica de processamento de dados é tratada no lado do servidor, enquanto o cliente é responsável por apresentar os dados e recolher os *inputs* do utilizador. Esta separação permite uma maior escalabilidade, organização dos diferentes componentes do sistema.

Uma imagem com texto, captura de ecrã, diagrama, Tipo de letra

Os conteúdos gerados por IA podem estar incorretos.

Figura 1 - Arquitetura do Sistema

O sistema foi implementado utilizando Django para gestão de dados e lógica da aplicação (*backend* ou servidor) e React para a interface com o utilizador (*frontend* ou cliente).

Django é uma *framework* em Python que segue o modelo MVT (*Model-View-Template*). O *Model* é responsável pela representação dos dados e pela interação com a base de dados. O *View* gere o fluxo de dados e lida com os pedidos do utilizador. Já o *Template* descreve a apresentação dos dados.

No contexto deste sistema, Django foi utilizado para construir a API, através da qual o *frontend* pode aceder aos dados e funcionalidades da aplicação. Esta API expõe os dados em formato JSON (*JavaScript Object Notation*) e permite operações como consulta, criação, atualização e eliminação de dados, garantindo uma separação clara entre a lógica de dados e a interface. As interações entre o React e Django são feitas através de chamadas HTTP (*Hypertext Transfer Protocol*).

React é a ferramenta principal para a construção da interface do utilizador. Esta *framework*, em JavaScript, permite criar componentes dinâmicos e reutilizáveis que interagem com fontes externas – neste caso, a API criada em Django. Desta forma, o conteúdo da página pode ser atualizado em tempo real, sem que seja necessário recarregar a página por completo. De maneira a facilitar a construção da interface, e permitindo também a sua responsividade, foi usado Bootstrap [4], uma *framework* de código aberto desenvolvida com o objetivo de facilitar a organização de conteúdo usando classes e estruturas definidas de modo a acelerar o processo de desenvolvimento.

### Tecnologias usadas

Durante o desenvolvimento do sistema, foram utilizadas algumas ferramentas que apoiaram a implementação e organização da aplicação. Durante esta fase foram usados dois IDEs distintos, cada um escolhido com base nas necessidades especificas do *frontend* e do *backend*.

As tecnologias e ferramentas utilizadas incluem:

* **Visual Studio Code (VSCode)** [5]: IDE usado para o desenvolvimento do *frontend*. Foi escolhido por ser leve e altamente extensível, com diversas extensões de comunidade que melhoram a produtividade no desenvolvimento [6]. Embora não seja ideal para *debug[[1]](#footnote-1)*, não representa uma limitação significativa, no caso do *frontend*, pois o *debug* é realizado diretamente no *browser*.
* **PyCharm** [7]: IDE usado para o desenvolvimento da API em Django. Oferece suporte avançado a Python, *debug* gráfico, testes unitários integrados, gestão de dependências e integração com sistemas de controlo de versões [8].
* **GitHub** [9]: Plataforma usada para controlo de versões onde foi possível manter todo o código desenvolvido guardado de uma maneira segura.
* **Figma** [10]: Ferramenta utilizada para planear e visualizar o *design* da interface gráfica. Serviu de apoio ao desenvolvimento do *frontend*, permitindo definir layouts e testar a organização visual antes da implementação.

# Implementação da API

Uma vez definidos os objetivos do sistema, foi necessário reestruturar o modelo de dados e refazer a API de forma a acomodar as mudanças introduzidas. A estrutura foi pensada de maneira a respeitar as regras de negócio identificadas. A API foi desenvolvida com o objetivo de ser modular, facilitando a organização do código, tornando o de fácil manutenção e permitindo futuras adaptações.

## Modelo de Dados

Para responder às necessidades da aplicação, procedeu-se à reconstrução do modelo de dados. O modelo de dados foi pensado para representar todas as entidades e informação relevantes, incluindo utilizadores e outros dados essenciais. A nova estrutura está ilustrada no diagrama a seguir:

Uma imagem com texto, file, diagrama, captura de ecrã

Os conteúdos gerados por IA podem estar incorretos.

Figura 2 - Diagrama Entidade-Relação

## Implementação da API

A API foi desenvolvida em Django, uma framework em Python que oferece uma abstração para a interação com a base de dados através de ORM (*Object Relational Mapping*), isto é, as operações sobre os dados podem ser feitas diretamente no código, sem a necessidade de escrever *queries* em SQL manualmente.

A arquitetura da API segue o padrão *RESTful* [11], permitindo a comunicação entre cliente e servidor através de métodos HTTP como GET, POST, PUT e DELETE. Toda a troca de informação é feita em JSON, serializando os dados de maneira a garantir compatibilidade e simplicidade na comunicação.

Para assegurar a segurança e o controlo de acesso, cada pedido deve ser acompanhado de um JWT (JSON *Web Token*), que é obtido no momento de *login*. Este *token* identifica o utilizador perante o sistema, que, por sua vez, verifica se o utilizador tem permissões para aceder ao recurso solicitado. A ausência do *token*, ou envio de um *token* inválido, resulta na rejeição do pedido.

### Organização do projeto

A estrutura da API segue a organização típica de uma aplicação em Django, com responsabilidades bem definidas em ficheiros distintos:

* ***controllers***–pasta que contém subpastas com os ficheiros *views.py* e *urls.py* organizados por áreas funcionais, dividindo a lógica da aplicação em partes distintas e promovendo uma melhor modularidade.
* ***tasks*** – pasta onde se encontram todas as tarefas assíncronas do sistema. Cada tarefa está organizada em ficheiros separados, de forma a facilitar a manutenção e modularidade. Existe ainda um ficheiro *base.py*, responsável por centralizar as chamadas dessas tarefas
* ***templates*** – pasta que contém os *templates* de documentos oficiais utilizados pelo sistema. Inclui, por exemplo, ficheiros Word para propostas e ficheiros Excel para listagens exportadas. Estes *templates* permitem gerar documentos automaticamente, preenchendo-os com os dados de forma consistente e padronizada.
* ***models.py*** – ficheiro onde é definida a estrutura de dados da aplicação, como os Cursos, Propostas ou tipos de utilizador, representando a estrutura da base de dados.
* ***views.py*** –ficheiroonde estão todas as funções de manipulação de dados, tais como listar, visualizar, editar, eliminar e outras operações necessárias. Estas funções são diretamente chamadas quando uma chamada é feita à API.
* ***urls.py***–ficheiro onde são definidos todos os caminhos (*endpoints*) para cada funcionalidade da API.

### Funcionamento da API

O funcionamento da API assenta numa lógica onde cada recurso (como alunos, propostas ou cursos) tem um conjunto de endpoints associados que seguem uma estrutura padronizada. Esta organização facilita tanto a manutenção do código como facilita a utilização da API.

Cada controlador está associado a um modelo da aplicação e define os endpoints responsáveis por um conjunto de operações. Os caminhos definidos para cada operação seguem o seguinte padrão:

Figura 3 – Estrutura padrão dos endpoints da API

|  |  |  |
| --- | --- | --- |
| Método | Caminho | Descrição |
| GET | api/students/ | Lista os Alunos |
| GET | api/student/{id} | Retorna os dados de um Aluno |
| POST | api/student/create | Cria um Aluno novo |
| PUT | api/student/{id}/edit | Atualiza os dados de um Aluno |
| DELETE | api/student/{id}/delete | Elimina um Aluno |

Esta estrutura é aplicada a todos os modelos principais da aplicação, como Alunos, Docentes, Cursos, entre outros.

Todos os endpoints protegidos exigem que o token de autenticação seja enviado no cabeçalho do pedido. Caso o token esteja ausente ou inválido, a API responde com o código[[2]](#footnote-2) 401. Em caso de sucesso, é devolvido o código apropriado (200 ou 201) acompanhado dos dados solicitados, quando aplicável.

# Interface e design visual

A interface da aplicação não serve apenas para interagir com as funcionalidades disponíveis – ela representa a imagem do instituto. Sendo uma aplicação com o objetivo de ser usada tanto internamente, por alunos e docentes, como externamente por empresas e os seus representantes, foi necessário planear uma nova “face” para o sistema.

Com esse objetivo em mente, foi criado design que respeita princípios de simplicidade, clareza e consistência visual, garantindo uma experiência de utilização mais acessível para todos os perfis de utilizador.

## Conceção visual do sistema

A estrutura visual da aplicação baseia-se num *design system* com o objetivo de garantir consistência visual em toda a interface da aplicação, facilitar a reutilização de componentes e acelerar o desenvolvimento. Neste sistema foram definidas as cores, tipografia, espaçamentos e componentes visuais a utilizar em todas as páginas.

### Cores

A paleta de cores foi pensada para refletir a identidade visual do instituto, escolhendo como cor de destaque a cor tijolo, representativa do ISEC. Para garantir legibilidade e conforto visual, foram incluídas cores neutras destinadas a fundos e textos. Além disso, uma cor secundária foi adicionada para criar contrastes e trazer dinamismo ao design.

Cada cor da paleta é composta por nove tonalidades, organizadas numa escala que vai do tom mais claro ao mais escuro. Esta estrutura permite utilizar diferentes intensidades da mesma cor conforme o contexto facilitando a aplicação coerente e intuitiva das cores em toda a interface.

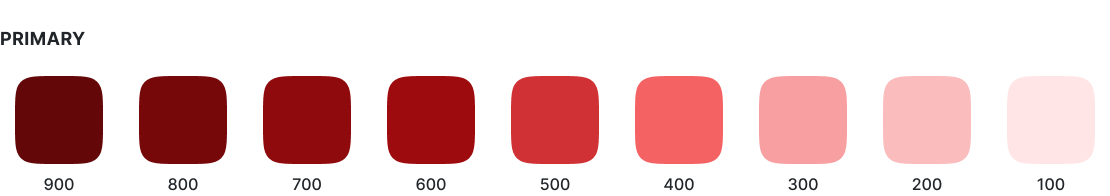


Figura 4 - Representação de uma cor com as suas nove variações

### Tipografia

A tipografia da aplicação utiliza duas fontes distintas, com papéis complementares na hierarquia visual da interface.

* **Montserrat** é aplicada em títulos, subtítulos e textos de maior destaque.
* **Inter** é utilizada no corpo de texto.

Os tamanhos tipográficos foram definidos com base numa escala proporcional do tipo *Minor Third*, que utiliza uma progressão harmónica para garantir consistência entre os diferentes níveis de texto.

O cálculo é feito a partir de um tamanho base — por exemplo, 16px — e os restantes tamanhos são obtidos multiplicando esse valor pelo fator da escala (aproximadamente 1,2). Assim, os níveis seguintes seriam aproximadamente 19px, 23px, 28px, e assim sucessivamente. Este sistema assegura uma evolução coerente entre títulos, subtítulos e corpo de texto, reforçando a hierarquia visual da interface.

### Espaçamentos

Para garantir uma experiência visual harmoniosa e adaptada a diferentes dispositivos, foram definidos dois sistemas de espaçamentos distintos: um para dispositivos com ecrãs de maiores dimensões e outro para dispositivos móveis, como telemóveis. Em ambos os casos, aplicou-se uma escala modular semelhante à utilizada na tipografia, mas com fatores de escala diferentes.

Nos dispositivos com ecrãs maiores, utilizou-se a escala *Golden Ratio* (fator de escala 1,618), que proporciona uma progressão suave e visualmente agradável. Para dispositivos móveis, adotou-se a escala *Major Second* (fator de escala 1,125), oferecendo uma progressão mais subtil e adequada ao espaço limitado disponível.

### Componentes

Com todos os estilos definidos, avançou-se para a criação dos componentes da interface, como a barra de navegação, botões, cartões, entre outros. Estes componentes foram desenvolvidos de forma modular e reutilizável, podendo ser aplicados em diferentes páginas e contextos, adaptando-se aos diversos perfis de utilizador.

Esta abordagem assegura uma experiência visual coerente e ao mesmo tempo simplifica a manutenção e permite a expansão do sistema.

## *Design* da Aplicação

Com a identidade visual do sistema definida, avançou-se para o desenho dos ecrãs da aplicação, desenvolvidos a partir das funcionalidades requeridas. A estrutura da interface foi organizada em quatro fluxos principais, correspondentes aos diferentes perfis de utilizador: Administrador, Docente, Aluno e Representante.

Para assegurar flexibilidade e facilitar a manutenção, as páginas foram projetadas de maneira modular e reutilizável, permitindo o compartilhamento de componentes comuns entre os diferentes fluxos. Essa abordagem modular também possibilita a adaptação das interfaces conforme as permissões específicas de cada utilizador, incluindo o acesso avançado de alguns docentes, similar ao do administrador, conforme definido nos requisitos do sistema.

Adicionalmente, a navegação foi estruturada tendo em conta a experiência do utilizador, garantindo rapidez e clareza no acesso às diversas funcionalidades, minimizando a curva de aprendizagem e melhorando o uso da aplicação. Foram criados 74 ecrãs que abrangem todas as funcionalidades previstas para os diferentes perfis, assegurando uma cobertura completa das necessidades do sistema.

A documentação completa relativa ao design encontra-se no Apêndice X, onde estão descritas todas as decisões tomadas ao nível visual.

## Implementação da Interface

A implementação da interface do sistema foi feita utilizando React, o que possibilitou estruturar o frontend em componentes modulares e reutilizáveis. O CSS foi usado para personalizar os estilos. Além disso, o Bootstrap foi incorporado para acelerar o desenvolvimento de layouts responsivos e consistentes, facilitando a adaptação da aplicação a diferentes dispositivos. Essa combinação de tecnologias resultou numa interface flexível e de fácil manutenção.

### Organização do projeto

A estrutura do projeto seguiu a organização típica de uma aplicação desenvolvida em React, com uma separação clara entre os diferentes tipos de elementos: componentes, páginas, estilos, serviços, entre outros. Esta divisão permite manter o código limpo, modular e fácil de escalar. Na raiz do projeto temos:

* ***assets*** – Contém os recursos estáticos utilizados na aplicação, como imagens, ícones e ficheiros gráficos.
* ***styles*** –Contém os ficheiros CSS responsáveis pela definição visual da aplicação.
* ***componentes*** –Reúne os componentes reutilizáveis da interface, como botões, cabeçalhos, cartões, entre outros.
* ***pages*** –Contém todas as páginas da aplicação.
* ***helpers*** – Inclui funções auxiliares que suportam a lógica do *frontend*. Estas funções servem, por exemplo, para realizar pedidos à API ou para outras operações se suporte.

### Arquitetura da Interface

#### Navegação e gestão de rotas

A navegação da aplicação foi implementada utilizando a biblioteca *react-router-dom*, com o controlo centralizado no ficheiro *App.js*. Este ficheiro define os caminhos que permitem o acesso às várias páginas e módulos do sistema, assegurando que cada utilizador visualiza apenas as secções adequadas ao seu perfil e respetivas permissões. Esta abordagem garante uma navegação fluida e intuitiva, facilitando a utilização diária.

#### Componentes reutilizáveis e flexibilidade

Os componentes da interface foram organizados na pasta *components* e construídos com base nos elementos do *design system*. Estes componentes são reutilizáveis e flexíveis, permitindo a passagem de estilos adicionais através da propriedade *className*, o que possibilita a adaptação do seu estilo conforme o contexto de utilização.

#### Estrutura modular das páginas

A estrutura das páginas segue uma lógica modular, com cada módulo funcional organizado na pasta *pages/Modules*. Aqui cada módulo possui as suas páginas específicas de listagem, visualização e edição. Esta organização facilita a escalabilidade do sistema e mantém o código organizado e de fácil gestão.

Exemplo de um module

#### Definição e organização dos estilos

A gestão dos estilos na aplicação foi pensada de forma modular e organizada. As variáveis CSS que sustentam a identidade visual do sistema estão definidas em ficheiros separados dentro da pasta *styles*, com cada ficheiro dedicado a uma responsabilidade específica – por exemplo, cores (*\_colors.css*) ou tipografia (*\_fonts.css*). Estes ficheiros são todos importados através de um único ponto central, o ficheiro *imports.css*, garantindo que todas as definições ficam disponíveis globalmente na aplicação.

A nomenclatura das variáveis segue uma convenção coerente, baseada nas diretrizes do *design system*. Os nomes utilizam uma estrutura padronizada com valores dinâmicos indicados entre chavetas {}*[[3]](#footnote-3)*:

* **Cores:** *--{*nome*}-color-{***valor***}* – onde o valor vai de 100 a 900 (em saltos de 100), representando diferentes intensidades da mesma cor.
* **Espaçamentos:** *--spacing-{***tamanho***}* – com valores em *t-shirt size* (xs, s, m, l, xl, 2xl, 3xl, 4xl).
* **Tamanhos de texto[[4]](#footnote-4):** *--fs--{***valor***}* – de 100 a 950, com saltos de 100.
* **Tamanhos de ícones:** *--icons-size-{***tamanho***}* – também seguindo o padrão *t-shirt* size (xs a l).

Adicionalmente, foram definidas cores utilitárias que seguem uma convenção padronizada para facilitar o seu uso em diferentes contextos da interface. Estas cores não estão associadas diretamente à identidade visual da aplicação, mas servem como um conjunto de apoio para elementos funcionais.

Cada cor utilitária possui três variantes principais:

* *{***cor***}* – a cor base.
* {**cor**}*-lighter* – uma versão mais clara da cor.
* {**cor**}*-darker* – uma versão mais escura da cor.

O conjunto inclui cores como vermelho, verde, azul, roxo, rosa, turquesa, castanho, laranja e amarelo, assegurando diversidade suficiente para múltiplas necessidades visuais sem comprometer a coerência do design.

Por sua vez, o ficheiro *App.css* aplica estilos base e genéricos à interface, definindo comportamentos comuns para elementos HTML e componentes globais. Este ficheiro funciona como camada de normalização visual da aplicação, aplicando regras consistentes que usam as variáveis importadas a partir do *imports.css*. Esta abordagem permite manter a aplicação visualmente consistente, facilita a manutenção de estilos e torna a adição de novos elementos mais eficiente e alinhada com o sistema existente.

#### Suporte para *Markdown*

Durante o desenvolvimento do *frontend*, surgiu a necessidade de permitir que os utilizadores tivessem algum controlo sobre a formatação do texto em determinados campos (por exemplo, a descrição de propostas ou cursos). Como o foco da aplicação não é fornecer um editor de texto avançado, criar um sistema completo de edição de texto seria demasiado complexo e desnecessário para o objetivo do projeto.

Para resolver este problema, optou-se por utilizar Markdown [12], uma linguagem de marcação simples que possibilita a formatação de texto de forma intuitiva e leve. O texto é armazenado na base de dados em formato literal, ou seja, contendo todos os símbolos que definem o estilo[[5]](#footnote-5).

No frontend, a biblioteca “react-markdown” é responsável por interpretar este conteúdo e renderiza-lo como HTML. Para garantir que o resultado respeita as definições visuais da aplicação, o componente *<Markdown>...</Markdown>* é sempre envolvido por uma *<div class="markdown">*.

# Casos de uso

Entre os diversos fluxos implementados durante o desenvolvimento, apresentam-se a seguir aqueles que melhor representam as funcionalidades-chave e os objetivos do sistema. Estes fluxos foram selecionados por refletirem processos essenciais e por permitirem uma compreensão mais clara do funcionamento global da aplicação. Através da análise destes exemplos, será possível observar tanto a interação do utilizador com a interface como a lógica subjacente que suporta as operações mais relevantes do sistema.

## Autenticação de um utilizador (*login*)

O processo de autenticação de um utilizador segue o procedimento habitual, no qual são solicitadas as credenciais (email e palavra-passe). Estas são enviadas à API para validação. Em caso de sucesso, a API retorna um token de autenticação juntamente com informações adicionais relevantes, como o tipo de utilizador e as respetivas permissões (caso seja um docente). Esta informação é então utilizada para apresentar a página inicial (*dashboard*) correta, adaptada ao perfil do utilizador autenticado.

Além disso, o sistema verifica, no momento do acesso, se já existe um token armazenado localmente. Caso este token seja considerado válido – após uma verificação junto à API – o utilizador é automaticamente redirecionado para a sua *dashboard*, evitando assim a necessidade de efetuar o login manualmente.

Uma imagem com céu, captura de ecrã, texto, edifício

Os conteúdos gerados por IA podem estar incorretos.

Figura 5 - Página de *Login*

**Visão do Docente**

Um dos casos em que a página inicial apresenta maior variação é na visão do docente. Para além das funcionalidades básicas, como consultar as suas orientações, submeter propostas e acompanhar as suas propostas submetidas, o sistema adapta-se para oferecer acessos adicionais semelhantes aos de um administrador, dependendo das permissões atribuídas ao utilizador.

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Os conteúdos gerados por IA podem estar incorretos.

Figura 6 - Página inicial com as permissões padrão

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Os conteúdos gerados por IA podem estar incorretos.

Figura 7 - Página inicial com todas as permissões

## Filtragem de Listas

Considerando que o sistema lida com grandes volumes de informação, a funcionalidade de filtragem revela-se essencial para facilitar a localização rápida e eficiente dos dados desejados. Sem esta ferramenta, seria complexo e demorado navegar pelas diversas listas.

A filtragem funciona de forma intuitiva: o utilizador pode clicar no cabeçalho da lista para inserir termos de pesquisa diretamente, ou utilizar menus *dropdown* que permitem selecionar critérios específicos para refinar a informação apresentada. Esta abordagem combina simplicidade e flexibilidade, tornando a busca mais eficaz.

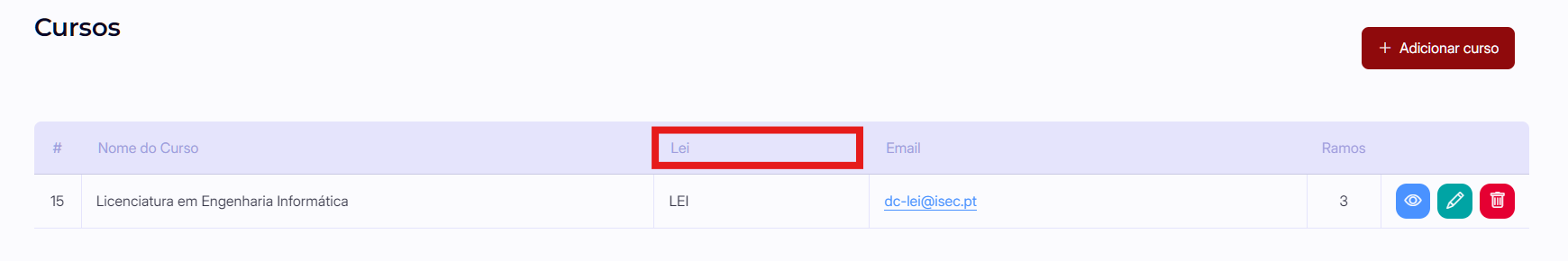


Figura 8 - filtragem de listas

Embora exista ainda margem para melhorias e otimizações futuras, a filtragem implementada já garante uma usabilidade adequada, permitindo ao utilizador restringir a visualização conforme vários critérios e encontrar rapidamente o que procura.

## Gestão de Docentes

Este fluxo ilustra a gestão dos docentes pela visão do administrador, abrangendo as funcionalidades de listagem, visualização, adição/edição e remoção.

A listagem dos docentes apresenta todos os docentes registados no sistema, com a possibilidade de filtragem (como referido anteriormente) para facilitar a consulta rápida. Nesta vista, o administrador tem acesso direto às opções de visualizar, editar, remover e adicionar um novo docente.

Uma imagem com texto, software, número, Página web

Os conteúdos gerados por IA podem estar incorretos.

Figura 9 - Listagem de docentes

O sistema permite a criação de novos registos de docentes, bem como a edição das informações existentes. Esta funcionalidade é essencial para manter os dados atualizados, incluindo contactos, áreas de especialização e outros detalhes relevantes.

A funcionalidade de visualização de um docente permite consultar informação relevante sobre o docente.

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Os conteúdos gerados por IA podem estar incorretos.

Figura 10 - Visualização de um docente

O mesmo ecrã é utilizado tanto para a criação de um novo docente como para a edição de um registo existente. A diferença principal está no preenchimento prévio dos campos: no caso da edição, os dados do docente são carregados por defeito, enquanto na criação os campos estão vazios. Esta abordagem visa reutilizar código, dado que ambas as operações partilham a mesma interface e estrutura de dados.

Uma imagem com texto, captura de ecrã, número, Paralelo

Os conteúdos gerados por IA podem estar incorretos.

Figura 11 - Edição de um docente

No sistema, a remoção pode assumir formas distintas conforme o contexto. Em casos específicos, como a gestão de docentes a remoção não elimina os registos definitivamente, mas desativa-os. Esta abordagem garante que a informação permaneça preservada para futuras consultas, mantendo a integridade e consistência dos dados. Em outros contextos onde não existe esta necessidade, a remoção poderá ser efetiva, eliminando totalmente o registo do sistema.

Uma imagem com texto, software, número, Ícone de computador

Os conteúdos gerados por IA podem estar incorretos.

Figura 12 - Listagem de com um docente removido

Os docentes removidos aparecem na lista com um fundo escuro, como se pode ver na Figura 12.

## Gestão de Calendários

## Submissão de uma Proposta

(todo)

# Desenvolvimento Futuro

Tendo em conta que este projeto foi desenvolvido no âmbito de uma unidade curricular com duração limitada, não foi possível implementar na íntegra todas as funcionalidades idealizadas. Ainda assim, o desenvolvimento foi orientado por uma visão completa do sistema, e todas as decisões tomadas – tanto a nível de estrutura como de implementação – tiveram em conta o conjunto completo das funcionalidades previstas.

Algumas funcionalidades foram concluídas, outras ficaram parcialmente desenvolvidas (nomeadamente ao nível da interface gráfica), e há ainda elementos que se encontram apenas definidos ou planeados, mas não implementados.

A tabela seguinte apresenta um resumo do estado de desenvolvimento de cada funcionalidade ou módulo do sistema:

Figura 13 - Estado das fucnionalidades[[6]](#footnote-6)

|  |  |  |
| --- | --- | --- |
| Funcionalidade | *Frontend* | *Backend* |
| Autenticação | Completo | Completo |
| Registo de Docentes | Completo | Completo |
| Registo de Alunos | Completo | Completo |
| Registo de Empresas/Representantes | Completo | Completo |
| Gestão de Áreas Científicas | Completo | Completo |
| Gestão de Cursos | Completo | Completo |
| Gestão de Calendários | Completo | Completo |
| Gestão de Docentes | Completo | Completo |
| Gestão de Alunos | Completo | Parcial |
| Gestão de Empresas | Completo | Completo |
| Gestão de Representantes | Completo | Completo |
| Gestão de Propostas | Completo | Completo |
| Gestão de Candidaturas | Completo | Planeado |
| Submissão de Propostas (Empresas) | Completo | Completo |
| Submissão de Propostas (Docentes) | Completo | Completo |
| Submissão de Candidaturas | Planeado | Planeado |
| Filtragem de Lista | Parcial | Parcial |
| Exportação de Atribuições | Planeado | Planeado |
| Preenchimento automático do ficheiro da Proposta | — | Completo |
| Preenchimento automático de Protocolos | — | Planeado |
| Algoritmo de Atribuição | — | Planeado |
| Histórico de Comunicação | Por fazer | Por fazer |
| Notificação de atribuições às empresas | — | Planeado |

Em reflexão, de todo o trabalho realizador existem alguns pontos que poderiam ser melhorados e ou adicionados à aplicação, como por exemplo:

* Melhorar a filtragem de listas com mais opções, como filtrar propostas e alunos pelo calendário a que pertencem, entre outros.
* Melhorar a interface para dispositivos moveis que, apesar de estar implementada a estruturação pode ser melhorada para permitir uma melhor experiência de utilização em dispositivos com ecrãs menores.
* Aplicar um sistema de importação de dados completo, como a importação de Alunos, Docentes, Propostas, Empresas e os seus Representantes, entre outros dados relevantes.
* Criar um sistema de *log* para guardar toda as alterações feitas no sistema para poder manter um histórico de tudo….

# Conclusão

Apreciação do trabalho feito

Referencias bibliográficas

|  |  |
| --- | --- |
| [1] | Django Software Foundation, “The web framework for perfectionists with deadlines.,” [Online]. Available: https://www.djangoproject.com. [Acedido em 27 junho 2025]. |
| [2] | Meta Platforms, Inc, “React,” [Online]. Available: https://react.dev. [Acedido em 27 junho 2025]. |
| [3] | K. Brush, “What is the MoSCoW method?,” 29 março 2023. [Online]. Available: https://www.techtarget.com/searchsoftwarequality/definition/MoSCoW-method. [Acedido em 25 agosto 2025]. |
| [4] | Bootstrap team, “Bootstrap · The most popular HTML, CSS, and JS library in the world.,” [Online]. Available: https://getbootstrap.com. [Acedido em 26 junho 2025]. |
| [5] | Microsoft, “Visual Studio Code,” Microsoft, [Online]. Available: https://code.visualstudio.com. [Acedido em 29 maio 2025]. |
| [6] | A. Uspenski, “Why VS Code remains a developer favorite, year after year,” 25 abril 2023. [Online]. Available: https://shiftmag.dev/vs-code-171/?utm\_source=chatgpt.com. [Acedido em 25 agosto 2025]. |
| [7] | JetBrains, “PyCharm,” JetBrains, [Online]. Available: https://www.jetbrains.com/pycharm/?source=google&medium=cpc&campaign=EMEA\_en\_WEST\_PyCharm\_Branded&term=pycharm&content=698987581404&gad\_source=1&gad\_campaignid=14123077402&gclid=CjwKCAjwi-DBBhA5EiwAXOHsGVWZECsl6gVPToYcfhEKlfGmqaiBEyoQGJCnWr1xOuD7hVrcMKXZz. [Acedido em 29 maio 2025]. |
| [8] | A. Ryabtsev, “10 Best IDEs and Code Editors for Python/Django Development,” 28 outubro 2024. [Online]. Available: https://djangostars.com/blog/python-ide/?utm\_source=chatgpt.com. [Acedido em 25 agosto 2025]. |
| [9] | GitHub, Inc., “About GitHub,” [Online]. Available: https://github.com/about. [Acedido em 25 agosto 2025]. |
| [10] | Figma, Inc., “Figma,” Figma, Inc., [Online]. Available: https://www.figma.com. [Acedido em 29 maio 2025]. |
| [11] | L. Gupta, “REST API Tutorial: What is REST?,” 1 abril 2025. [Online]. Available: https://restfulapi.net. [Acedido em 20 julho 2025]. |
| [12] | M. Cone, “Markdown Guide,” 2025. [Online]. Available: www.markdownguide.org. [Acedido em 26 agosto 2025]. |
| [13] | L. Gupta, “HTTP Status Codes,” 9 agosto 2024. [Online]. Available: https://restfulapi.net/http-status-codes. [Acedido em 20 julho 2025]. |
| [14] | “Markdown Syntax Cheatsheet,” 2021. [Online]. Available: https://www.markdown-cheatsheet.com. [Acedido em 26 agosto 2025]. |

Apêndices

Text

Description automatically generated with medium confidence

1. No contexto de um IDE, *debug* refere-se ao processo de execução controlada de um programa para identificar, analisar e corrigir erros ou comportamentos inesperados. Os IDEs oferecem ferramentas como pausar a execução, inspecionar variáveis e percorrer o código passo a passo facilitando a deteção e correção de erros durante o desenvolvimento. [↑](#footnote-ref-1)
2. A lista completa de códigos de resposta HTTP e os seus respetivos significados pode ser consultada na referência [13]. [↑](#footnote-ref-2)
3. Os valores entre chavetas **{}** representam parâmetros variáveis que são substituídos conforme o contexto de utilização (por exemplo, nomes de cores, tamanhos ou intensidades). [↑](#footnote-ref-3)
4. Os elementos HTML como h1, h2, entre outros, utilizam por defeito os tamanhos de texto definidos nas variáveis. A aplicação destes tamanhos fora desses contextos ocorre apenas em casos pontuais. [↑](#footnote-ref-4)
5. Para referência adicional da sintaxe de Markdown, pode ser consultado o recurso disponível em [14]. [↑](#footnote-ref-5)
6. O símbolo “—” indica que a funcionalidade não está presente na respetiva secção, ou seja, não é responsabilidade desse componente. [↑](#footnote-ref-6)