



Técnico de Gestão e Programação de Sistemas Informáticos

Prova de Aptidão Profissional

Jet Hire



Cantanhede

Tomé Almeida, 12º TGPSI, Nº 15

2022 / 2025

Escola Técnico Profissional de Cantanhede

Técnico de Gestão e Programação de Sistemas Informáticos

Prova de Aptidão Profissional: Jet Hire

Equipa de Acompanhamento: Michael Teixeira, Elisabete Cavaleiro e Ana Marques

Cantanhede

2024/2025

CRIATIVIDADE É SÓ CONECTAR AS COISAS.

Steve Jobs

Agradecimentos

Gostaria de agradecer em primeiro lugar ao meu **coordenador de curso**, professor **Michael Teixeira**, que se mostrou sempre disponível para me esclarecer dúvidas e que me orientou nas diversas decisões que precisei de tomar ao longo da realização do curso e da respetiva Prova de Aptidão Profissional. Vejo nele uma inspiração e um modelo a seguir.

Um agradecimento muito especial à professora **Sónia Moço**, minha ex-diretora de turma, que, mesmo depois de ter saído da escola, nunca deixou de me apoiar. Continuou sempre presente, a ajudar, a incentivar e a acreditar em mim — e isso, para mim, valeu muito. Foi mais do que uma professora, foi uma verdadeira orientadora e alguém em quem sempre pude confiar.

À professora **Ana Marques**, minha diretora de turma atual, quero também agradecer sinceramente por todo o apoio, pela atenção constante e por estar sempre disponível, especialmente nas alturas em que tudo parecia mais difícil. A sua preocupação genuína com cada aluno fez toda a diferença neste último ano.

Aos **restantes professores** do curso, o meu obrigado por tudo o que ensinaram e por todo o esforço ao longo dos três anos. Para além do que aprendemos nas aulas, guardo também a forma como nos ajudaram a crescer enquanto pessoas.

Ao meu tutor de estágio, **Cláudio Oliveira**, agradeço por me ter recebido de braços abertos, por confiar em mim desde o início e por me desafiar a sair da minha zona de conforto. Com ele aprendi muito mais do que apenas técnicas — aprendi como é o mundo do trabalho.

Ao meu colega de estágio, **Emanuel Oliveira**, um grande obrigado por estar sempre por perto, pronto a ajudar e a ensinar com paciência. Trabalhar ao seu lado tornou tudo mais leve e mais fácil — foi realmente bom ter alguém assim durante o estágio.

Quero também deixar uma palavra de carinho à **equipa de docentes e assistentes da escola**, em especial à **Dona Vera**, à **Dona Sandra** e à **Dona Isabel**. Sempre simpáticas, sempre disponíveis, sempre com uma palavra amiga. A forma como tratam os alunos cria mesmo um ambiente especial dentro da escola.

E, por fim, um agradecimento enorme à minha **família** e aos meus **amigos**, que estiveram sempre comigo em tudo. Ajudaram-me com testes, deram opiniões, testaram funcionalidades e ouviram-me quando eu só precisava de desabafar. Este projeto também é um bocadinho deles.

Índice Geral

Introdução (Português)	1
Introduction (Inglês)	3
Planeamento.....	5
Fundamentação	5
Recursos	7
Hardware	7
Software.....	7
Metodologia	9
Atividades.....	10
Fase de Definição	10
Fase de Desenvolvimento.....	10
Fase de Manutenção	11
Cronograma.....	11
Instalações.....	12
Visual Studio Code.....	12
MongoDB Compass	13
Node.js (node / npm).....	14
Github Desktop.....	15
Inkscape	16
Krita.....	17
Postman	18
Sincronização entre dois computadores	19
Criação do repositório	19
Push	20
Pull	21
Sitemap	24
Projeto	25
Elementos gerais.....	25
Download e preparação do <i>template</i>	25
Instalação de dependências	25

Execução do projeto.....	25
Publicação online com domínio personalizado	25
<i>Deploy automático no Github e Vercel.....</i>	26
Ligaçāo a domínios personalizados.....	26
Certificados SSL e HTTPS.....	27
Considerações legais e políticas da plataforma	27
Termos e Condições	27
Política de Privacidade.....	28
Política de Cookies	28
Logótipo	29
Escolha da paleta de cores.....	29
Cor principal	29
Cores secundárias	31
Cores neutras	32
Implementação da paleta de cores no projeto	33
Valorização do <i>design</i> e acessibilidade (UI & UX)	33
Criação de um ambiente seguro	34
Ligaçāo à base de dados com Mongoose	34
O que é o MongoDB?	34
O que é o Mongoose?	35
Instalação.....	35
Ligaçāo à base de dados	35
Submissão segura de dados com API e Mongoose	36
Exemplo com <i>fetch</i>	36
API (Back-End)	36
Carregamento inteligente de dados com React	37
Declaração do estado de carregamento	37
Requisição de dados com <i>useEffect</i>	37
Firebase – Base de dados em tempo real.....	38
Criação da conta.....	39
Integração com o projeto	39
Cloudinary — Gestão de imagens e documentos	40
Funções para envio e eliminação.....	40
NextAuth – Sistema de autenticação com sessões seguras	41
Autorização personalizada com Credentials Provider	41
Sessões com JWT.....	43
O que é um JWT?	43
Sessões acessíveis no front-end.....	44

Exemplo prático	44
Botões de <i>login</i> com NextAuth	45
Formulários dinâmicos e reutilizáveis em React	45
Projeto principal	46
Edição do <i>template</i>	46
Rodapé.....	46
Barra de navegação	47
Página inicial	47
Página de ofertas.....	53
Página de empresas.....	55
Página de candidatos.....	58
Página de início de sessão	60
Página de registo	60
Página de perfil de utilizador	61
Página de edição de perfil	63
Página de perfil de empresa	67
Página de detalhes de oferta	69
Página de contactos	72
Página de chat com AI	74
Registo de utilizadores.....	75
Tratamento na API	76
Barra de navegação dinâmica	77
Como funciona?	77
Com sessão ativa	77
Sem sessão ativa	78
Página de candidatos	78
Estrutura geral da página	79
API de candidatos	79
Pesquisa em tempo real	80
Ordenação por data de registo	80
Renderização condicional com mensagens de <i>fallback</i>	81
Perfil dinâmico e adaptável	83
Estrutura da página	83
Carregamento dinâmico com <i>getServerSideProps</i>	84
Renderização condicional	85
Percentagem de preenchimento	85
Segurança e personalização.....	86

Edição de perfil.....	87
Organização modular	88
Gestão do estado e validação.....	88
Portfólio com <i>upload presets</i>	88
Envio final e atualização da base de dados	89
Página de empresas.....	89
Listagem de empresas e pesquisa em tempo real.....	89
Pedido de Registo de Empresa	90
Página de detalhes de empresas.....	91
Obtenção e apresentação dos dados	91
Dados públicos	92
Página de ofertas.....	92
Obtenção de ofertas.....	92
Filtros avançados.....	93
Filtro “Mostrar apenas favoritos”	95
Página de detalhes das ofertas	96
Tempo de publicação	96
Candidaturas com modal de confirmação.....	96
Favoritos com estrela dinâmica	97
Página de contactos.....	97
Jet AI	98
Funcionamento geral.....	98
Envio de mensagem	99
Lógica da API	99
Comportamento inteligente	101
Dashboard	102
Middleware de proteção	102
Estrutura de dados	103
Administradores.....	103
Empresas.....	103
Página de erro (404)	104
Barra lateral.....	105
Objetivos da <i>sidebar</i>	105
Funcionamento base	105
Sistema de permissões	105
Exemplo: declarar quem pode ver o quê	106
Estrutura do menu	107

Submenus dinâmicos.....	107
Registo de empresas.....	108
Integração com o Resend	108
O que é o Resend?.....	108
Preparação	108
Processo de validação.....	109
Geração do <i>token</i>	109
Envio do email com o componente TokenEmail	109
Conteúdo do email.....	110
Registo de empresa com <i>token</i>	110
Interface de registo	111
Criação da conta e geração automática do nome de utilizador	111
Confirmação e envio de email com credenciais	112
Convite de novos membros para a equipa	113
Interface de gestão de membros	113
Validação e expiração de convites	114
Envio do convite por email	114
Criação de conta de convidado	115
Interface de convite	115
Registo com base no <i>token</i>	116
Visualização da equipa.....	116
Interface de membros da equipa.....	117
API de apresentação de membros da equipa	117
Edição de perfil de empresa	117
Estrutura geral da página	118
Carregamento inicial dos dados.....	118
Comparação de alterações	119
Gestão de imagens com o Cloudinary.....	119
Organização das descrições	120
Atualização final do perfil	120
Segurança.....	121
Edição de perfil de utilizador	121
Lógica de carregamento de perfil	121
Dados apresentados na interface	121
Atualização dos dados	122
Segurança.....	123
Benefícios da modularização	124
Visualização de empresas	124

API de listagem.....	124
Interface de apresentação.....	125
Criação de oferta	126
Estrutura do formulário	126
Envio dos dados com validação de sessão	127
Confirmação visual	128
Visualização e gestão de ofertas	128
Ativação e remoção de ofertas	129
Visualização de candidatos	129
Marcação de candidatos como "visto"	130
Atualização visual imediata	131
Registo de <i>logs</i>	132
Estrutura do <i>log</i>	132
Criação de <i>logs</i> automáticos.....	132
Exemplo prático	133
Privacidade e segurança	133
Página inicial da <i>dashboard</i>	134
Estatísticas	134
Gráficos de barras	134
Gráfico de linhas.....	135
Tabela de <i>logs</i>	136
Live chat com Firebase	137
Porquê Firebase?.....	137
Organização do Realtime Database	137
Leitura de mensagens em tempo real.....	138
Envio de mensagens	138
Integração com a sessão.....	139
Interface visual.....	140
Testes e validação da aplicação	141
Testes técnicos	141
Validação com utilizadores	141
Conclusão.....	143
Webgrafia	144

Índice de Figuras

Figura 1 - Metodologia Kanban	9
Figura 2 - Cronograma	11
Figura 3 - Instalação Visual Studio Code	12
Figura 4 - Extensões.....	12
Figura 5 - Instalação MongoDB Compass: Passo 1	13
Figura 6 - Instalação MongoDB Compass: Passo 2	13
Figura 7 - sudo apt install nodejs	14
Figura 8 - sudo apt install npm	14
Figura 9 - Verificação do npm	14
Figura 10 - Instalação GitHub Desktop: Passo 1	15
Figura 11 - Instalação GitHub Desktop: Passo 2.....	15
Figura 12 - Instalação GitHub Desktop: Passo 3.....	16
Figura 13 - Instalação do Inkscape: Passo 1	16
Figura 14 - Instalação do Krita: Passo 1	17
Figura 15 - Instalação do Krita: Passo 2	17
Figura 16 - Instalação do Postman: Parte 1.....	18
Figura 17 - Instalação do Postman: Parte 2.....	18
Figura 18 - Sincronização entre dois computadores: Criação do repositório	19
Figura 19 - Sincronização entre dois computadores: Publicação do repositório	19
Figura 20 - Sincronização entre dois computadores: Commit.....	20
Figura 21 - Sincronização entre dois computadores: Push	20
Figura 22 - Sincronização entre dois computadores: Verificação do repositório	21
Figura 23 - Sincronização entre dois computadores: Clonar um repositório	21
Figura 24 - Sincronização entre dois computadores: Fetch	22
Figura 25 - Sincronização entre dois computadores: Pull	22
Figura 26 – Sitemap principal.....	24
Figura 27 - Domínios da dashboard	26
Figura 28 - Termos e Condições	28
Figura 29 - Política de privacidade	28
Figura 30 - Política de cookies	29
Figura 31 - Logótipo.....	29
Figura 32 - Cor principal	30
Figura 33 - Escolha de um tom claro	30
Figura 34 - Escolha de um tom escuro	30
Figura 35 - Criação de um gradiente claro.....	31

Figura 36 - Criação de um gradiente escuro.....	31
Figura 37 - Amarelo	32
Figura 38 - Azul.....	32
Figura 39 - Verde	32
Figura 40 - Cores neutras.....	32
Figura 41 - Dashboard da Firebase	39
Figura 42 - Rodapé inicial.....	46
Figura 43 - Rodapé final	46
Figura 44 - Barra de navegação inicial.....	47
Figura 45 - Barra de navegação final	47
Figura 46 - Destaque principal inicial	47
Figura 47 - Destaque principal final.....	47
Figura 48 – Slider de categorias inicial	48
Figura 49 – Slider de categorias final.....	49
Figura 50 - Como funciona inicial.....	49
Figura 51 - Como funciona final	49
Figura 52 - Ofertas mais recentes inicial.....	50
Figura 53 - Ofertas mais recentes final	50
Figura 54 - Procurar trabalho inicial	51
Figura 55 - Procurar trabalho final.....	51
Figura 56 - Candidates recentes inicial	52
Figura 57 - Candidatos recentes final	52
Figura 58 - Cria o teu perfil inicial.....	53
Figura 59 - Cria o teu perfil final	53
Figura 60 - Secção de pesquisas inicial.....	54
Figura 61 - Secção de pesquisas final	54
Figura 62 - Menu lateral - Modalidade	55
Figura 63 - Menu lateral - Tipo de trabalho.....	55
Figura 64 - Apresentação de oferta.....	55
Figura 65 - Placeholder do formulário do registo de empresas.....	56
Figura 66 - Formulário de registo de empresas	56
Figura 67 - Cabeçalho da página de empresas.....	57
Figura 68 - Apresentação de empresas	57
Figura 69 - Cabeçalho de página de candidatos	58
Figura 70 - Dropdown de ordenação.....	58
Figura 71 - Cartão de utilizador	59

Figura 72 - Página de início de sessão	60
Figura 73 - Página de registo	60
Figura 74 - Banner	61
Figura 75 - Competências profissionais	61
Figura 76 - Educação	62
Figura 77 - Experiência	62
Figura 78 - Portfólio	62
Figura 79 - Portfólio com hover	62
Figura 80 - Barra lateral	63
Figura 81 - Dados básicos	63
Figura 82 - Descrições	64
Figura 83 - Competências profissionais	64
Figura 84 - Educação e experiência	65
Figura 85 - Exemplo de elemento de educação	65
Figura 86 – Portfólio	66
Figura 87 - Exemplo de portfólio	66
Figura 88 - Links pessoais	66
Figura 89 - Banner de empresa	67
Figura 90 - Identidade visual da empresa	67
Figura 91 - "Contacte-nos" de empresa	67
Figura 92 - Descrição de empresas	68
Figura 93 - Barra lateral da página de empresas	69
Figura 94 - Identidade visual da página de oferta	69
Figura 95 - Informação geral da oferta	70
Figura 96 - Barra lateral de oferta	71
Figura 97 - Tags da oferta	72
Figura 98 - Formulário de contactos	73
Figura 99 - Apresentação pessoal	73
Figura 100 - Slider de testemunhos fictícios	74
Figura 101 - Página do Jet AI	75
Figura 102 - Barra de navegação sem sessão	78
Figura 103 - Barra de navegação com sessão	78
Figura 104 - Dropdown da página de candidatos	81
Figura 105 - Exemplo de pesquisa por "Mais recente"	81
Figura 106 - Candidatos a carregar	82
Figura 107 - Sem resultado de candidatos	82

Figura 108 - Pesquisa de candidatos válida	83
Figura 109 - Perfil completo	85
Figura 110 - Perfil por concluir	85
Figura 111 - Exemplo de perfil por concluir	86
Figura 112 - Exemplo de perfil completo	87
Figura 113 - Exemplo de pesquisa de empresa	90
Figura 114 - Exemplo de filtragem combinada	95
Figura 115 - A mostrar favoritos	95
Figura 116 - Exemplo do tempo de publicação	96
Figura 117 - Botão para se candidatar	97
Figura 118 - Botão para remover a candidatura	97
Figura 119 - Modal de confirmação de candidatura	97
Figura 120 - Modal de confirmação de remoção	97
Figura 121 - Botão para tornar favorito	97
Figura 122 - Botão para remover favorito	97
Figura 123 - Exemplo de interação com o Jet AI	101
Figura 124 - Página 404	104
Figura 125 - Barra lateral de administrador	107
Figura 126 - Barra lateral de empresa	107
Figura 127 - Pedido de registo de empresa	108
Figura 128 - Domínios do Resend	109
Figura 129 - Exemplo de mail de token de acesso	110
Figura 130 - Página de registo de empresas	111
Figura 131 - Exemplo de email de credenciais	113
Figura 132 - Listagem de convites	114
Figura 133 - Exemplo de email de convite	115
Figura 134 - Página de convite	116
Figura 135 - Exemplo de membro de equipa	117
Figura 136 - UserMetaCard	122
Figura 137 - UserInfoCard	122
Figura 138 - UserAddressCard	122
Figura 139 - Exemplo de empresa	125
Figura 140 - Exemplo de confirmação visual	128
Figura 141 - Exemplo de oferta ativa e oferta removida	129
Figura 142 - Apresentação de candidatos	130
Figura 143 - Candidato "visto"	132

Figura 144 - Estatísticas da dashboard	134
Figura 145 - Gráfico de barras da dashboard	135
Figura 146 - Gráfico de linhas	136
Figura 147 - Tabela de logs.....	137
Figura 148 - Mensagem de outro utilizador	140
Figura 149 - Mensagem própria	140

Índice de Tabelas

Tabela 1 - Tabela de objetivos	5
Tabela 2 - Estrutura geral da página de empresas.....	118
Tabela 3 - Dados de perfil de utilizador	122
Tabela 4 - Apresentação de empresas	125

Introdução (Português)

Para a minha **Prova de Aptidão Profissional (PAP)**, desenvolvi a **Jet Hire**, uma aplicação web para **ligação entre candidatos e empresas** na área das **Tecnologias de Informação**. A escolha deste projeto surgiu de uma experiência pessoal vivida nas férias de verão, em que, ao tentar procurar um trabalho temporário, percebi o quanto difícil pode ser encontrar oportunidades de forma rápida e acessível. Muitas das vagas não eram divulgadas online, e as que existiam estavam espalhadas por diferentes plataformas, o que me fez sentir que faltava uma solução mais centralizada e adaptada aos tempos atuais.

Com isso em mente, decidi criar uma aplicação que permitisse **apresentar ofertas de emprego de forma clara, acessível e moderna**, dando resposta tanto às necessidades de quem procura emprego, como às das empresas que precisam de recrutar. O foco na área das **TI** surgiu naturalmente, por ser a área que estou a estudar e onde pretendo desenvolver a minha carreira profissional. Ao longo do curso, fui tendo contacto com várias ferramentas ligadas ao desenvolvimento *web*, e esta prova foi a oportunidade perfeita para aplicar esses conhecimentos de forma prática, autónoma e organizada.

A Jet Hire é composta por **duas aplicações distintas**, com objetivos bem definidos:

A **página principal** da aplicação (Jet Hire), voltada para os utilizadores em geral, onde qualquer pessoa pode **procurar ofertas, filtrar resultados, criar e editar o seu perfil profissional, guardar ofertas favoritas e até interagir com um assistente virtual** que responde a dúvidas sobre o funcionamento da plataforma e sobre a área de tecnologia em geral. Esta aplicação foi desenvolvida com **Next.js 12**, em **JavaScript**, utilizando a **estrutura clássica com pages/**, e está ligada a uma base de dados **MongoDB** através da biblioteca **Mongoose**.

A **dashboard** da plataforma, acessível apenas a utilizadores com funções específicas (empresas ou administradores), foi construída de forma independente, recorrendo à versão mais recente do framework: **Next.js 15**, com **App Router, TypeScript, TailwindCSS** e uma estrutura mais moderna e escalável. A dashboard permite às empresas **gerir a sua equipa, editar o perfil da empresa, publicar novas ofertas, acompanhar candidatos e visualizar métricas detalhadas com gráficos**. Além disso, foi implementado um **chat interno em tempo real**, utilizando a **Firebase Realtime Database**, para facilitar a comunicação entre membros da mesma empresa.

Ambos os projetos estão alojados na plataforma **Vercel**, que oferece integração direta com o GitHub. O projeto principal está publicado no domínio <https://jethire.pt>, e a *dashboard* encontra-se acessível através do subdomínio <https://dashboard.jethire.pt>. Esta separação permite uma maior organização, segurança e controlo de permissões entre diferentes tipos de utilizadores.

Durante o desenvolvimento da Jet Hire, organizei o meu trabalho em **três fases principais**:

Planeamento – fiz uma análise detalhada dos requisitos, defini os tipos de utilizadores e funcionalidades essenciais, e desenhei a estrutura geral da aplicação com base nas tecnologias que pretendia usar.

Desenvolvimento – comecei a construir cada funcionalidade, separando bem os componentes e os contextos entre a aplicação pública e a *dashboard*. Fui validando cada parte com testes e melhorando o desempenho.

Ajustes finais e publicação – finalizei os detalhes visuais, reforcei a segurança, publiquei os projetos com domínios próprios e tratei da documentação para o relatório.

A Jet Hire representa não só uma **resposta prática a um problema real**, como também o **culminar do meu percurso formativo**. É um projeto que me obrigou a **pensar como utilizador e como programador, a tomar decisões técnicas com base em objetivos reais**, e a aplicar tudo o que aprendi de forma sólida, funcional e útil.

Acredito que esta aplicação pode servir de base para algo maior no futuro, e que reflete a dedicação, o esforço e a evolução que tive ao longo do curso.

Introduction (Inglês)

For my **Final Professional Aptitude Project (PAP)**, I developed **Jet Hire**, a web platform designed to **connect job seekers with companies** in the field of **Information Technology (IT)**. The idea for this project came from a personal experience I had during the summer holidays. While looking for a temporary job to save some money, I realized how difficult it can be to find job opportunities quickly and efficiently. Many vacancies weren't available online, and those that were were scattered across different platforms, making the process confusing and time-consuming. That's when I saw the need for a more centralized and modern solution.

With this in mind, I decided to create an application that would allow **job offers to be presented in a clear, accessible and modern way**, helping both those looking for work and companies that need to recruit. The focus on the **IT sector** came naturally, as it's the area I'm studying and the one I want to pursue professionally. Throughout the course, I've worked with various web development tools, and this project was the perfect opportunity to apply that knowledge in a **practical, autonomous and well-structured way**.

Jet Hire is divided into **two separate applications**, each with a specific purpose:

The **main public-facing website** (Jet Hire), where anyone can **search for jobs, apply filters, create and edit their professional profile, save favourite job offers, and even interact with a virtual assistant** that helps answer questions about the platform and the tech field in general. This application was built using **Next.js 12**, written in **JavaScript**, using the classic **pages/ folder structure**, and connected to a **MongoDB** database using **Mongoose**.

The **dashboard**, reserved for authenticated users (companies and administrators), was developed independently using the latest version of the framework: **Next.js 15**, with **App Router, TypeScript, TailwindCSS**, and a modern, scalable architecture. The dashboard allows companies to **manage their teams, edit their company profile, publish job offers, review candidates, and view detailed metrics through graphs**. Additionally, I implemented a **real-time internal chat system** using **Firebase Realtime Database**, to allow communication between team members inside the same company.

Both projects are hosted on **Vercel**, a platform that offers seamless integration with GitHub. The main website is deployed at <https://jethire.pt>, while the dashboard is hosted on the subdomain <https://dashboard.jethire.pt>. This separation ensures better

organization, security, and user permission control between the public and private parts of the system.

During development, I structured my work into **three main phases**:

Planning – I analyzed all the requirements, defined the different types of users, outlined the key features, and chose the technologies that best fit the goals of the platform.

Development – I started implementing the core functionality, building each component and keeping the logic clearly separated between the public app and the dashboard. I continuously tested and improved performance throughout this stage.

Final adjustments and deployment – I polished the visual details, improved security, deployed both projects to their custom domains, and wrote all the documentation for the report.

Jet Hire represents not only a **practical solution to a real problem**, but also the **culmination of my learning journey**. It's a project that forced me to **think like both a user and a developer**, make **technical decisions with real goals in mind**, and apply everything I've learned in a consistent and useful way. I believe this platform can grow into something bigger in the future and that it reflects the **dedication, effort, and growth** I've experienced throughout this course.

Planeamento

Fundamentação

Decidi desenvolver a Jet Hire com base numa experiência pessoal que vivi durante as férias de verão. O meu objetivo era arranjar um trabalho para conseguir juntar dinheiro e comprar um portátil. No entanto, apesar de procurar em várias entidades, acabei por não conseguir nenhuma oportunidade.

Cheguei a pedir ajuda a amigos e conhecidos, que me indicaram outras instituições, mas sem sucesso. Com isso, percebi que, muitas vezes, **não basta querer trabalhar — é preciso estar no sítio certo, à hora certa**, e nem sempre temos acesso à informação necessária no momento certo.

Foi a partir dessa dificuldade que surgiu a ideia da Jet Hire — uma **plataforma web** pensada para facilitar o contacto entre empresas e candidatos, **centralizando as oportunidades de emprego** num só local.

A aplicação permite que empresas publiquem ofertas de forma prática, e que qualquer pessoa possa encontrá-las facilmente, sem necessidade de deslocações. Uma funcionalidade adicional importante é a inclusão de vagas para **trabalhos de verão destinados a maiores de 16 anos**, permitindo candidaturas por menores com o respetivo **termo de responsabilidade parental**.

Este projeto não tem apenas como objetivo resolver um problema real, mas também dar-me a oportunidade de **aplicar os conhecimentos adquiridos** ao longo do curso. Com a Jet Hire, pude trabalhar com tecnologias modernas como **React, Next.js** e **MongoDB**, aprofundando assim as minhas competências e criando algo com impacto prático.

Tabela 1 - Tabela de objetivos

Objetivos	
Gerais	Específicos
Terminar o Curso	<ul style="list-style-type: none">• Ter boa nota na PAP;• Terminar o curso com uma boa média.
Desenvolvimento da PAP	<ul style="list-style-type: none">• Aumentar os meus conhecimentos em Programação;• Melhorar as minhas competências de concretização de projetos;

- | | |
|--|---|
| | <ul style="list-style-type: none">• Adquirir conhecimentos em Next.js, Tailwind e MongoDB• Aplicar os conhecimentos adquiridos ao longo dos três anos do curso.• Explorar novas tecnologias utilizadas para o desenvolvimento da PAP.• Colocar o site online para que seja visível por muitos utilizadores.• Desenvolver competências na área de desenvolvimento de projetos de software. |
|--|---|

Recursos

Hardware

Desktop pessoal

- AMD Ryzen 5 7600X @ 4.7Ghz
- RAM 32Gb DDR5 5600Mhz
- Radeon RX 6700 10Gb
- Disco SSD 512Gb
- Disco HDD 2Tb
- Linux Ubuntu 22.04.1

Portátil pessoal

- Apple Silicon M3 Chip
- RAM 16Gb
- Disco SSD 512Gb
- macOS Sequoia 15.01

Software

- Microsoft Office 2021 Pro Plus

Requisitos Mínimos

- Sistema Operativo – Windows 10+ ou macOS 11.6+
- Processador – 1.6 GHz ou superior
- Memória RAM – 4Gb (64 bits)
- Disco – 4Gb
- Resolução Mínima – 1280px X 768px

- Visual Studio Code

Requisitos Mínimos

- Processador – 1.6 GHz ou superior
- Memória RAM – 1Gb

- Github Desktop

Requisitos Mínimos

- Sistema Operativo – Windows 10+ (64 bits) ou macOS 10.5+

- Inkscape

Requisitos Mínimos

- Processador – 1 GHz ou superior
- Memória RAM – 256Mb

- Google Chrome

Requisitos Mínimos

- Sistema Operativo – Windows 7+ ou macOS 10.9+
- Processador – 2 GHz ou superior
- Memória RAM – 2Gb

- Krita

Requisitos Mínimos

- Processador – 1.6 GHz ou superior
- Memória RAM – 4Gb
- Disco – 1Gb

- MongoDB Compass

Requisitos Mínimos

- Sistema Operativo – Windows 10+ ou macOS 11+ ou Ubuntu 16.04+

- Postman

Requisitos Mínimos

- Sistema Operativo – Sistema operativo 64-bits

Metodologia

Para desenvolver a Jet Hire optei por usar a metodologia *Kanban*. Esta metodologia é muito visual e facilita o controlo da gestão de tarefas tanto como garante um fluxo contínuo do desenvolvimento.

A metodologia *Kanban* consiste na utilização de um quadro de tarefas para monitorizar o fluxo de trabalho. As tarefas são categorizadas e assim distribuídas respetivamente.

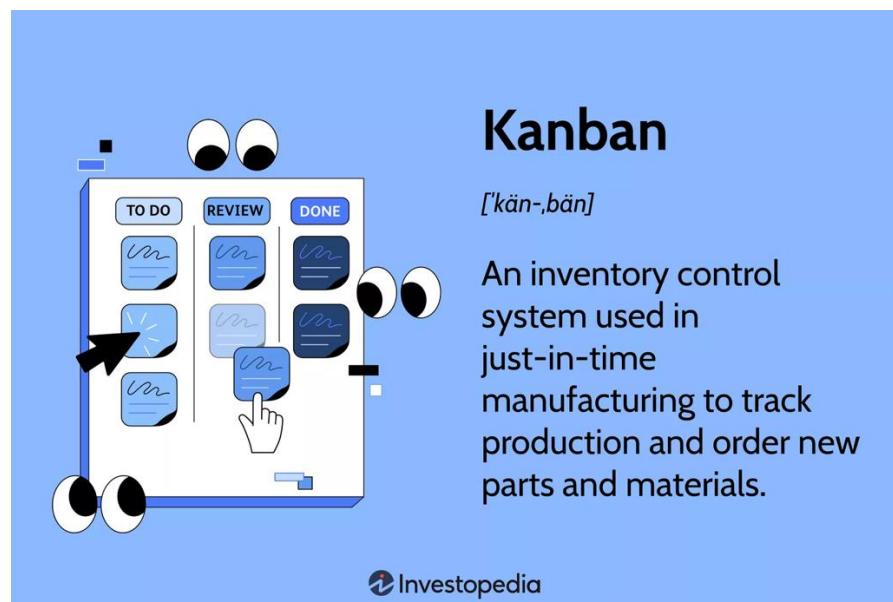


Figura 1 - Metodologia Kanban

Fonte: <https://www.investopedia.com/terms/k/kanban.asp>

Escolhi esta metodologia porque ajuda a adaptar o trabalho conforme as necessidades mudam e dá uma visão clara do que se está a fazer, do que já se fez e do que se vai fazer. Como os requisitos da aplicação podem evoluir ao longo do tempo, o *Kanban* permite fazer estes ajustes de maneira fácil. Além disso, esta metodologia ajuda a identificar e resolver problemas rapidamente, garantindo que o projeto avança de forma organizada e eficiente.

Atividades

A Prova de Aptidão Profissional irá ser desenvolvida em três fases. Cada fase é constituída por várias atividades. A primeira fase, a Fase de Definição tem como principais atividades: pesquisa bibliográfica, análise do sistema, análise dos requisitos e planeamento do projeto. A segunda fase, a Fase de Desenvolvimento é constituída pelas seguintes atividades, desenho, codificação e testes. A terceira e última fase, a Fase de Manutenção é onde encontramos as atividades de correção, adaptação e evolução.

Fase de Definição

- ▶ Pesquisa Bibliográfica – etapa inicial do trabalho com o objetivo de reunir todas as informações necessárias para o desenvolvimento do projeto.
- ▶ Análise do Sistema – atividade onde se realizou o levantamento de todas as funcionalidades do projeto e a forma como elas vão funcionar. Nesta atividade foi também realizada a escolha das tecnologias para a realização do mesmo.
- ▶ Análise dos Requisitos – nesta atividade foi necessário fazer a verificação dos requisitos mínimos do Software necessário para implementação do projeto.
- ▶ Planeamento do Projeto – com esta atividade foi desenvolvido o cronograma em que se estabelece as balizas temporais para cada uma das fases do projeto.

Fase de Desenvolvimento

- ▶ Desenho – nesta atividade faz-se o levantamento dos dados necessários para criação da Base de Dados utilizada no projeto, implementando o Diagrama Entidade Relacionamento e o Modelo de Dados. Faz-se ainda a estruturação da navegação do website e do backoffice.
- ▶ Codificação – atividade em que toda a parte de codificação é realizada.
- ▶ Testes – ao longo desta atividade realiza-se os testes às funcionalidades implementadas por forma a garantir que nesta fase as mesmas estão a funcionar de acordo com o que foi planeado.

Fase de Manutenção

- ▶ Correção – é nesta atividade em que irei realizar correções ao projeto. Estas correções tanto podem ser à interface gráfica ou ao código realizado até ao momento. Se for necessário também se poderá fazer alterações estruturais à base de dados.
- ▶ Adaptação – depois de executado todos os testes e correções, é nesta atividade que vão ser realizadas adaptações ao projeto. Estas adaptações podem ser ao nível da interface gráfica ou ao nível da otimização de código. Estas adaptações surgem da necessidade do projeto ser testado por outras pessoas.
- ▶ Evolução – depois de ter os objetivos iniciais concluídos, é nesta fase, havendo tempo que irei tentar implementar novas funcionalidades que foram surgindo ao longo do desenvolvimento do projeto e que inicialmente não estavam previstas.

Cronograma

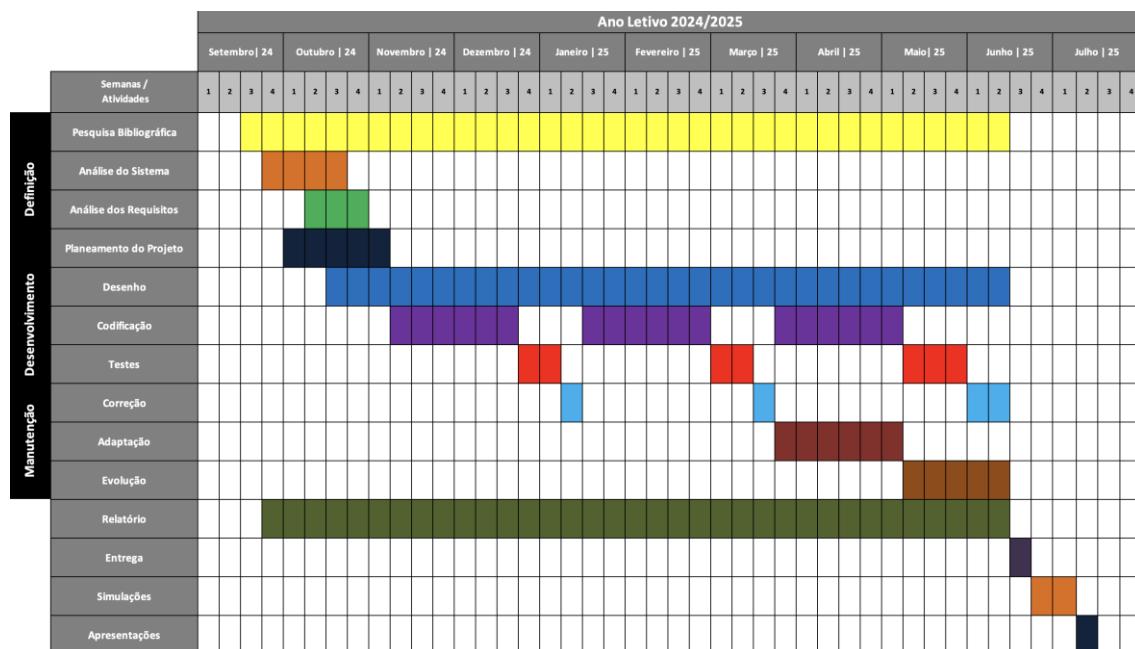


Figura 2 - Cronograma

Instalações

Durante a preparação do ambiente de desenvolvimento, foi necessário instalar várias ferramentas essenciais. Cada uma delas foi configurada no sistema Ubuntu / MacOS, através do terminal ou de instaladores oficiais. Abaixo, apresenta-se o processo de instalação de cada uma.

Visual Studio Code

Para começar a instalação do Visual Studio Code é preciso fazer *download* do instalador através do site oficial.

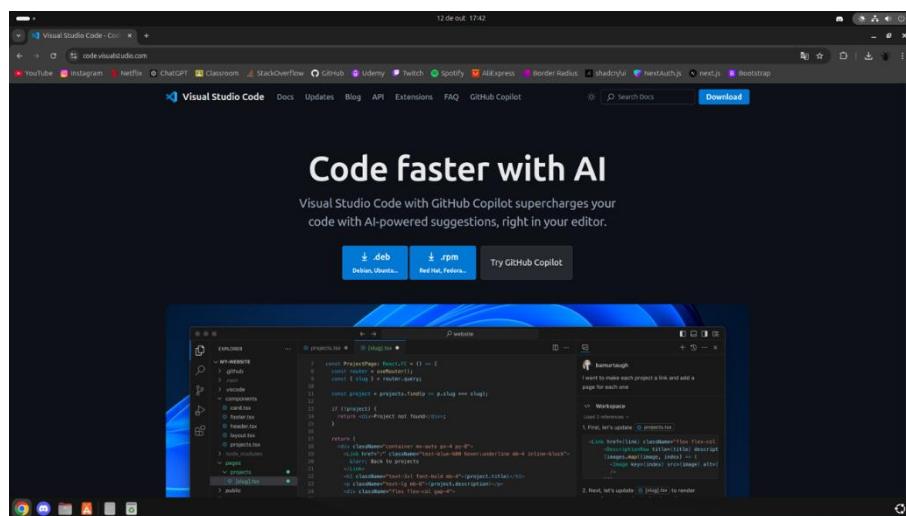


Figura 3 - Instalação Visual Studio Code

Depois de executar o instalador e a instalação estiver pronta, a página inicial abre.

Para personalizar o Visual Studio Code a meu gosto e de acordo com as minhas necessidades adicionei as seguintes extensões:

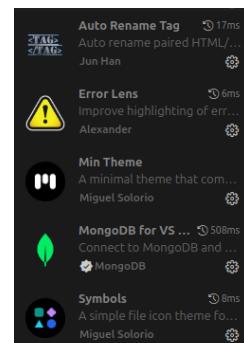


Figura 4 - Extensões

MongoDB Compass

Tal como o VS Code é necessário fazer *download* do instalador através do site oficial.

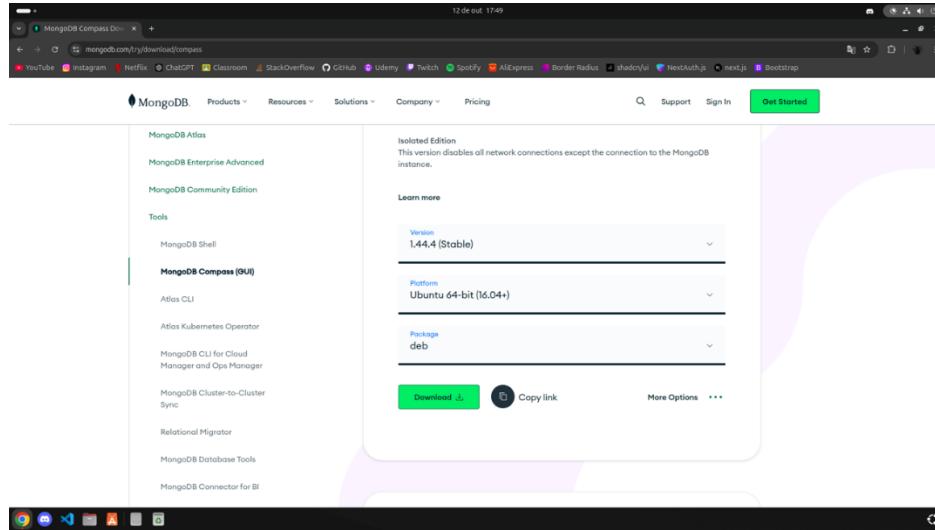


Figura 5 - Instalação MongoDB Compass: Passo 1

De seguida, é só executar o instalador.

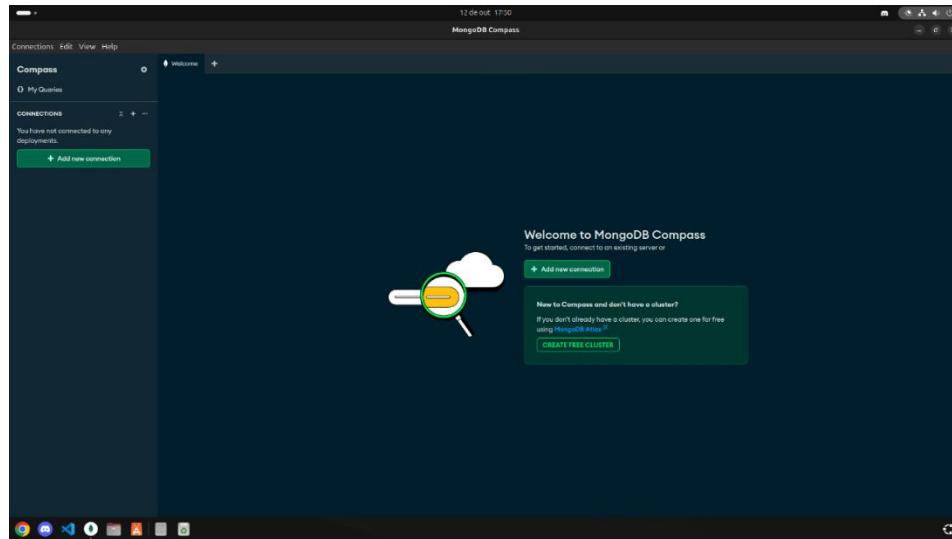
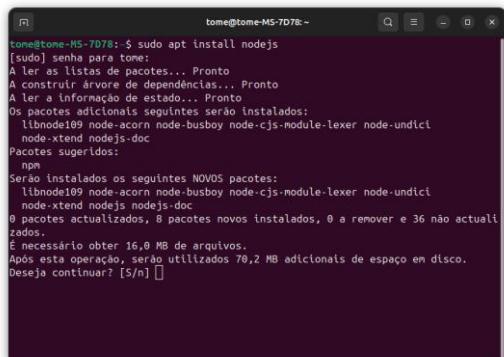


Figura 6 - Instalação MongoDB Compass: Passo 2

Node.js (node / npm)

No terminal do Ubuntu apenas é preciso utilizar os seguintes comandos para instalar o *node* e o *npm*:

sudo apt install nodejs



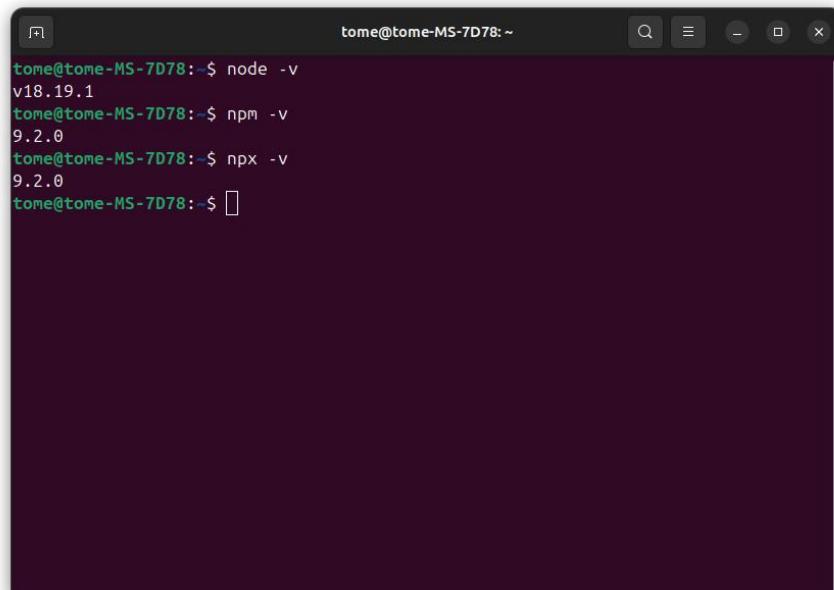
```
tome@tome-MS-7D78:~$ sudo apt install nodejs
[sudo] senha para tome:
A ler as listas de pacotes... Pronto
A construir árvore de dependências... Pronto
A ler a informação de estado... Pronto
Os pacotes adicionais seguintes serão instalados:
  libnode109 node-acorn node-busboy node-cjs-module-lexer node-undici
  node-xend nodejs-doc
Pacotes sugeridos:
  npm
Serão instalados os seguintes NOVOS pacotes:
  libnode109 node-acorn node-busboy node-cjs-module-lexer node-undici
  node-xend nodejs-doc
0 pacotes actualizados, 8 pacotes novos instalados, 0 a remover e 36 não actualizados.
É necessário obter 16,0 MB de arquivos.
Após esta operação, serão utilizados 70,2 MB adicionais de espaço em disco.
Deseja continuar? [S/n] 
```

Figura 7 - *sudo apt install nodejs*

sudo apt install npm

Figura 8 - *sudo apt install npm*

Para verificar se foi instalado corretamente executei os comandos para verificar a respetiva versão:



```
tome@tome-MS-7D78:~$ node -v
v18.19.1
tome@tome-MS-7D78:~$ npm -v
9.2.0
tome@tome-MS-7D78:~$ npx -v
9.2.0
tome@tome-MS-7D78:~$ 
```

Figura 9 - Verificação do npm

Github Desktop

Para instalar o Github Desktop no Ubuntu é necessário fazer *download* através do terminal, visto que não existe instalador no site oficial. Para isso é preciso utilizar o seguinte comando:

```
 wget https://github.com/shiftkey/desktop/releases/download/release-3.1.7-
linux1/GitHubDesktop-linux-3.1.7-linux1.deb
```

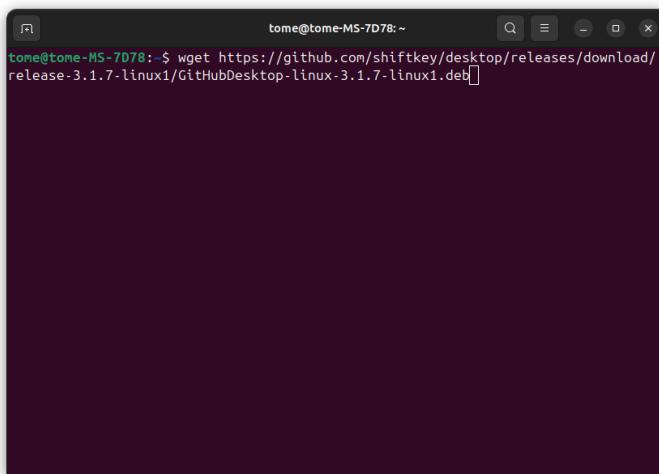


Figura 10 - Instalação GitHub Desktop: Passo 1

De seguida, é só executar o instalador com o comando:

```
 sudo apt install -f ./GitHubDesktop-linux-3.1.7-linux1.deb
```

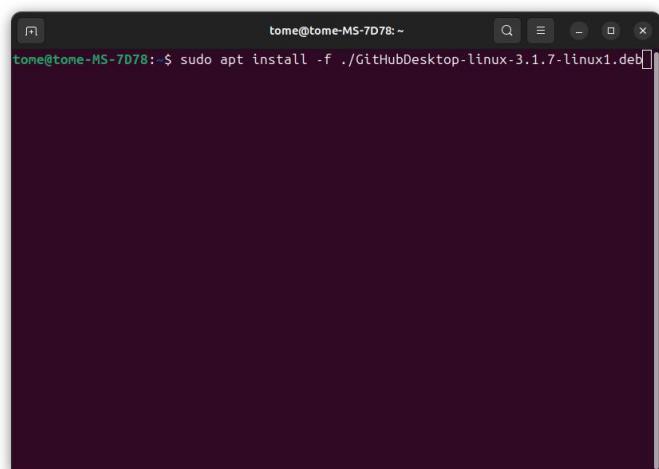


Figura 11 - Instalação GitHub Desktop: Passo 2

No fim da execução do instalador só falta abrir a aplicação.

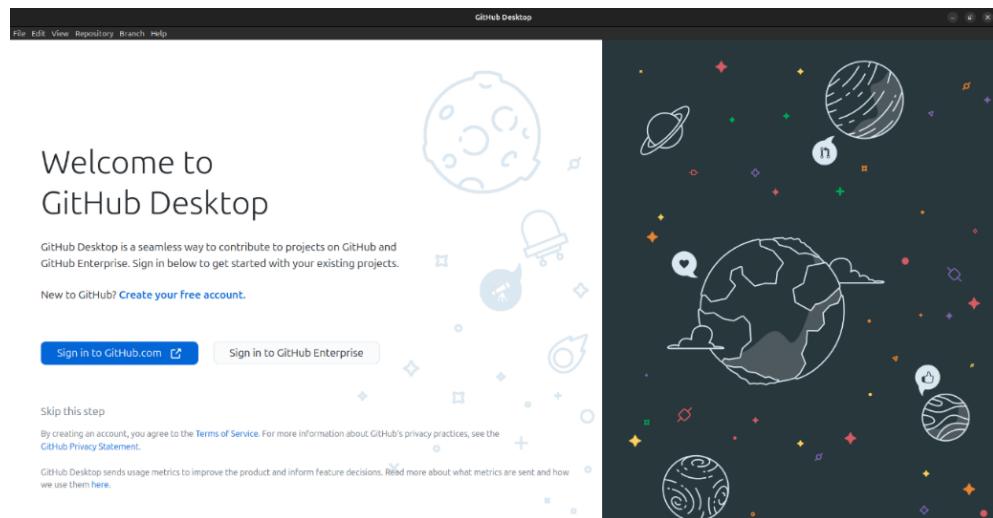


Figura 12 - Instalação GitHub Desktop: Passo 3

Inkscape

Para a instalação do Inkscape só foi preciso utilizar o comando `sudo apt install inkscape` no terminal.

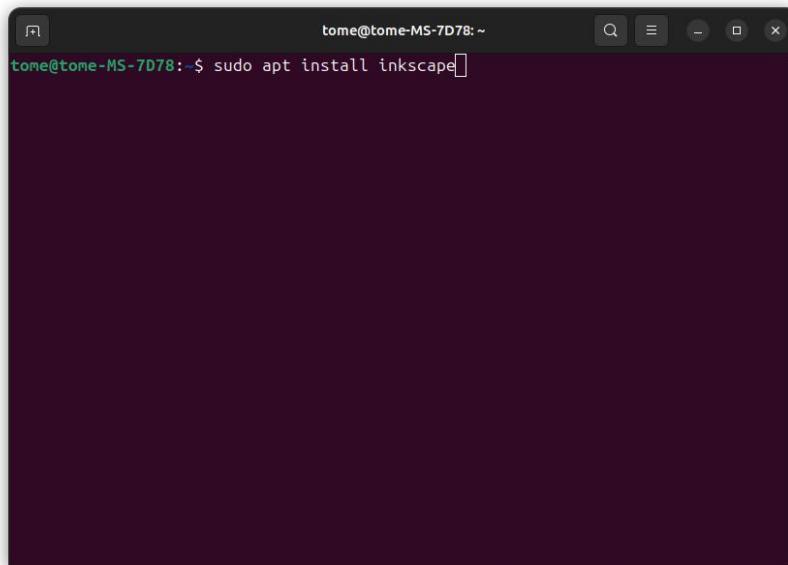


Figura 13 - Instalação do Inkscape: Passo 1

A instalação está concluída e pronta a ser utilizada.

Krita

Como o Krita está disponível no **Centro de Aplicações** do Ubuntu apenas foi preciso instalar através do mesmo.

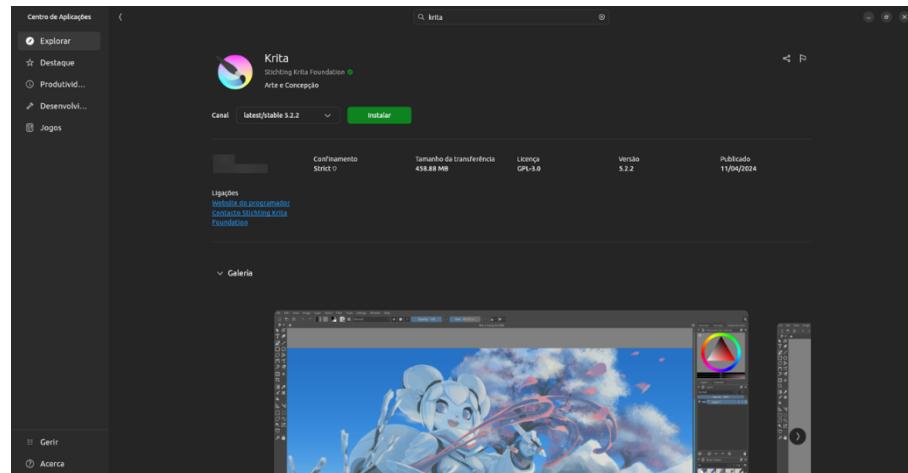


Figura 14 - Instalação do Krita: Passo 1

A aplicação está pronta a ser utilizada.

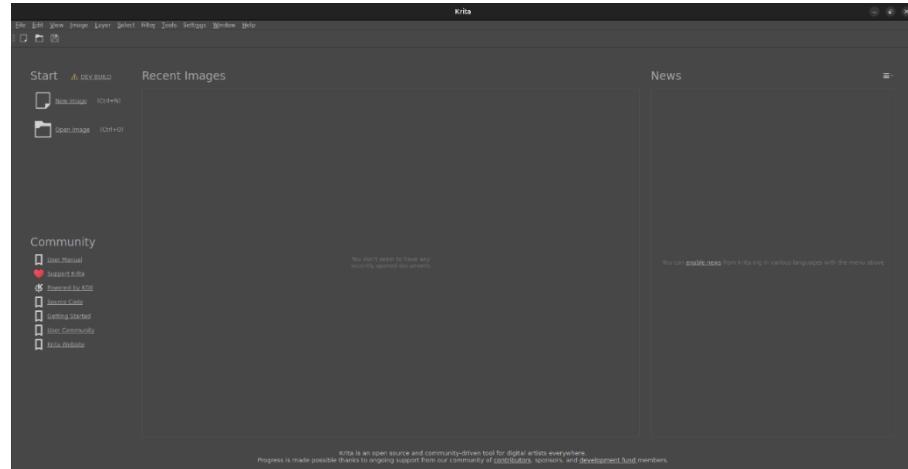
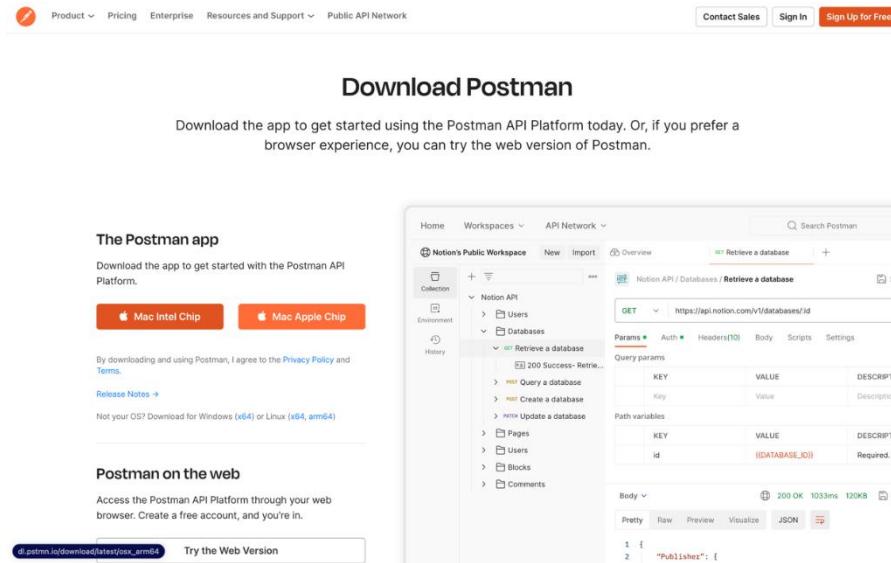


Figura 15 - Instalação do Krita: Passo 2

Postman

Para instalar o Postman, apenas é necessário transferir o executável do site oficial da empresa.



The screenshot shows the official Postman website. On the left, there's a section for "The Postman app" with download links for Mac Intel Chip and Mac Apple Chip. Below that is a "Postman on the web" section with a link to "Try the Web Version". On the right, the main content area displays the Postman application interface. It shows a sidebar with "Notion API" and "Databases" sections. A central panel shows a GET request to "https://api.notion.com/v1/databases/:id" with parameters like "Auth", "Headers", and "Body". At the bottom, there's a JSON response preview.

Figura 16 - Instalação do Postman: Parte 1

Após a instalação do executável, apenas é preciso criar conta e o Postman está pronto a ser utilizado.

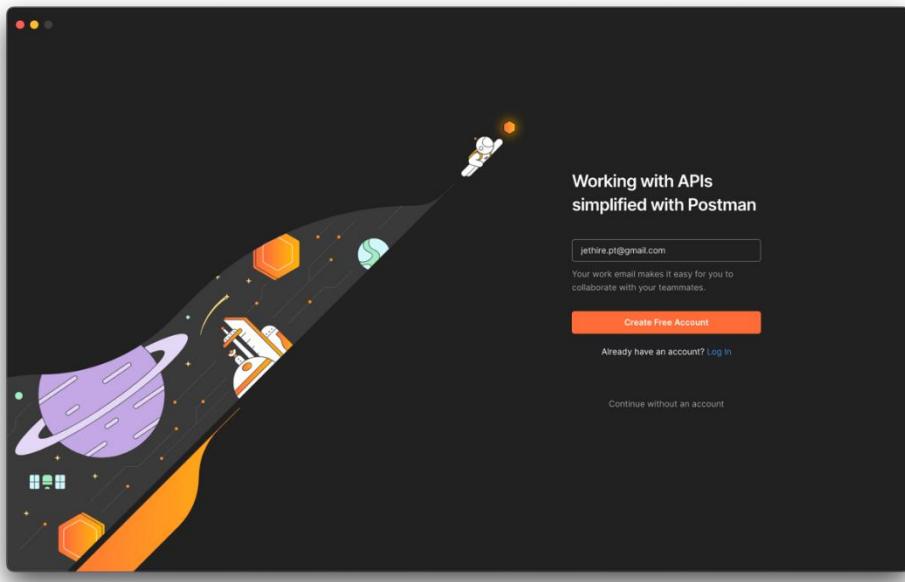


Figura 17 - Instalação do Postman: Parte 2

Sincronização entre dois computadores

Durante o desenvolvimento do projeto, utilizei **dois computadores diferentes**. Para garantir que todos os ficheiros estivessem sempre atualizados em ambos, recorri ao **GitHub Desktop** como ferramenta de sincronização.

Criação do repositório

O primeiro passo consistiu em **criar um repositório local** onde seriam guardados todos os ficheiros do projeto.

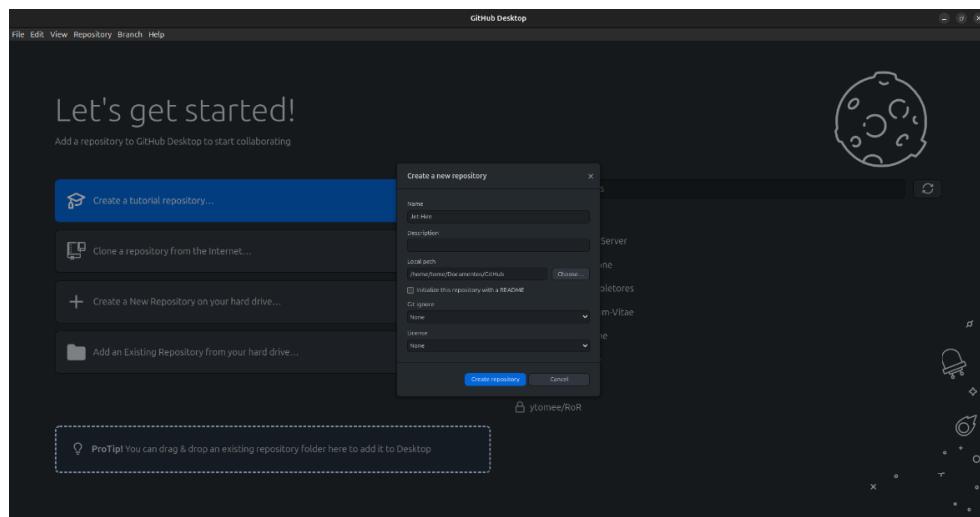


Figura 18 - Sincronização entre dois computadores: Criação do repositório

Depois da criação do repositório é preciso publicá-lo:

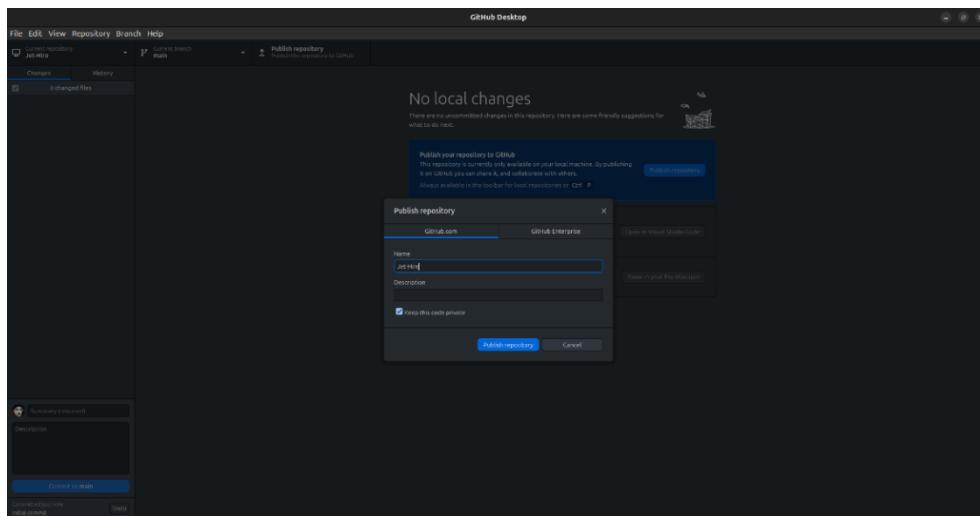


Figura 19 - Sincronização entre dois computadores: Publicação do repositório

Push

O GitHub Desktop cria automaticamente uma **pasta local** dentro da pasta *Documentos*. Basta **mover os ficheiros do projeto** para essa pasta e abrir novamente a aplicação.

De seguida:

- são exibidas as alterações detetadas;
- dá-se **commit** com uma mensagem descriptiva;
- por fim, faz-se **push** para enviar as alterações para o GitHub.

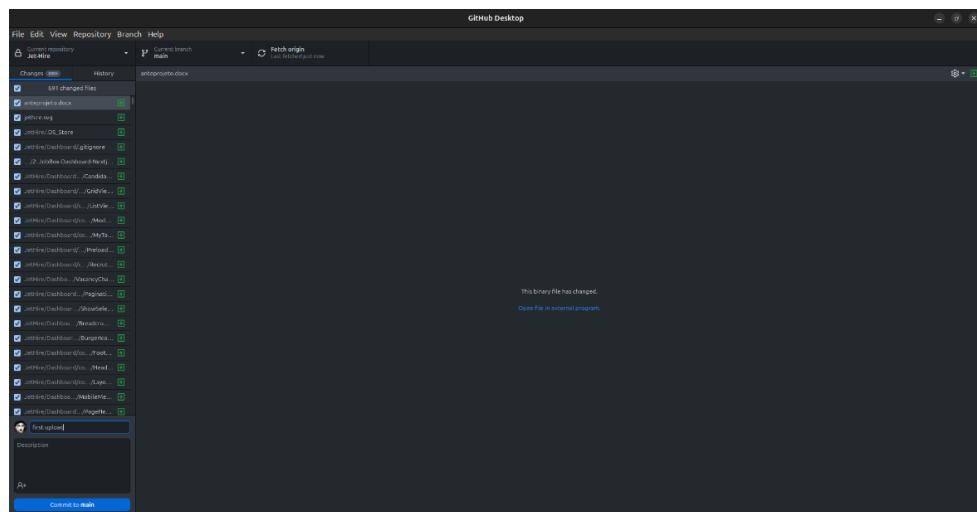


Figura 20 - Sincronização entre dois computadores: Commit

Depois de dar *commit*, é preciso dar *push*, ou seja, o pedido final para a alteração dos dados na *cloud*:

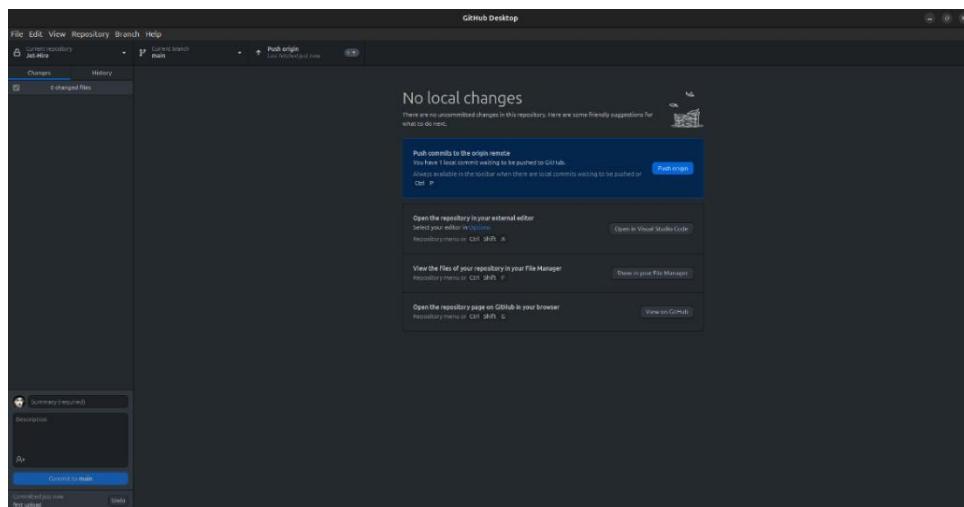


Figura 21 - Sincronização entre dois computadores: Push

Para confirmar se o repositório foi criado corretamente basta aceder pelo browser:

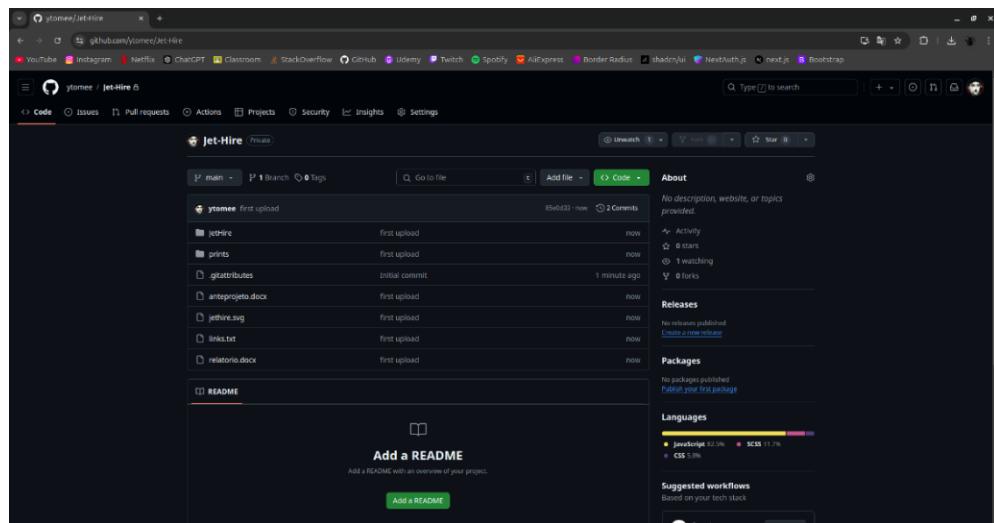


Figura 22 - Sincronização entre dois computadores: Verificação do repositório

Pull

Se for necessário continuar o trabalho num segundo computador, existem duas possibilidades:

- Clonar o repositório (primeira vez):

Se ainda não existir localmente, deve-se **clonar o repositório a partir do GitHub**.

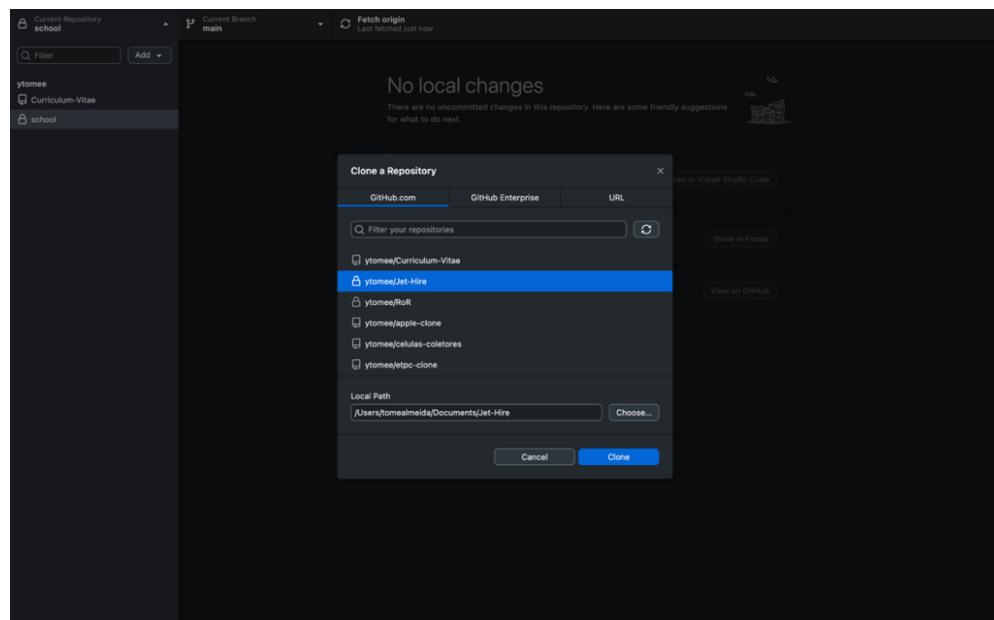


Figura 23 - Sincronização entre dois computadores: Clonar um repositório

- Atualizar o **repositório** (já existente):

Se o repositório já estiver presente, basta:

- Fazer **fetch** para verificar se há alterações disponíveis
- Caso existam, aplicar **pull** para atualizar os ficheiros locais

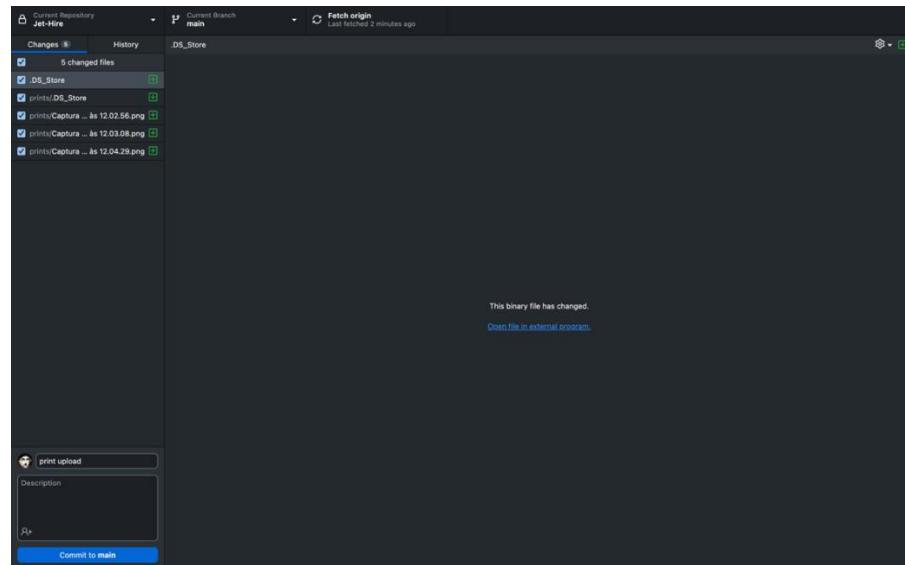


Figura 24 - Sincronização entre dois computadores: Fetch

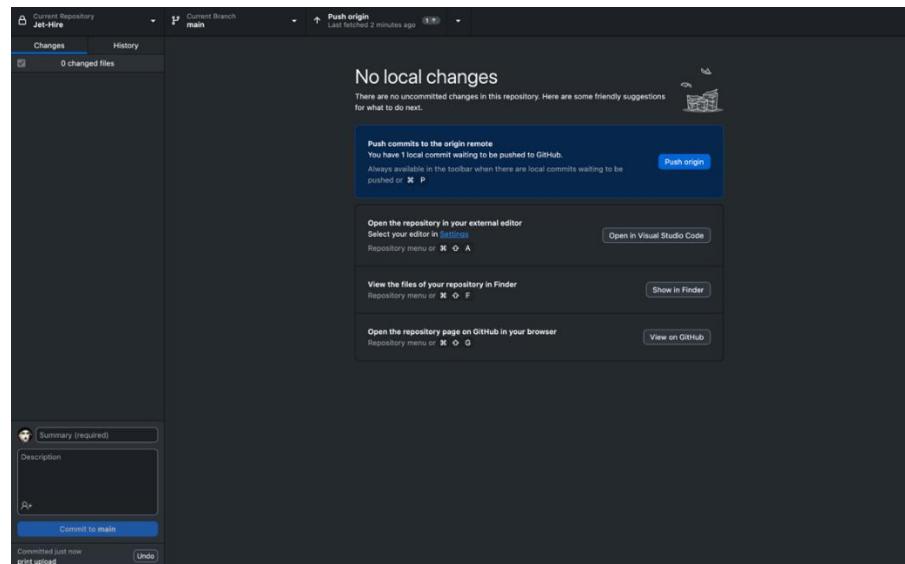


Figura 25 - Sincronização entre dois computadores: Pull

Este processo garante que todo o trabalho realizado em **ambos os computadores** esteja sempre sincronizado e seguro na *cloud*. O GitHub Desktop revelou-se uma ferramenta prática e intuitiva, ideal para gerir versões do projeto de forma distribuída.

Sitemap

Para o desenho do projeto decidi criar um *sitemap* na aplicação web FlowMapp.

Optei por não criar um modelo DER visto que não vou utilizar uma base de dados relacional.

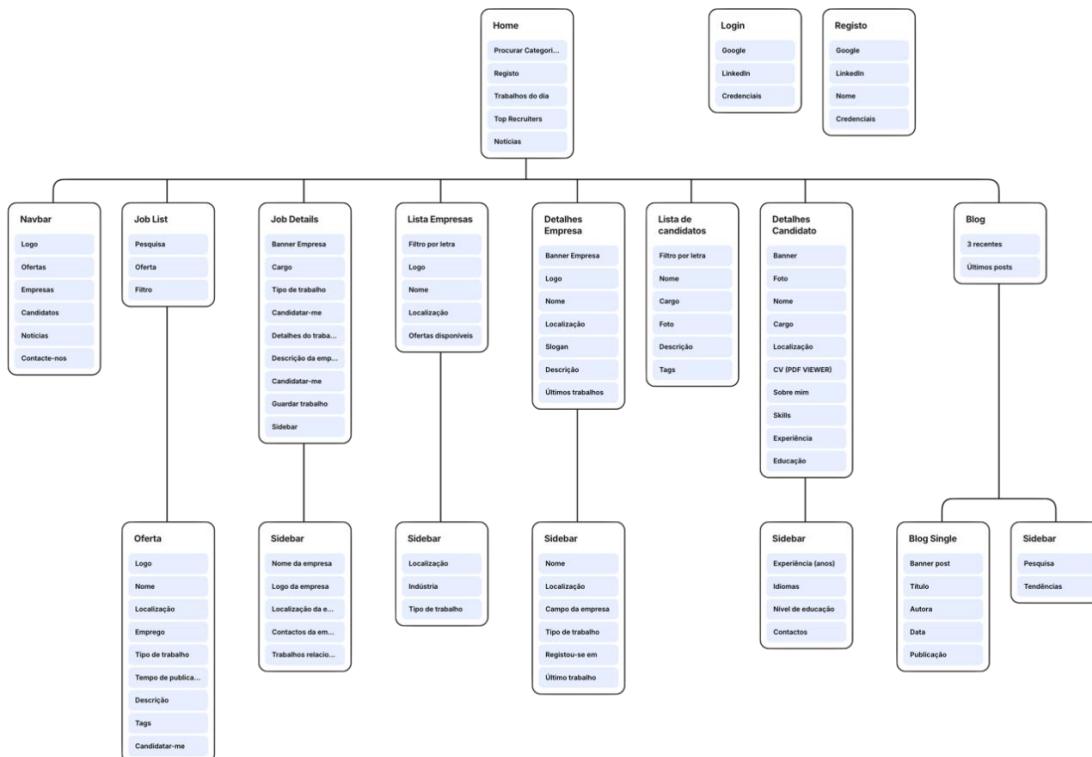


Figura 26 – Sitemap principal

Projeto

Elementos gerais

Download e preparação do *template*

Após a escolha do **template base**, foi feito o **download** dos ficheiros necessários.

O projeto foi então **aberto no Visual Studio Code**, onde se iniciou um terminal integrado e se acedeu à pasta do projeto com o comando:

```
cd [nome-da-pasta]
```

Instalação de dependências

Dentro da pasta, foi executado o comando:

```
npm install
```

Este comando é responsável por instalar todos os **módulos Node (node_modules)** e gerar automaticamente o ficheiro **package-lock.json**, necessário para garantir que todas as dependências estão corretamente sincronizadas com a versão do projeto.

Execução do projeto

Com a instalação concluída, o projeto ficou pronto a ser executado com o comando:

```
npm run dev
```

Este comando inicia o servidor de desenvolvimento. Ao aceder ao **link apresentado no terminal**, é possível visualizar o projeto a correr **em tempo real no browser**.

Publicação online com domínio personalizado

A aplicação Jet Hire foi dividida em **duas partes** distintas: a **página principal** (voltada para o público) e a **dashboard** (acesso restrito para empresas e administradores). Para garantir uma separação clara entre estas duas áreas, publiquei cada projeto de forma **independente**, utilizando a plataforma **Vercel**.

Deploy automático no Github e Vercel

O primeiro passo foi **ligar os dois repositórios GitHub à Vercel**:

- **jethire**: contém o código da página principal;
- **dashboard-jethire**: contém o código da *dashboard*.

A Vercel deteta automaticamente que se trata de um projeto Next.js e trata da **build, otimização e alojamento** sem necessidade de configuração manual.

Cada *push* feito no GitHub é automaticamente atualizado na Vercel, garantindo assim uma integração contínua.

Ligaçāo a domínios personalizados

Com os projetos já publicados na Vercel, associe os seguintes domínios:

- Para a página principal:
 - Foi utilizado o domínio principal **jethire.pt**
 - A ligação foi feita através da **zona DNS do fornecedor do domínio**, onde apontei os registo A e CNAME para os servidores da Vercel
- Para a dashboard:
 - Foi criado um **subdomínio personalizado**: **dashboard.jethire.pt**
 - Também aqui foram adicionados os registo DNS necessários no painel de controlo do domínio principal
 - Na Vercel, configurei o subdomínio para apontar para o projeto dashboard-jethire

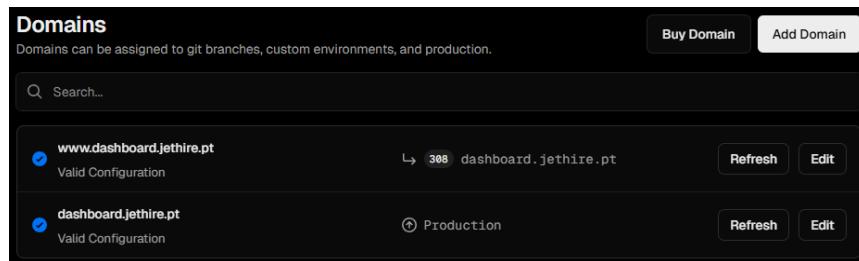


Figura 27 - Domínios da dashboard

Certificados SSL e HTTPS

A Vercel gera automaticamente os **certificados SSL**, garantindo que ambos os domínios usam **HTTPS**, sem configurações adicionais. Isto assegura que a navegação é **segura e confiável** para todos os utilizadores.

Vantagens deste processo

- separação clara entre **parte pública e área privada**;
- possibilidade de gerir permissões de forma isolada;
- estrutura escalável e profissional;
- domínios personalizados que reforçam a identidade do projeto.

Considerações legais e políticas da plataforma

Como uma plataforma que promove a ligação entre candidatos e empresas, a Jet Hire tem o dever de assegurar não só a proteção dos dados pessoais dos seus utilizadores, mas também o cumprimento de boas práticas legais e de transparência digital.

Para isso, foram desenvolvidas três páginas específicas: os **Termos e Condições**, a **Política de Privacidade** e a **Política de Cookies**, acessíveis no rodapé do website.

Estas páginas foram redigidas com uma abordagem clara, formal e informativa, com o objetivo de garantir que todos os utilizadores compreendem os seus direitos, responsabilidades e o modo como a sua informação é tratada.

Termos e Condições

A página de Termos e Condições define as regras de utilização da plataforma. É explicado que o registo implica a **aceitação integral** destes termos, sendo responsabilidade do **utilizador** fornecer dados verdadeiros, manter a confidencialidade da sua conta e utilizar a plataforma de forma ética e legal.

As empresas são responsáveis pelas ofertas publicadas, e é proibida a utilização da plataforma para fins ilícitos, enganosos ou fraudulentos. A secção reforça ainda que todos os conteúdos da Jet Hire (texto, imagem, código, etc.) são propriedade da plataforma, não podendo ser copiados sem autorização. Está também prevista a possibilidade de atualização destes termos a qualquer momento.

 [Procurar trabalho](#) [Empresas](#) [Candidatos](#) [Jet AI](#) [Sobre nós](#) Tomé Almeida 

Termos e Condições

Ao aceder e utilizar esta plataforma ([jet-hire.pt](#) – [disponível em `https://pt` e `http://www.jethire.pt`](#)) – o utilizador aceita integralmente os termos e condições aqui descritos. Caso não concorde com algum dos pontos, deverá cessar imediatamente a utilização da plataforma.

A Jet Hire funciona como um intermediário digital entre candidatos à procura de oportunidades e empresas que procuram novos talentos. Envolve-se na elaboração e na facilitar a criação de forma eficaz, não garantindo qualquer tipo de colocação profissional ou segurança no sucesso dos processos de recrutamento realizados através da plataforma.

O utilizador prescinde o fornecimento de dados reais, completos e atualizados. As contas criadas são pessoais, confidenciais e intransmissíveis. A utilização indevida, fraudeulenta ou com fins ilícitos poderá resultar na suspensão ou remoção da conta, sem aviso prévio.

Toda a informação publicada pelas empresas, incluindo ofertas de trabalho, é da exclusiva responsabilidade das mesmas. A Jet Hire não se responsabiliza pelo conteúdo dessas ofertas nem por eventuais informações incorrectas, desactualizadas ou enganosas publicadas por terceiros.

O utilizador compromete-se a utilizar a plataforma de forma ética, legal e respeitadora dos direitos de terceiros. É expressamente proibida a publicação de conteúdos ofensivos, discriminatórios, ofensivos ou enganosos.

Todos os elementos presentes na plataforma – incluindo logótipos, textos, gráficos, funcionalidades, imagens e código-fonte – são propriedade da Jet Hire ou encontram-se licenciados para uso exclusivo da mesma. É proibida qualquer reprodução, distribuição ou utilização desses conteúdos sem autorização expressa e por escrito.

A Jet Hire reserva-se o direito de atualizar os presentes Termos e Condições sempre que necessário. Qualquer modificação entra em vigor a partir do momento em que é publicada, sendo da responsabilidade do utilizador consultar regularmente esta página.

Para quaisquer questões, sugestões ou esclarecimentos adicionais, poderá contactar a nossa equipa através do endereço de email suporte@jethire.pt.

Jet Hire, Junho 2025

Copyright © 2025. Desenvolvido por Tomé Almeida  [Política de privacidade](#) [Termos & Condições](#) [Política de cookies](#)

Figura 28 - Termos e Condições

Política de Privacidade

A Jet Hire recolhe, trata e protege os dados dos seus utilizadores de acordo com o Regulamento Geral sobre a Proteção de Dados (RGPD). Esta página detalha quais os dados recolhidos — como nome, email, localização, CVs e informação técnica (como IP e tipo de navegador) — e como são utilizados: para personalizar a experiência, melhorar a segurança da aplicação e facilitar a comunicação entre candidatos e empresas. Estão também definidos os **direitos dos utilizadores**, como o acesso, retificação, eliminação ou portabilidade dos dados, e os meios pelos quais estes direitos podem ser exercidos (ex: contacto por email).

 [Procurar trabalho](#) [Empresas](#) [Candidatos](#) [Jet AI](#) [Sobre nós](#) Tomé Almeida 

Política de Privacidade

A Jet Hire respeita a privacidade dos seus utilizadores e está empenhada em proteger os dados pessoais que recolhe e trata, em total conformidade com o Regulamento Geral sobre a Proteção de Dados (RGPD).

No âmbito da utilização da plataforma, recolhemos informações como o nome, endereço de email, localização, bem como dados facultados no perfil, como experiência profissional, currículo, foto ou candidaturas submetidas. Também podem ser recolhidos dados técnicos relativos ao dispositivo e navegador, como o endereço IP e o sistema operativo.

Entre destes, os utilizadores fornecem o funcionamento eficiente e seguro da plataforma, bem como para personalizar a experiência de cada utilizador. Permitimos, por exemplo, que candidatos encontrem ofertas relevantes e que empresas identifiquem os perfis mais adequados às suas necessidades. Os dados são também usados para melhorar continuamente os nossos serviços e responder questões relacionadas com suporte ou segurança.

A Jet Hire compromete-se a não partilhar os seus dados com terceiros sem o seu consentimento, exceto quando tal for necessário para o cumprimento de obrigações legais ou para a execução dos serviços prestados, como no caso de comunicações entre empresas e candidatos incluídas pela própria candidatura.

Todos os dados são armazenados de forma segura e acessível apenas a entidades autorizadas. Tomamos medidas técnicas e organizativas para garantir a confidencialidade e integridade da informação tratada.

Nos termos da legislação aplicável, o utilizador tem o direito de aceder aos seus dados, solicitar a sua retificação ou eliminação, opor-se ao tratamento ou pedir a portabilidade dos mesmos. Estes direitos podem ser exercidos a qualquer momento, mediante pedido enviado para o endereço de email suporte@jethire.pt.

Esta política poderá ser actualizada periodicamente, sendo todas as alterações comunicadas nesta página. Recomendamos a sua consulta regular.

Jet Hire, Junho 2025

Copyright © 2025. Desenvolvido por Tomé Almeida  [Política de privacidade](#) [Termos & Condições](#) [Política de cookies](#)

Figura 29 - Política de privacidade

Política de Cookies

Esta política explica o que são **cookies**, que tipos são utilizados na plataforma (essenciais, analíticos e funcionais) e como podem ser geridos pelos utilizadores. Os

cookies permitem melhorar a experiência de navegação, guardar preferências e analisar métricas de desempenho da aplicação.

É deixado claro que o utilizador pode desativar cookies no seu navegador, embora algumas funcionalidades possam deixar de funcionar corretamente. O tom da página mantém-se profissional e acessível, reforçando a transparência na utilização de dados técnicos.



The screenshot shows the header of the Jet Hire website with navigation links: Procurar trabalho, Empresas, Candidatos, Jet AI, Sobre nós, and Torna Almeida. Below the header is the title "Política de Cookies". A detailed text in Portuguese explains the use of cookies for improving user experience, tracking traffic, and personalizing content. It also describes the different types of cookies used, such as session cookies for site functionality and persistent cookies for user identification and preferences. It emphasizes that cookies are used exclusively for technical purposes and to improve the user's visit. The text also mentions that users can manage their cookie settings in their browser. At the bottom, there are links to the Privacy Policy, Terms & Conditions, and Cookie Policy, along with a small red circular icon.

Figura 30 - Política de cookies

Logótipo

O logótipo foi desenvolvido de forma a representar o conceito de uma **estrutura empresarial dinâmica**, destacando **quatro elementos modulares**. Para este projeto, um dos elementos é destacado individualmente — simbolizando o **novo membro** que se integra na equipa da empresa, tal como acontece no fluxo real da aplicação.

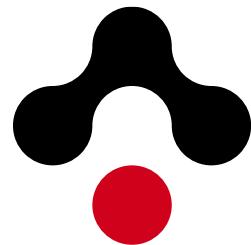


Figura 31 - Logótipo

Escolha da paleta de cores

Cor principal

A escolha da cor principal foi realizada com base na teoria de *design* de interfaces. Utilizou-se o modelo **HSB (Hue, Saturation, Brightness)** para gerar tons coerentes.

A cor principal escolhida foi o **vermelho**, por ser uma cor marcante, associada à ação e visibilidade.



Figura 32 - Cor principal

Definição de tons

Para criar uma paleta equilibrada, o processo consistiu em:

- Gerar o **tom mais claro** aumentando a **luminosidade** e reduzindo a **saturação**;

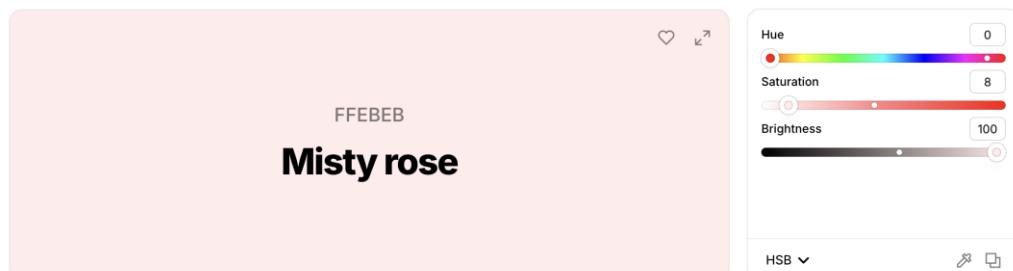


Figura 33 - Escolha de um tom claro

- Gerar o **tom mais escuro** diminuindo a **luminosidade** e aumentando a **saturação**.

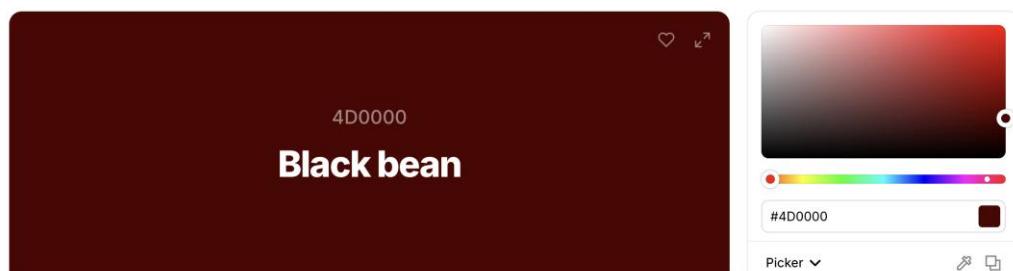


Figura 34 - Escolha de um tom escuro

Criação dos gradientes

Com os três tons definidos, foram gerados gradientes:

- **Gradiente claro:** do tom mais claro até à cor principal.



Figura 35 - Criação de um gradiente claro

- **Gradiente escuro:** da cor principal até ao tom mais escuro.

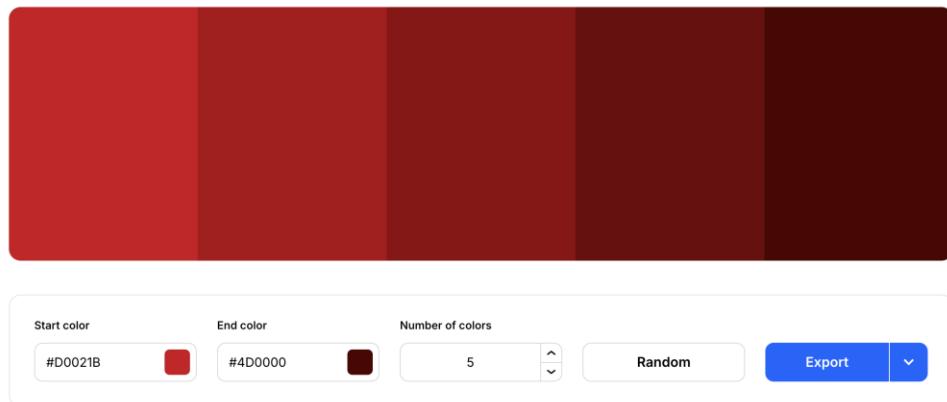


Figura 36 - Criação de um gradiente escuro

Cores secundárias

As cores secundárias foram definidas para representar diferentes **estados de notificação** no sistema:

Amarelo - avisos

Azul – informações

Verde - confirmações

Vermelho - erros



Figura 37 - Amarelo

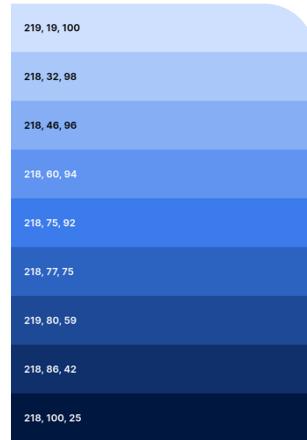


Figura 38 - Azul



Figura 39 - Verde

Cada uma destas cores passou pelo mesmo processo de **definição de tons e criação de gradientes**.

Cores neutras

A paleta de cores neutras acompanha a principal e é usada em fundos, textos e elementos complementares. Inclui tons que vão do **branco ao preto**, com variantes de cinzentos.

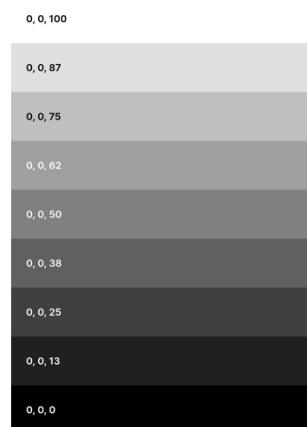


Figura 40 - Cores neutras

Implementação da paleta de cores no projeto

Para tornar estas cores acessíveis a todo o projeto, foram definidas como **variáveis globais** no CSS, dentro da pseudo-classe `:root`:

```
:root {  
    --red-900: #7b0000;  
    --red-500: #d40000;  
    --red-100: #ffb3b3;  
    ...  
}
```

Estas variáveis podem ser reutilizadas facilmente em qualquer parte do CSS:

```
a {  
    color: var(--red-900);  
}
```

Valorização do *design* e acessibilidade (UI & UX)

Desde o início do projeto, tive a preocupação de construir uma aplicação não só funcional, mas também visualmente agradável, coerente e adaptada a diferentes tipos de utilizadores.

Para isso, segui várias boas práticas de *design*, como:

- escolha de uma **paleta de cores contrastante**, com tons de vermelho e neutros, garantindo boa visibilidade tanto em modo claro como escuro;
- utilização de **ícones SVG personalizados** e minimalistas, que reforçam a identidade da plataforma;
- criação de **componentes modulares e responsivos**, que se adaptam a qualquer tipo de ecrã, mantendo sempre a legibilidade e a organização da informação;
- reorganização e simplificação do **layout de várias páginas** do *template* original, para melhorar a navegação e reduzir distrações;
- adição de animações suaves (com Framer Motion e Swiper) para melhorar a experiência sem prejudicar o desempenho;

- implementação de mensagens de estado visuais, como carregamentos, confirmações e alertas, que ajudam o utilizador a saber sempre o que está a acontecer.

Mesmo não sendo um projeto focado exclusivamente no design, o objetivo foi sempre proporcionar uma **experiência intuitiva, moderna e acessível**.

Criação de um ambiente seguro

Num projeto **Next.js**, o ficheiro `.env` é utilizado para armazenar **informações sensíveis** de forma segura, como **palavras-passe, chaves de API ou URLs de ligação a bases de dados**.

Estas variáveis não ficam visíveis no código-fonte e são acessíveis apenas dentro do ambiente da aplicação, o que ajuda a manter a segurança e organização do projeto.

Por padrão, o ficheiro `.env` é adicionado ao `.gitignore`, o que impede que seja incluído em repositórios públicos ou partilhado por engano, evitando a exposição de dados confidenciais.

As variáveis definidas neste ficheiro podem ser acedidas no código através de **process.env**, como ilustrado no exemplo:

```
GOOGLE_SECRET_ID: process.env.GOOGLE_SECRET_ID;
```

Este mecanismo permite centralizar e proteger valores críticos da aplicação de forma prática, tornando o ficheiro `.env` uma **ferramenta essencial para a segurança e manutenção do projeto**.

Ligação à base de dados com Mongoose

O que é o MongoDB?

O **MongoDB** é uma base de dados **NoSQL**, orientada a documentos. Ao contrário das bases de dados relacionais (como MySQL ou PostgreSQL), onde os dados são organizados em **tabelas e linhas**, o MongoDB armazena os dados em **coleções de documentos**, semelhantes a objetos **JSON**.

Cada documento pode ter uma estrutura diferente, o que torna o MongoDB especialmente útil em projetos com dados variados ou dinâmicos — como aplicações web modernas.

É uma base de dados **flexível, escalável e rápida**, muito utilizada em conjunto com aplicações feitas em **JavaScript/TypeScript**, como é o caso deste projeto.

O que é o Mongoose?

O **Mongoose** é uma biblioteca para **Node.js** que facilita a ligação e o uso do MongoDB.

Com o Mongoose, é possível:

- definir **schemas** que estruturam os dados de forma consistente;
- validar automaticamente os dados inseridos;
- executar operações **CRUD** (criar, ler, atualizar, eliminar);
- gerir relações entre dados;

Instalação

A instalação do Mongoose é feita com o seguinte comando no terminal:

```
npm install mongoose
```

Se a instalação for bem-sucedida, a dependência será adicionada à pasta **node_modules**.

Ligaçāo à base de dados

Para estabelecer a ligação, foi criado um ficheiro chamado **mongodb.js** dentro da pasta **/lib**, com o seguinte código:

```
import mongoose from "mongoose";
export const connectMongoDB = async () => {
  try {
    await mongoose.connect(process.env.MONGODB_URL);
    console.log("Conectado a MongoDB");
  } catch (error) {
    console.log("Erro a conectar:", error);
  }
};
```

A função **connectMongoDB** tenta ligar-se ao URL definido no ficheiro **.env**. Se for bem-sucedida, é apresentada a mensagem de sucesso. Caso contrário, é mostrado o **erro** no terminal.

Submissão segura de dados com API e Mongoose

O envio de dados segue sempre uma **estrutura lógica** e sequencial:

- **Front-end:** onde o utilizador insere os dados (ex.: formulário);
- **API:** recebe os dados do *front-end* e processa-os;
- **Base de dados:** armazena a informação permanentemente através do modelo Mongoose.

Exemplo com *fetch*

No *front-end*, utiliza-se a função *fetch* para enviar dados para uma API:

```
await fetch("/api/users/add", {  
  method: "POST",  
  headers: { "Content-Type": "application/json" },  
  body: JSON.stringify({  
    name: "Ana",  
    email: "ana@email.com",  
  }),  
});
```

- o método **POST** indica que estamos a enviar dados;
- o conteúdo é convertido para **JSON**;
- o destino é uma rota interna, que trata a inserção dos dados.

API (Back-End)

A API serve como **ponte entre o front-end e a base de dados**. Abaixo está um exemplo de API em Next.js com Mongoose:

```
import dbConnect from "@/lib/dbConnect";  
import User from "@/models/User";
```

```
export default async function handler(req, res) {  
    await dbConnect();  
    if (req.method === "POST") {  
        const user = new User(req.body);  
        await user.save();  
        res.status(201).json({ success: true });  
    }  
}
```

- **dbConnect()** estabelece ligação à base de dados.
- Criamos um novo objeto User com os dados recebidos.
- Por fim, utilizamos **.save()** para guardar na MongoDB.

Carregamento inteligente de dados com React

Em várias partes da aplicação Jet Hire, é necessário **esperar por dados vindos do servidor** (como empresas, ofertas ou candidatos). Para melhorar a experiência do utilizador, foi implementado um sistema de **renderização condicional** com **useState**, permitindo mostrar **mensagens de carregamento**, **spinners** ou **mensagens alternativas**.

A lógica base é sempre semelhante e pode ser reutilizada em qualquer componente:

Declaração do estado de carregamento

```
const [loading, setLoading] = useState(true);
```

Requisição de dados com **useEffect**

```
useEffect(() => {  
    async function fetchData() {  
        try {  
            const res = await fetch("/api/...");  
            const data = await res.json();  
            setDados(data);  
        } catch (err) {  
            console.error("Erro:", err);  
        } finally {  
        }  
    }  
}, []);
```

```
    setLoading(false);  
}  
}  
  
fetchData();  
}, []);
```

O *finally* garante que o *loading* é sempre desativado, quer o pedido tenha sucesso ou não.

Dependendo do valor de *loading*, o conteúdo muda automaticamente:

```
{loading ? (  
    <p>A carregar dados...</p>  
) : dados.length === 0 ? (  
    <p>Nenhum resultado encontrado.</p>  
) : (  
    <ul>  
        {dados.map((item) => (  
            <li key={item.id}>{item.nome}</li>  
        ))}  
    </ul>  
)}
```

Firebase – Base de dados em tempo real

Uma das funcionalidades mais importantes da aplicação foi a implementação de um **sistema de chat em tempo real**, que permite a comunicação entre os membros da mesma empresa dentro da plataforma Jet Hire.

Para garantir uma experiência fluida, segura e sincronizada, optei por usar o **Firebase Realtime Database**, uma base de dados que permite armazenar e sincronizar dados entre os utilizadores em tempo real, sem necessidade de recarregar a página.

Criação da conta

O primeiro passo foi criar uma conta no site oficial do Firebase e configurar um novo projeto com o nome **Jet Hire**.

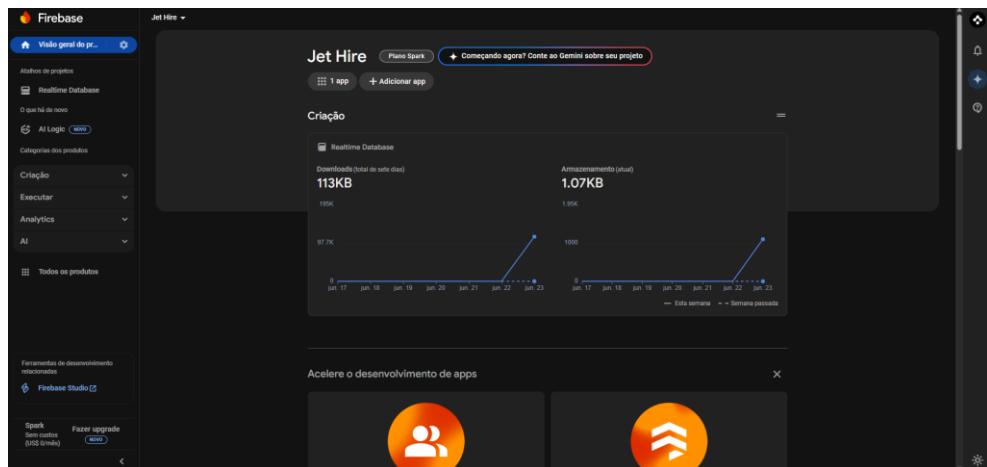


Figura 41 - Dashboard da Firebase

Integração com o projeto

A integração com Firebase foi feita através da instalação do SDK oficial:

```
npm install firebase
```

E a seguir, criei o ficheiro `src/lib/firebase.ts` com a configuração da aplicação:

```
import { initializeApp, getApps, getApp } from "firebase/app";
import { getDatabase } from "firebase/database";

const firebaseConfig = {
  apiKey: "...",
  authDomain: "...",
  databaseURL: "...",
};

const app = getApps().length ? getApp() : initializeApp(firebaseConfig);
export const db = getDatabase(app);
```

Cloudinary — Gestão de imagens e documentos

Para armazenar imagens e documentos — como **logótipos, fotografias de perfil, banners e CVs** — utilizei a plataforma **Cloudinary**.

Esta ferramenta permite **fazer o upload e gestão de ficheiros na cloud**, poupando espaço e **evitar que a base de dados de armazene ficheiros pesados**. Em vez de guardar o conteúdo de imagens diretamente, **apenas é guardada uma string (URL)**, o que torna o acesso mais rápido e a gestão mais leve.

Funções para envio e eliminação

No projeto, foram criadas duas funções principais no ficheiro **cloudinary.ts**:

Upload de imagens

```
async function uploadToCloudinary(buffer: Buffer) {  
  return new Promise<{ secure_url: string; public_id: string }>((resolve, reject) => {  
    cloudinary.uploader  
      .upload_stream(  
        {  
          folder: "pfp",  
          resource_type: "image",  
          overwrite: true,  
        },  
        (error, result) => {  
          if (error || !result) reject(error);  
          else resolve({ secure_url: result.secure_url, public_id: result.public_id });  
        }  
      )  
      .end(buffer);  
  });  
}
```

Esta função:

- recebe a imagem em **buffer**;
- envia-a para a pasta **pfp** no Cloudinary;
- retorna o **URL seguro** e o **ID público**, que são depois guardados na base de dados.

Eliminação de imagens

```
async function deleteFromCloudinary(publicId: string) {  
  try {  
    await cloudinary.uploader.destroy(publicId);  
  } catch (error) {  
    console.error("Erro ao apagar imagem:", error);  
  }  
}
```

Esta função:

- Elimina a imagem através do seu **publicId**.
- É utilizada, por exemplo, quando um utilizador atualiza a sua foto de perfil.

NextAuth – Sistema de autenticação com sessões seguras

A autenticação da aplicação foi implementada com **NextAuth**, uma biblioteca desenvolvida especialmente para aplicações **Next.js**, que permite gerir sessões de forma segura e flexível.

Neste projeto, utilizei apenas um método de autenticação: **as credenciais**.

Autorização personalizada com Credentials Provider

A autenticação com **credenciais** é configurada no *CredentialsProvider*, e a lógica principal acontece dentro da função *authorize*. Este método foi essencial para adaptar o *login* à minha estrutura de utilizadores, onde existem **administradores** e **membros de empresas** (armazenados como elementos de um array na empresa).

```
async authorize(credentials) {  
    await dbConnect();  
    const { user, pass } = credentials!;  
    const admin = await Administrator.findOne({ user });  
  
    if (admin && await bcrypt.compare(pass, admin.password)) {  
        return {  
            id: admin._id.toString(),  
            name: admin.name,  
            email: admin.email,  
            type: "admin",  
            role: admin.role,  
        };  
    }  
  
    const company = await Company.findOne({ "team.user": user });  
    const member = company?.team.find((m) => m.user === user);  
  
    if (member && await bcrypt.compare(pass, member.password)) {  
        return {  
            id: member._id.toString(),  
            name: member.name,  
            email: member.email,  
            type: "company",  
            company: company.name,  
            role: member.role,  
        };  
    }  
  
    throw new Error("Credenciais inválidas.");  
}
```

Esta lógica garante que:

1. A ligação à **base de dados** está ativa com dbConnect();
2. A palavra-passe é **verificada com bcrypt**, comparando a versão encriptada;
3. Se tudo estiver correto, os dados do utilizador são **devolvidos e armazenados na sessão**.

Sessões com JWT

Para guardar os dados da sessão, optei por usar **JWT (JSON Web Tokens)** como estratégia, o que evita guardar sessões no servidor e torna a aplicação mais leve e escalável.

```
session: {  
    strategy: "jwt",  
}
```

Quando o utilizador é autenticado com sucesso, os seus dados são guardados dentro do *token* através do **callback jwt()**:

```
async jwt({ token, user }) {  
    if (user) {  
        token.id = user.id;  
        token.name = user.name;  
        token.type = user.type;  
        token.role = user.role;  
        token.company = user.company;  
    }  
    return token;  
}
```

O que é um JWT?

Um **JWT** é como um **cartão de identificação digital**: contém informação do utilizador (como o nome, tipo, cargo, etc.), codificada e assinada, e acompanha cada pedido feito à aplicação. Desta forma, não é preciso consultar a base de dados constantemente.

Sessões acessíveis no front-end

Para tornar os dados acessíveis no lado do cliente, o `callback session()` transforma o *token JWT* numa sessão visível:

```
async session({ session, token }) {  
  session.user.id = token.id;  
  session.user.name = token.name;  
  session.user.type = token.type;  
  session.user.role = token.role;  
  session.user.company = token.company;  
  return session;  
}
```

Isto permite:

- **mostrar dados do utilizador** autenticado (ex.: nome, empresa, cargo);
- **proteger rotas** (ex.: páginas só acessíveis a *admins*);
- **personalizar a experiência** com base no tipo de utilizador.

Exemplo prático

Se o utilizador **José Silva**, da empresa **TechNova**, com cargo de **recruiter**, iniciar sessão, o seu *token JWT* será algo como:

```
{  
  "id": "664abec123",  
  "name": "José Silva",  
  "role": "recruiter",  
  "type": "company",  
  "company": "TechNova"  
}
```

Botões de *login* com NextAuth

Para integrar o *login* na interface, utilizei os métodos `signIn()` e `signOut()` da própria biblioteca:

```
import { signIn, signOut } from "next-auth/react";

<button onClick={() => signIn()}>Entrar</button>
<button onClick={() => signOut()}>Sair</button>
```

Formulários dinâmicos e reutilizáveis em React

Ao longo da aplicação, a gestão de formulários é feita com três elementos centrais do React: **`useState`, `handleChange` e `handleSubmit`**. Estes são usados em praticamente todas as interações com o utilizador, como o **registro, edição de perfis, submissão de dados**, entre outros.

O **`useState`** serve para armazenar o estado dos dados inseridos.

```
const [formData, setFormData] = useState({ nome: "", email: "", mensagem: ""});
```

O **`handleChange`** atualiza automaticamente os campos à medida que o utilizador escreve:

```
const handleChange = (e) => {
  setFormData({ ...formData, [e.target.name]: e.target.value });
};
```

O **`handleSubmit`** processa o envio, impedindo o comportamento padrão do navegador e permitindo aplicar validações ou enviar dados para a API:

```
const handleSubmit = (e) => {
  e.preventDefault();
  console.log(formData);
};
```

Esta abordagem permite uma **gestão controlada e reutilizável** de formulários, essencial para manter o código limpo e funcional.

Projeto principal

A página principal da Jet Hire foi desenvolvida com **Next.js 12** e **JavaScript**, recorrendo ao sistema tradicional de **pages/** e **rotas API internas**.

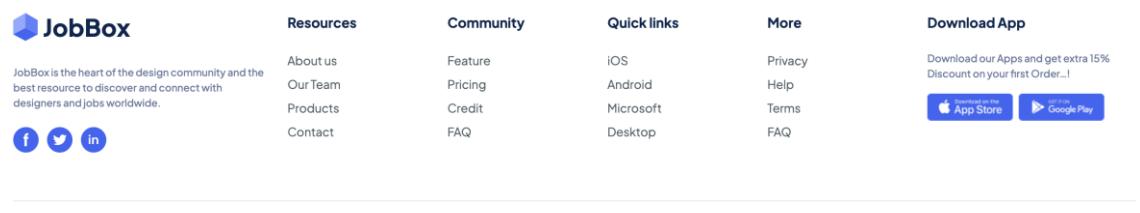
A base de dados utilizada é o **MongoDB**, com ligação feita através de **Mongoose**. O *layout* base partiu de um **template** que foi adaptado para React, com componentes funcionais personalizados. A interface é construída através de Bootstrap.

Após concluir as alterações estruturais iniciais, iniciei a edição do *template* base com o objetivo de o adaptar ao **sitemap definido** e à **paleta de cores personalizada** da plataforma. O foco principal foi tornar o *design* mais limpo, funcional e alinhado com a identidade visual da Jet Hire.

Edição do *template*

Rodapé

Comecei por simplificar o **rodapé**, removendo o primeiro bloco que considerei redundante. Editei também o texto do **copyright** e traduzi as secções correspondentes às **políticas** e **termos de utilização**, mantendo uma linguagem mais acessível e adequada ao contexto do projeto.



The screenshot shows the initial footer section of the Jet Hire website. It includes the JobBox logo, social media links (Facebook, Twitter, LinkedIn), a brief description of JobBox, and links to Resources, Community, Quick links, More, and Download App sections. The footer also contains copyright information and links to Privacy Policy, Terms & Conditions, and Security.

Figura 42 - Rodapé inicial

Copyright © 2025. Desenvolvido por Tomé Almeida

Política de privacidade | Termos & Condições | Política de cookies

Figura 43 - Rodapé final

Barra de navegação

Na **barra de navegação**, substituí o logótipo existente pelo novo logótipo da Jet Hire e reorganizei os botões para refletirem as páginas principais da aplicação. Esta alteração permitiu uma navegação mais direta e coerente com a estrutura planeada.



Figura 44 - Barra de navegação inicial



Figura 45 - Barra de navegação final

Página inicial

Secção “Destaque principal”

A **primeira secção** da página inicial é dedicada ao destaque principal, onde está visível o **slogan** da Jet Hire acompanhado por imagens que ilustram o propósito da plataforma. Para dar mais impacto visual, apliquei um **fundo com gradiente entre preto e vermelho**, tornando esta área mais atrativa e moderna. Os **botões** desta secção foram ajustados: um deles direciona o utilizador para a secção “Como funciona”, enquanto o outro conduz à página **Sobre nós**.

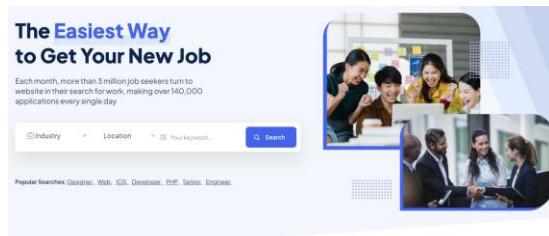


Figura 46 - Destaque principal inicial



Figura 47 - Destaque principal final

Secção “Slider de categorias”

A seguir, desenvolvi uma secção com um **slider de categorias** que permite explorar os diferentes tipos de ofertas disponíveis. Esta secção aparece logo no **viewport inicial**, para garantir um impacto direto sem sobrecarregar o utilizador.

Utilizei a biblioteca **Swiper.js** para implementar o *slider*, que foi configurado com as seguintes opções:

- **loop** ativo para repetição contínua;
- **autoplay** com atraso e velocidade personalizados para criar fluidez;
- **breakpoints** definidos para garantir **responsividade** em diferentes resoluções;
- **e 4 elementos visíveis** por slide em ecrãs maiores.

Os dados de cada categoria são armazenados num **array de objetos** com estrutura semelhante a JSON, incluindo o **ícone SVG**, o **título** e a **descrição**.

Os ícones foram importados do site **Heroicons**, bastando copiar o código SVG correspondente. Aqui está um exemplo resumido da estrutura:

```
const data = [
  {
    icon: <svg>...</svg>,
    title: "Desenvolvimento",
    description: "Software, Web, Jogos, etc..."
  },
  ...
];
```

Com esta abordagem, o *slider* ficou dinâmico, organizado e altamente reutilizável.

Browse by category

Find the job that's perfect for you. about 800+ new jobs everyday

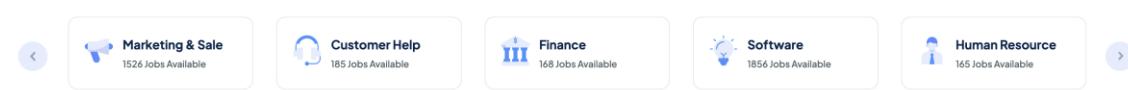


Figura 48 – Slider de categorias inicial

Procurar por categoria

Encontra o trabalho ideal para ti. Está atento/a a todas as oportunidades!



Figura 49 – Slider de categorias final

Secção “Como Funciona”

A secção “Como Funciona” é o destino do botão principal da secção de destaque. Aqui apenas foram feitas alterações **estéticas**, como a troca da **cor de fundo** e a **adaptação dos textos**, para melhor transmitir os passos que um utilizador pode seguir ao usar a plataforma.

How It Works

Just via some simple steps, you will find your ideal candidates you are looking for!

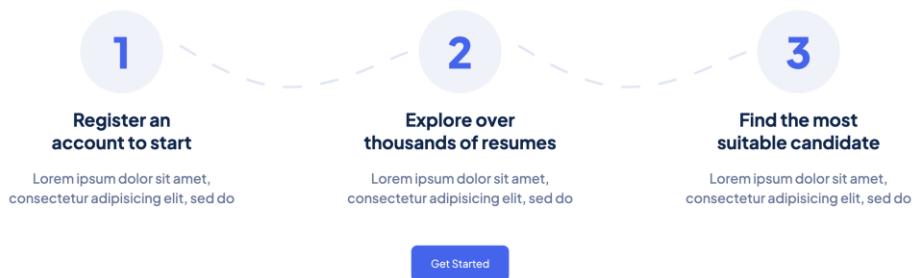


Figura 50 - Como funciona inicial

Como funciona

Com apenas alguns passos, vais encontrar o trabalho ideal para ti!



Figura 51 - Como funciona final

Secção “Ofertas Mais Recentes”

Nesta secção, mantive a estrutura já existente e fiz alterações semelhantes às anteriores: adaptação das **cores** para respeitar a **paleta definida**, alteração dos **títulos** e reestruturação do *layout* para melhor destacar as ofertas. Esta secção ajuda a mostrar rapidamente as **novidades** da plataforma.

Jobs of the day

Search and connect with the right candidates faster.

[Management](#)
[Marketing & Sale](#)
[Finance](#)
[Human Resource](#)
[Retail & Products](#)
[Content Writer](#)

 **LinkedIn** New York, US

UI / UX Designer fulltime Fulltime | 4 minutes ago

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Recusandae architecto eveniet, dolor quo repellendus pariatur.

Adobe XD Figma Photoshop

\$500/Hour [Apply Now](#)

 **Adobe Illustrator** New York, US

Full Stack Engineer Part time | 5 minutes ago

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Recusandae architecto eveniet, dolor quo repellendus pariatur.

React NodeJS

\$800/Hour [Apply Now](#)

 **Bing Search** New York, US

Java Software Engineer Full time | 6 minutes ago

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Recusandae architecto eveniet, dolor quo repellendus pariatur.

Python AWS Photoshop

\$250/Hour [Apply Now](#)

 **Dailymotion** New York, US

Frontend Developer Full time | 6 minutes ago

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Recusandae architecto eveniet, dolor quo repellendus pariatur.

TypeScript Java

\$250/Hour [Apply Now](#)

Figura 52 - Ofertas mais recentes inicial

Ofertas mais recentes

Encontra as ofertas de trabalho mais recentes.

[Desenvolvimento](#)
[Redes](#)
[Cibersegurança](#)
[Dados & IA](#)
[Suporte Técnico](#)

 **LinkedIn** New York, US

UI / UX Designer fulltime Fulltime | 4 minutes ago

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Recusandae architecto eveniet, dolor quo repellendus pariatur.

Adobe XD Figma Photoshop

\$500/Hour [Apply Now](#)

 **Adobe Illustrator** New York, US

Full Stack Engineer Part time | 5 minutes ago

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Recusandae architecto eveniet, dolor quo repellendus pariatur.

React NodeJS

\$800/Hour [Apply Now](#)

 **Bing Search** New York, US

Java Software Engineer Full time | 6 minutes ago

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Recusandae architecto eveniet, dolor quo repellendus pariatur.

Python AWS Photoshop

\$250/Hour [Apply Now](#)

 **Dailymotion** New York, US

Frontend Developer Full time | 6 minutes ago

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Recusandae architecto eveniet, dolor quo repellendus pariatur.

TypeScript Java

\$250/Hour [Apply Now](#)

Figura 53 - Ofertas mais recentes final

Secção “Procurar Trabalho”

A secção “Procurar Trabalho” foi importada de outra página do *template*. Esta incluía inicialmente três imagens, que reduzi para apenas **uma** para simplificar o visual. Também foi adicionado um **texto de destaque** que promove a plataforma, juntamente com um **botão** que redireciona diretamente para a página de **ofertas de emprego**.

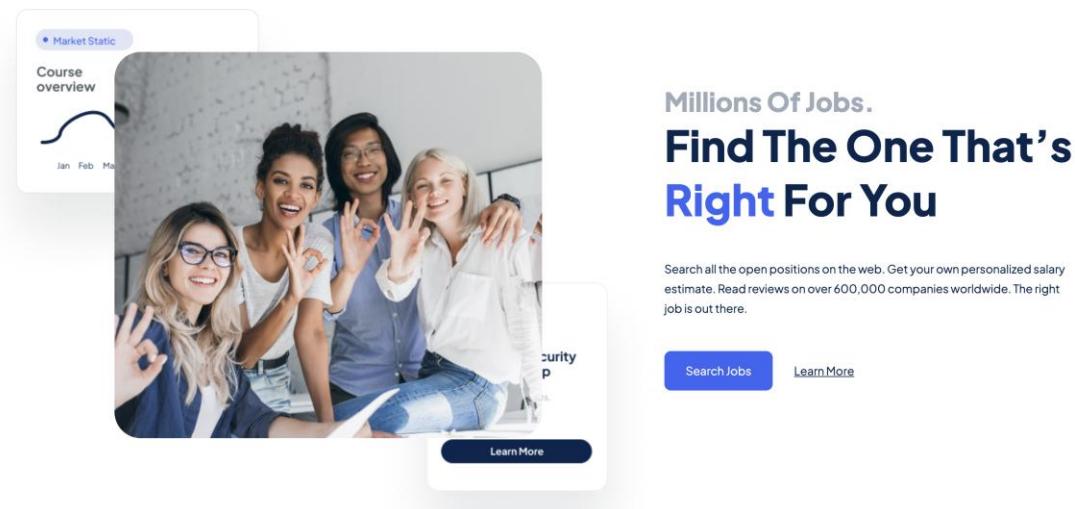


Figura 54 - Procurar trabalho inicial



Figura 55 - Procurar trabalho final

Secção “Candidatos Recentes”

Esta secção apresenta os **últimos candidatos registados** na plataforma, através de **cartões personalizados** com foto de perfil, *banner*, nome, descrição, competências e localização. Cada cartão inclui ainda um botão “**Ver Perfil**” que leva à página individual do utilizador.

A estrutura é **modular e responsiva**, adaptando-se a vários tamanhos de ecrã. Os candidatos são renderizados dinamicamente a partir de um *array* de objetos, usando um componente reutilizável (CandidateCard), o que facilita a manutenção.

```
{candidates.map((candidate, index) => (
  <CandidateCard key={index} data={candidate} />
))}
```

Foi também integrado um **slider interativo** que permite navegar pelos perfis de forma fluida. No final da secção, existe um botão “**Ver mais candidatos**” que redireciona para a listagem completa.

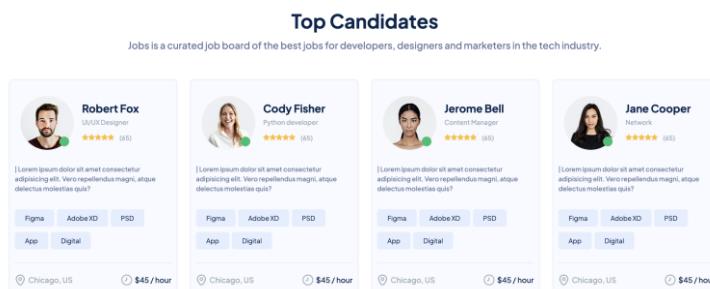


Figura 56 - Candidates recentes inicial

Candidatos recentes

Encontra os candidatos mais recentes da plataforma.

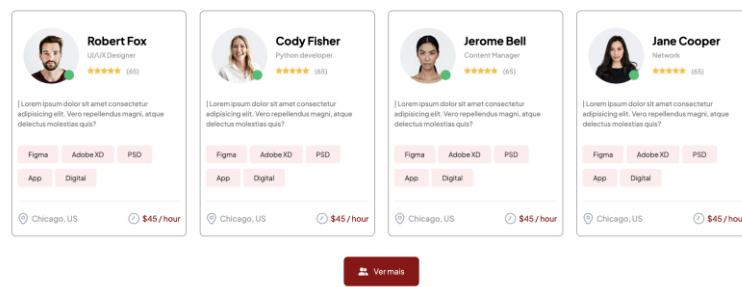


Figura 57 - Candidates recentes final

Secção “Cria o Teu Perfil”

Seguindo o mesmo princípio, a secção “Cria o Teu Perfil” foi editada apenas a nível de **texto e cores**, adaptando o conteúdo à realidade da Jet Hire.

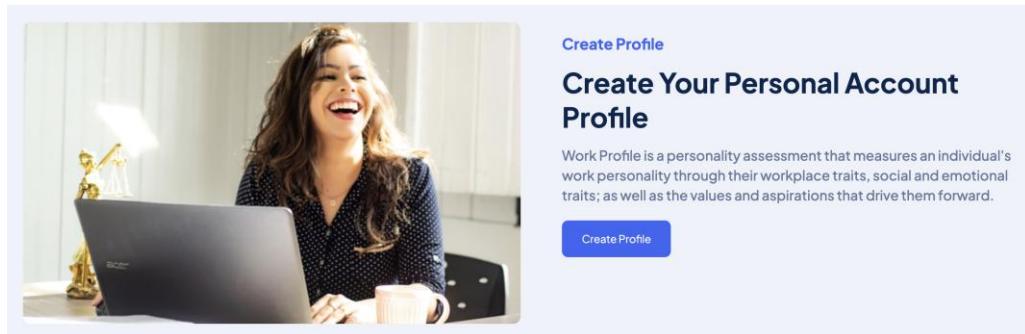


Figura 58 - Cria o teu perfil inicial



Figura 59 - Cria o teu perfil final

Página de ofertas

Esta é a página principal onde os utilizadores podem consultar todas as **ofertas de emprego disponíveis**. Foi dividida em três secções principais: **pesquisa, menu lateral com filtros e apresentação das ofertas**.

Secção de pesquisas

A primeira parte da página apresenta um **banner de destaque** com o número total de ofertas disponíveis e uma **barra de pesquisa por palavra-chave**. Esta funcionalidade ajuda o utilizador a encontrar oportunidades mais rapidamente, de forma intuitiva.

Fiz ainda **alterações visuais** relevantes — como o fundo, ícones e botão de pesquisa — para estarem de acordo com a **paleta de cores** da plataforma e transmitirem maior **consistência visual**.

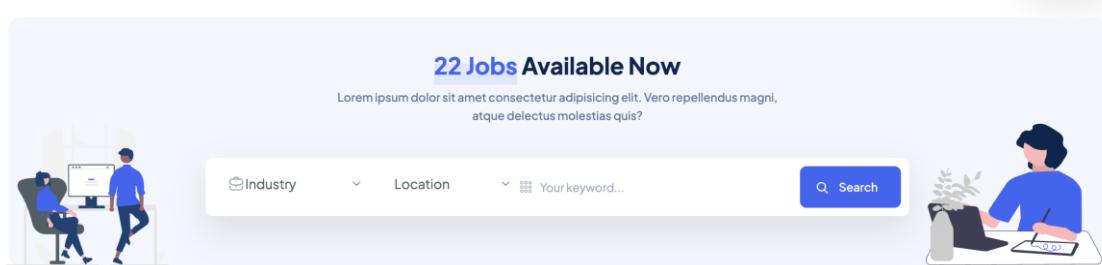


Figura 60 - Secção de pesquisas inicial



Figura 61 - Secção de pesquisas final

Menu lateral

Do lado esquerdo da página adicionei uma **barra de filtros avançados** que permite ao utilizador ajustar os resultados conforme as suas preferências. O utilizador pode filtrar por:

- área de atividade;
- nível de experiência;
- modalidade;
- tempo de publicação;
- tipo de trabalho.

Modalidade

- Presencial 12
- Remoto 65
- Híbrido 58

Figura 62 - Menu lateral - Modalidade

Tipo de trabalho

- Tempo inteiro 25
- Part-time 64
- Estágio 78
- Freelancer 97

Figura 63 - Menu lateral - Tipo de trabalho

Estes filtros foram **editados visualmente** e estruturados para garantir uma **experiência fluida**, mas também foram pensados para permitir **adaptações futuras** — por exemplo, novas categorias ou filtros dinâmicos que possam surgir no desenvolvimento.

Apresentação de oferta

Na parte central e mais importante da página, encontram-se listadas as **ofertas de emprego**. Cada oferta aparece num cartão igual ao exemplificado abaixo:



Figura 64 - Apresentação de oferta

Página de empresas

Esta página tem como principal objetivo apresentar o conjunto de empresas registadas na plataforma e permitir que novas empresas iniciem o seu processo de registo de forma simples e organizada.

Registo de empresas

No final da página, encontra-se um **bloco destacado** que convida as empresas a juntarem-se à plataforma:



Figura 65 - Placeholder do formulário do registo de empresas

Ao clicar no botão “Formulário”, é apresentado um formulário completo com vários campos organizados por secções: **dados da empresa, localização, contactos e responsável pelo registo**.

O formulário de registo de empresas é apresentado em modo modal sobre uma interface web. O topo mostra o logo "jethire" e uma barra com links: "Procurar trabalho", "Empresas", "Candidatos", "Jet AI", "Sobre nós", "Criar conta" e "Login". O formulário tem o seguinte layout:

- Título:** Formulário de registo de empresa
- Subtítulo:** Entraremos em contacto consigo o mais rápido possível.
- Seção 1: Dados da empresa**
 - Campos: Nome, NIF, Ano de fundação (com placeholder "Selec.")
- Seção 2: Localização**
 - Campos: País (placeholder "Selec."), Cidade, Morada completa
- Seção 3: Contactos**
 - Campos: Telefone, Email, Website
- Seção 4: Responsável pelo registo**
 - Campos: Nome, Email, Telefone
- Botão de envio:** Enviar (destacado em vermelho)

À base da modal, há uma barra com links: "Copyright © 2025. Desenvolvido por Tomé Almeida", "Política de privacidade", "Termos & Condições" e "Política de cookies".

Figura 66 - Formulário de registo de empresas

Todos os campos são obrigatórios, garantindo que a submissão do pedido de registo seja feita com a informação mínima necessária para análise por parte da administração.

Estrutura geral da página

A página é composta por três blocos principais:

- Cabeçalho com Barra de Pesquisa**

Na parte superior, é apresentado o número de empresas disponíveis, acompanhado de uma **barra de pesquisa**. Esta funcionalidade permite ao utilizador filtrar empresas rapidamente por **nome, setor ou palavras-chave relevantes**.



Figura 67 - Cabeçalho da página de empresas

- Apresentação das Empresas**

Por fim, são apresentados os **cartões informativos** de cada empresa, contendo:

- logótipo (se disponível);
- nome e localização;
- áreas de atuação (*tags* coloridas);
- botão para ver perfil completo.


 A screenshot of the Jet Hire platform displaying two company profiles side-by-side.
 The left profile is for 'Orima'. It features a blue-toned background image of a modern building, a circular logo with three black shapes, and the name 'Orima' in bold. Below the name is the text 'Sem descrição disponível.' and a row of colored buttons labeled 'Marketing', 'E-Commerce', and 'Cibersegurança'. At the bottom are the location 'Coimbra, Noruega' and a 'Ver Perfil' button.
 The right profile is for 'jhasdf'. It features a background image of a keyboard and a brown folder, a circular logo with a red dot, and the name 'jhasdf' in bold. Below the name is the text 'Sem descrição disponível.' and a row of colored buttons labeled 'Dados & IA', 'Desenvolvimento', and 'E-Commerce'. At the bottom are the location 'Coimbra, Portugal' and a 'Ver Perfil' button.

Figura 68 - Apresentação de empresas

Página de candidatos

Esta página foi desenvolvida para apresentar uma **listagem de todos os candidatos registados na plataforma**, permitindo uma navegação rápida e visual entre perfis.

A estrutura geral da página foi inspirada no *template* original, mas **introduzi várias melhorias** para tornar a experiência mais funcional, moderna e adaptada aos objetivos da Jet Hire.

Cabeçalho e pesquisa

No topo da página, adicionei uma **área de destaque com o número total de candidatos disponíveis**, uma breve descrição e um **campo de pesquisa por palavras-chave**.



Figura 69 - Cabeçalho de página de candidatos

Ordenação personalizada

Implementei também uma **dropdown de ordenação**, onde é possível alternar entre “Mais recente” e “Mais antigo”.

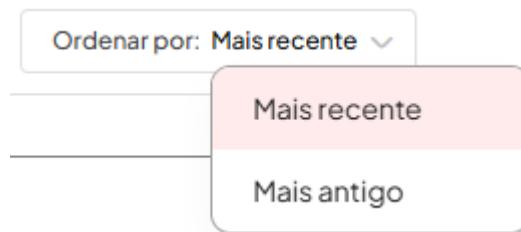


Figura 70 - Dropdown de ordenação

Cartão de utilizador

Para apresentar os candidatos na listagem, criei **cartões visuais personalizados**, baseando-me no estilo do *template* mas com **várias modificações e animações** que tornam a navegação mais dinâmica e intuitiva.

Cada candidato é apresentado dentro de um **cartão compacto e informativo**, com os seguintes elementos:

- **fotografia de perfil** (ou uma imagem padrão caso não exista);
- **nome do utilizador e função atual** (ou “À procura de emprego” por defeito);
- uma **descrição curta**, limitada a 120 caracteres, ou a indicação “Sem descrição” se o campo estiver vazio;
- uma área com **skills** do utilizador, até um máximo de 4, ordenadas por relevância (percentagem atribuída);
- a **localização** (cidade e país, se estiverem disponíveis);
- um **botão "Ver perfil"**, que redireciona para a página individual do candidato.

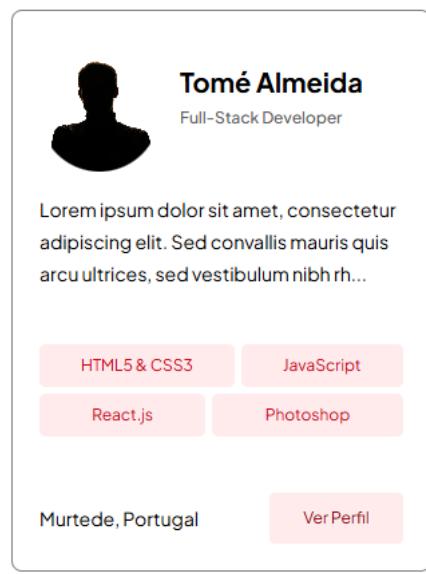


Figura 71 - Cartão de utilizador

Página de início de sessão

A página de autenticação foi apenas **adaptada** visualmente para se alinhar com a **identidade gráfica** do projeto. A estrutura principal foi mantida, mas houve pequenas alterações para **melhorar a experiência do utilizador**.

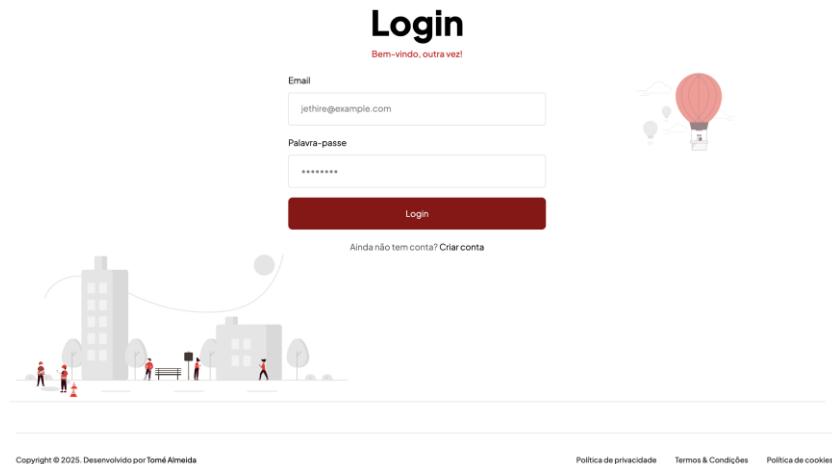


Figura 72 - Página de início de sessão

Página de registo

Esta página foi baseada na estrutura visual da **página de início de sessão**, garantindo **consistência no design** da plataforma. Apesar disso, foram incluídos todos os **campos essenciais para criar uma nova conta** de utilizador.

Estrutura e funcionalidades

A estrutura foi mantida minimalista, mas funcional. Os campos disponíveis são:

- **Nome completo**
- **Email**
- **Nome de utilizador**
- **Palavra-passe**
- **Confirmação da palavra-passe**

Além disso, adicionei a **verificação de aceitação dos termos e políticas** antes de submeter o formulário.



Figura 73 - Página de registo

Página de perfil de utilizador

Nesta página, optei por manter a **estrutura base do template**, mas introduzi **ajustes visuais e funcionais** para alinhar com a experiência pretendida para a Jet Hire.

Banner e CV

No topo da página, personalizei o **banner** de cada utilizador, removendo elementos do *template* que considerei desnecessários. Também **alterei a apresentação do currículo**: em vez de obrigar o utilizador a fazer download, adicionei um **ícone visual mais intuitivo**, que permite abrir diretamente o ficheiro no *browser*.

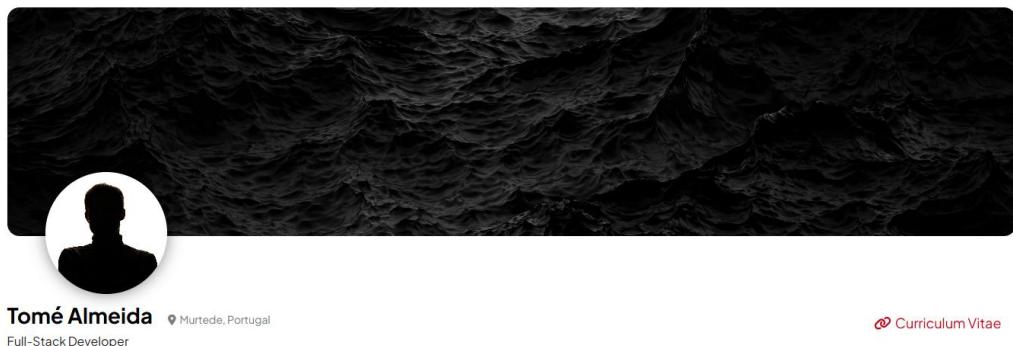


Figura 74 - Banner

Descrição e competências

A secção da descrição manteve-se simples, com uma **pequena alteração nas cores** para melhor integração com a **paleta visual** do projeto.

A seguir, na **área de competências**, utilizei **etiquetas coloridas** para representar as *skills* do utilizador. Substituí os tons azuis do *template* pelo **vermelho principal da aplicação**, tornando o *design* mais coeso.

Competências profissionais

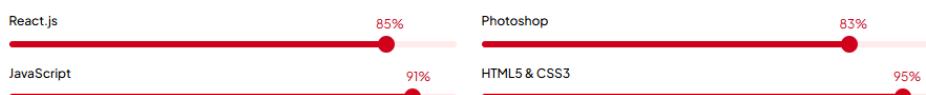


Figura 75 - Competências profissionais

Educação e experiência

Esta foi uma das áreas que mais **reestruturei**. O *template* usava uma lista genérica, pouco clara. Transformei-a numa secção com **blocos informativos**, onde cada item apresenta:

- **educação**: nome da instituição, curso e datas;
- **experiência**: empresa, cargo e datas.

Desta forma, a leitura ficou **mais organizada e direta**, facilitando a análise por parte das empresas.

Educação

- 🏠 Escola Técnico Profissional de Cantanhede
- 💻 Técnico de Gestão e Programação de Sistemas Informáticos
- ⌚ 2022 - Atual

Figura 76 - Educação

Experiência

- 🏠 Jet Hire
- 💻 Full-Stack Developer
- ⌚ 2022 - 2025

Figura 77 - Experiência

Portfólio

Acrescentei também uma secção para o **portfólio do utilizador**, algo que não estava presente no *template*. Cada item contém:

- **Uma imagem**
- **Uma legenda**
- Um **link clicável** para o projeto correspondente

Esta funcionalidade permite **visualizar rapidamente o trabalho do candidato**, o que é uma mais-valia na avaliação.



Figura 78 - Portfólio

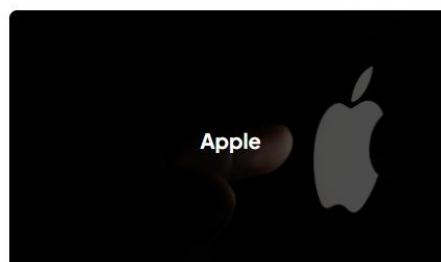


Figura 79 - Portfólio com hover

Barra Lateral

Por fim, mantive a **barra lateral direita** do *template*, mas com **pequenas adaptações visuais**. Esta secção mostra:

- Informações **principais** do utilizador
- **Localização**
- **Contactos**
- **Cargo**

É um **resumo essencial** que qualquer empresa pode consultar de forma rápida.



Figura 80 - Barra lateral

Página de edição de perfil

Para garantir uma experiência personalizada e adequada aos objetivos da Jet Hire, decidi **criar esta página de raiz**. Nenhuma das páginas do *template* original respondia às minhas necessidades, por isso optei por usar **Bootstrap** como base, especialmente pelo seu **sistema de grelhas (grids)**, o que me permitiu organizar os elementos de forma limpa e responsiva.

Dados básicos

A primeira secção reúne os **dados mais simples do utilizador**, como a **cidade, país, telemóvel, profissão atual, email de contacto** e a **imagem de perfil**. Esta informação ajuda a empresa a ter uma ideia inicial de quem é o candidato.

Criação de perfil

Olá, vamos melhorar o seu perfil!

Para se dar a conhecer melhor, e encontrar mais eficazmente a sua oportunidade de trabalho, preencha os seguintes dados.

Cidade <input type="text" value="ex: Cantanhede"/>	País <input type="text" value="Afeganistão"/>	Telemóvel <input type="text" value="ex: 918273878"/>	Foto de perfil <input type="button" value="+ Fazer upload"/>
Profissão atual <input type="text" value="(Se aplicável)"/>	Email de contactos <input type="text" value="(Deixar vazio se quiser manter o da conta)"/>		

Figura 81 - Dados básicos

Foto de fundo e descrições

Nesta secção, o utilizador pode **adicionar um banner personalizado** e escrever duas versões da sua descrição pessoal: uma **curta**, ideal para pré-visualizações, e uma **detalhada**, usada no perfil completo.

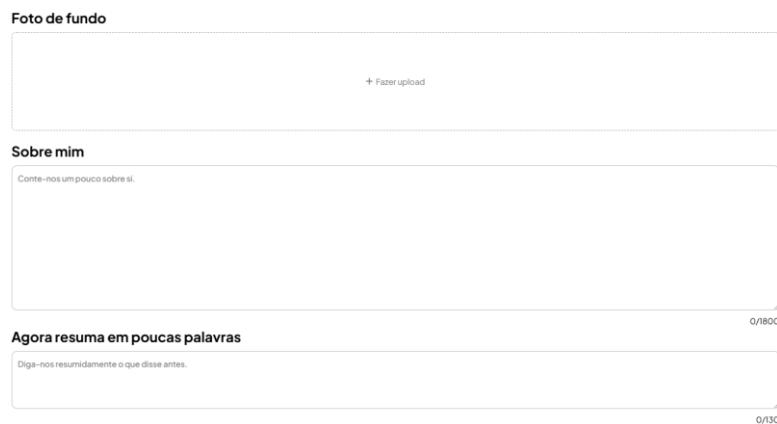


Foto de fundo

+ Fazer upload

Sobre mim
Conte-nos um pouco sobre si.

0/180

Agora resuma em poucas palavras
Diga-nos resumidamente o que disse antes.

0/130

Figura 82 - Descrições

Competências profissionais

Aqui, o utilizador pode indicar as suas **skills principais**, atribuindo a cada uma, uma **percentagem de proficiência** (de 0 a 100%).

A lógica base para a barra de progresso é:

```
const [skills, setSkills] = useState([]);
const handleAddSkill = () => {
  if (newSkill.name) setSkills([...skills, newSkill]);
};
```

Também inclui a funcionalidade de **remover competências e ajustar dinamicamente o nível** com eventos de rato, o que torna a interação mais visual e intuitiva.

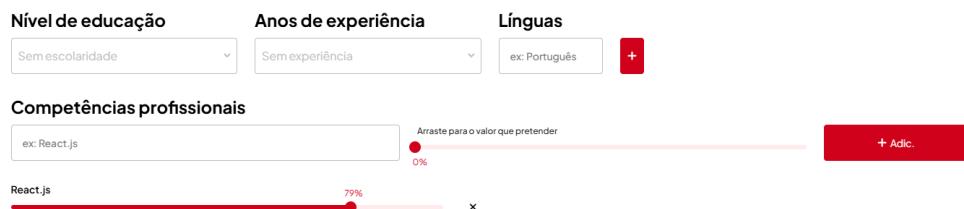


Figura 83 - Competências profissionais

Educação e experiência profissional

Esta foi uma das secções mais **desafiantes**, pois exigia um formulário flexível para múltiplos registo. Permite que o utilizador adicione:

- **experiência académica**: curso, instituição, ano de início e fim;
- **experiência profissional**: função, empresa, datas.

Cada registo é guardado num **array controlado por useState**, e o utilizador pode **adicionar ou remover entradas** conforme necessário.

```
const [educations, setEducations] = useState([]);

const addEducation = () => {
  setEducations([...educations, newEducation]);
};
```

Educação

Curso	Instituto	Início	Fim
Sem nenhum registo.			
Curso	Instituto	Início	Fim
ex: Engenheiro Informático	ex: ISEC	Selec. ▾	Selec. ▾
		Adicionar	

Experiência

Curso	Instituto	Início	Fim
Sem nenhum registo.			
Função	Empresa	Início	Fim
ex: Desenvolvedor Front-End	ex: Jet Hire	Selec. ▾	Selec. ▾
		Adicionar	

Figura 84 - Educação e experiência

A apresentação dos dados segue uma **estrutura em lista**, com todos os campos visíveis, organizados com grelhas *Bootstrap*.

Educação

Curso	Instituto	Início	Fim
Técnico de Gestão de Programação de Sistemas Informáticos		2022	Atual
Curso	Instituto	Início	Fim
ex: Engenheiro Informático	ex: ISEC	Selec. ▾	Selec. ▾
		Remover	Adicionar

Experiência

Função	Empresa	Início	Fim
ex: Desenvolvedor Front-End	ex: Jet Hire	Selec. ▾	Selec. ▾
		Adicionar	

Figura 85 - Exemplo de elemento de educação

Portfólio

Acrescentei uma secção dedicada ao **portfólio do utilizador**, onde este pode:

- inserir uma **imagem representativa**;
- adicionar uma **legenda**;
- colocar o **link direto para o projeto**.

Portfolio

Legenda	Link	Imagen	Adicionar
Waves	waves.pt	Adicionado	+ Adic.

Figura 86 – Portfólio

Cada projeto adicionado aparece numa grelha visualmente semelhante à da página pública.

Portfolio

Legenda	Link	Imagen	Adicionar
ex: Jet Hire	ex: jethire.pt	Fazer upload	+ Adic.

waves.pt



Figura 87 - Exemplo de portfólio

Links e documentos

Por fim, o utilizador pode partilhar os seus **links profissionais** (LinkedIn, GitHub e site pessoal), além de **enviar um ficheiro PDF com o currículo** ou **alterar o seu banner de perfil**.

Site pessoal	GitHub	LinkedIn	Curriculum Vitae
jethire.pt	www.github.com	www.linkedin.com	+ Fazer upload

Terminar o registo Ao confirmar esta caixa, está a concordar com os nossos [Termos & Condições](#).

Figura 88 - Links pessoais

Página de perfil de empresa

Nesta página, construí uma **estrutura visual sólida e completa** para apresentar os detalhes de cada empresa registada na plataforma.

Banner e identidade visual

No topo da página é apresentado um **banner personalizado**. Este elemento visual permite que cada empresa represente a sua imagem institucional com maior impacto. Logo abaixo, mostrei o **logótipo** e o **nome da empresa**, seguidos da **localização** (cidade e país, se estiverem disponíveis) e, opcionalmente, o **slogan corporativo**.



Figura 89 - Banner de empresa



Figura 90 - Identidade visual da empresa

Contactos

Na parte superior direita, adicionei um botão com o texto "**Contacte-nos**", que abre diretamente o email da empresa. Esta abordagem promove o contacto direto sem necessidade de qualquer tipo de navegação adicional.



Figura 91 - "Contacte-nos" de empresa

Descrição institucional

Mais abaixo, encontra-se a **descrição completa da empresa**, dividida em blocos com título e texto. Esta estrutura flexível permite que a empresa organize a sua apresentação em tópicos distintos, como missão, valores, história ou projetos atuais.

Cada bloco de descrição é mostrado com quebras de linha respeitadas, tornando o conteúdo fácil de ler e organizado.

Conectamos Talento ao Futuro do Trabalho

Na Jet Hire, acreditamos que o recrutamento deve ser rápido, eficaz e humano. A nossa missão é aproximar candidatos talentosos das empresas que realmente valorizam o seu potencial. Seja para um estágio de verão ou para uma carreira a longo prazo, oferecemos uma plataforma intuitiva que facilita o encontro perfeito entre quem procura e quem oferece oportunidades. Apostamos na inovação, na confiança e num processo simplificado para todos os envolvidos.

A Nova Forma de Contratar e Ser Contratado

A Jet Hire é mais do que uma plataforma de emprego — é um ecossistema de oportunidades. Ligamos empresas a candidatos de forma inteligente, com perfis personalizados, ofertas direcionadas e ferramentas que otimizam todo o processo de recrutamento. Com foco na experiência do utilizador, criámos um espaço onde as empresas podem destacar-se e os candidatos podem brilhar.

Figura 92 - Descrição de empresas

Barra lateral

Mantive uma **barra lateral à direita**, com várias secções de informação relevante:

- **áreas de trabalho:** Mostradas como etiquetas (*tags*). Limitei a visualização a 8 etiquetas para manter o *design* limpo;
- **remote friendly:** Indicação de se a empresa aceita trabalho remoto ou não;
- **data de registo:** Data formatada em português, com dia, mês e ano;

Contactos e morada

Logo abaixo, a barra lateral apresenta os contactos principais:

- **morada da empresa;**
- **telefone;**
- **email de contacto.**

No final da secção, é disponibilizado um **botão para enviar email diretamente**, facilitando novamente o contacto entre candidato e empresa.

Visão geral

Áreas de trabalho

Desenvolvimento • Dados & IA • Redes •
Cibersegurança • Cloud • Suporte Técnico •
E-Commerce

Remote friendly
Sim

Aderiu à Jet Hire em
6 de maio de 2025

Morada: Murtede, Rua dos Olivais, N°27
Telefone: 965360269
Email: jethire@gmail.com

[Enviar email](#)

Figura 93 - Barra lateral da página de empresas

Página de detalhes de oferta

A página de detalhes de oferta é uma das mais importantes de toda a aplicação, pois permite que os utilizadores consultem em profundidade todas as informações relacionadas com uma vaga específica.

Banner e identidade empresarial

No topo da página, utilizei o **banner da empresa associada à oferta**, obtido através do `company.banner` da base de dados. Logo abaixo, o utilizador encontra o **nome da função**, o **tipo de contrato** e o **tempo decorrido desde a publicação** da vaga, tudo isto formatado dinamicamente com base na data original.



Senior Full-Stack Developer

Tempo inteiro · Publicado há 4 dias

Figura 94 - Identidade visual da página de oferta

Informação geral da oferta

A primeira secção principal contém um conjunto de **informações essenciais** sobre a oferta, organizadas em blocos com ícones visuais:

- **área de trabalho** (ex: *Design, Marketing, TI*);
- **anos de experiência requeridos**;
- **faixa salarial** (mínimo e máximo, se aplicável);
- **nível de senioridade** (por exemplo, júnior, sénior, etc.);
- **modalidade de trabalho** (presencial, remoto ou híbrido);
- **data de publicação** da vaga.

Cada um destes blocos apresenta o conteúdo de forma clara e acessível, permitindo que o candidato avalie rapidamente se a vaga é adequada ao seu perfil.

Informação geral			
 Área	Consultoria, Desenvolvimento, E-Commerce	 Experiência	> 5 anos
 Salário	Não definido	 Nível	Sénior
 Modalidade	Híbrido	 Publicado	16/06/2025

Figura 95 - Informação geral da oferta

Descrição da oferta

Abaixo da secção informativa, a empresa pode incluir uma **descrição completa e detalhada da oferta**, organizada por blocos com títulos e texto. Esta estrutura flexível permite incluir várias secções, como responsabilidades, requisitos e benefícios.

Para garantir uma boa leitura, o texto suporta **quebras de linha automáticas** (\n) que são convertidas para HTML, o que melhora significativamente a apresentação da informação.

```
<p dangerouslySetInnerHTML={{ __html: item.text.replace(/\n/g, "<br />") }} />
```

Favoritos e candidaturas

Na página de detalhes da oferta, desenvolvi uma secção interativa com dois botões: “Candidatar-me” e “Favorito”. Estes permitem que o utilizador:

- **se candidate ou remova a candidatura** a uma oferta;
- **adicone ou remova** a oferta dos seus favoritos.

O estado de cada botão é dinâmico, sendo carregado automaticamente com base na sessão do utilizador. Para ações mais seguras, implementei um **modal de confirmação** sempre que o utilizador tenta candidatar-se ou desistir.

A interface adapta-se consoante o estado atual da candidatura e dos favoritos, tornando a interação mais intuitiva e moderna. Esta funcionalidade melhora a experiência dos utilizadores e torna a plataforma mais envolvente.

```
<FavoriteButton offerId={offer._id} />
```

Barra lateral com dados da empresa

No lado direito da página encontra-se uma **barra lateral com informações da empresa** que publicou a oferta:

- **logótipo da empresa;**
- **nome e localização;**
- **morada;**
- **telefone;**
- **email de contacto.**

Esta secção funciona como um pequeno perfil da empresa, permitindo que o candidato conheça melhor a entidade contratante sem sair da página da oferta.

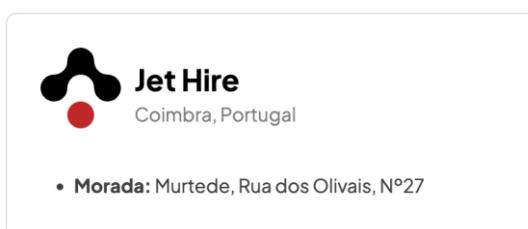


Figura 96 - Barra lateral de oferta

Tags da oferta

Ainda na barra lateral, implementei um bloco com **etiquetas representativas (tags)** associadas à oferta, como “AI”, “Marketing” ou “E-Commerce”. Estas *tags* ajudam os candidatos a identificar rapidamente as tecnologias ou áreas associadas à vaga e podem futuramente servir para **filtragem ou recomendação personalizada**.



Figura 97 - Tags da oferta

Página de contactos

A página de **contactos** foi uma adaptação visual e funcional do *template* pré-existente, onde aproveitei a estrutura base, mas introduzi alterações que alinham melhor com a identidade da Jet Hire. Esta página combina três secções principais: **formulário de contacto, apresentação pessoal e testemunhos de utilizadores**.

Formulário de contacto

Mantive o formulário já incluído no *template*, mas personalizei os campos e a lógica de submissão para torná-lo funcional e mais orientado ao contexto da Jet Hire. O formulário inclui os seguintes campos:

- **nome;**
- **empresa (opcional);**
- **email;**
- **telefone;**
- **mensagem.**

Incluí também uma **checkbox obrigatória** de aceitação dos “Termos & Condições”.

Contacte-nos

Entre em contacto

O movimento certo no momento certo faz toda a diferença. Viva o sonho de expandir o seu negócio.

Diga-nos o que se passa...

Ao clicar está a concordar com os [Termos & Condições](#)




Figura 98 - Formulário de contactos

Apresentação pessoal

Na segunda secção da página, adicionei uma **carta de apresentação** da equipa responsável pela aplicação.

Esta secção não existia no *template* original. Criei-a de raiz para **dar um rosto ao projeto**, tornando-o mais pessoal e próximo dos utilizadores.

JET HIRE

Conheça a nossa equipa

Acreditamos no poder da tecnologia para aproximar pessoas e oportunidades. A nossa missão é criar uma plataforma que facilite a ligação entre candidatos e empresas, de forma justa, rápida e eficaz. Conhece quem está por trás desta ideia.



Tomé Almeida
Full-Stack Developer

Sou estudante e desde que comecei a explorar o mundo programação, apaixonei-me por **web development**. Gosto especialmente de trabalhar com **Next.js**. Sinto que com esta ferramenta consigo construir coisas rápidas, agradáveis e à minha maneira. Passo horas a **testar**, a **ajustar**, e a **aprender** mais. Quando não estou a programar, gosto de estar perto de motas ou a jogar uns jogos no computador com os amigos — são duas formas diferentes de desligar e relaxar. O **web development** começou como um interesse, mas hoje já vejo isto como o caminho que quero seguir a sério.

Contactar

Ver perfil no

Figura 99 - Apresentação pessoal

Testemunhos de utilizadores

Na última secção da página, mantive o **slider de testemunhos** com pequenos ajustes visuais para melhor integração no *design*. Todos os testemunhos apresentados são **exemplos fictícios**, criados apenas para fins ilustrativos, uma vez que a aplicação **não está em funcionamento real**.

Os nossos clientes

Confiança constrói-se com experiências reais.
Lê o que os nossos utilizadores têm a dizer.

Graças à plataforma, conseguimos recrutar dois programadores em tempo recorde. A experiência foi simples, eficaz e muito intuitiva.

O processo de candidatura foi super rápido! Submeti o meu perfil e em menos de uma semana já tinha entrevista marcada.

A interface da plataforma é clara e muito bem pensada. Facilitou-nos imenso o processo de triagem de candidatos.



Rita Almeida
Gestora de RH, TechNova



Tiago Correia
Frontend Developer, candidato



João Fernandes
CEO, BrightPath Solutions

Figura 100 - Slider de testemunhos fictícios

Página de chat com AI

A página de **chat com inteligência artificial** segue um *layout* simples e funcional, baseado no componente principal <Layout>. O conteúdo está centrado numa **caixa de chat** com estilo limpo e moderno.

O topo da caixa inclui o título “**Jet AI**”, seguido por uma **área de conversação** com altura fixa, onde as mensagens trocadas são apresentadas em forma de **balões**.

As mensagens do utilizador aparecem alinhadas à direita com fundo azul, enquanto as mensagens do *bot* aparecem à esquerda com fundo cinza claro.

Na parte inferior, existe um **formulário com campo de texto e botão de envio**, permitindo escrever novas mensagens.

Tudo está organizado com *flex* para manter a disposição horizontal entre o *input* e o botão.

Para reforçar o aspeto visual, foram adicionadas duas **imagens decorativas** com posição absoluta: uma no canto inferior esquerdo (como fundo) e outra flutuante no lado direito, com animação suave.

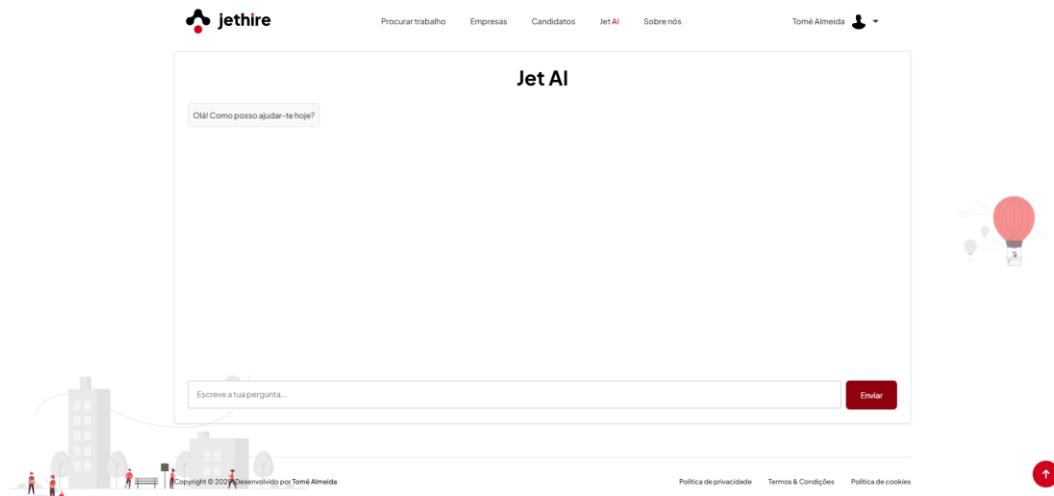


Figura 101 - Página do Jet AI

Registo de utilizadores

A primeira ligação concreta à **base de dados** foi feita através do **formulário de registo**.

No **front-end**, implementei uma **validação simples** para garantir que ambas as palavras-passe coincidiam antes do envio:

```
if (formData.password !== formData.confirmPassword) {
    alert("As palavras-passe não coincidem.");
    return;
}
```

Se a validação for bem-sucedida, os dados são enviados para a **API interna da aplicação** com um pedido POST:

```
await fetch("/api/register", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(formData),
});
```

Tratamento na API

Na rota /api/register, o servidor processa os dados com a seguinte lógica:

- Verifica se já existe um **utilizador** com o mesmo **email**:

```
const userExists = await User.findOne({ email });

if (userExists) {
  return res.status(400).json({ message: "Email já registado." });
}
```

- Encripta a palavra-passe com **bcrypt**:

```
const hashedPassword = await bcrypt.hash(password, 10);
```

- Guarda o novo utilizador na **base de dados**:

```
const newUser = new User({
  name,
  email,
  password: hashedPassword,
});

await newUser.save();
```

Se **tudo correr bem**, o utilizador recebe uma **mensagem de sucesso** e é redirecionado para a página de início de sessão:

```
res.status(201).json({ message: "Conta criada com sucesso." });
```

Se houver **algum erro**, como o email já estar a ser usado ou dados em falta, a aplicação apresenta uma **mensagem de aviso** no ecrã:

```
res.status(400).json({ message: "Erro ao criar conta." });
```

Este processo foi essencial para **entender a comunicação entre o front-end e a base de dados**, e serviu como base para outras funcionalidades como login, gestão de perfis e permissões de utilizador.

Barra de navegação dinâmica

Uma das melhorias mais importantes feitas no *template* original foi a **transformação da barra de navegação num elemento dinâmico e funcional**, adaptado ao estado da sessão do utilizador.

Como funciona?

Para tornar a navegação personalizada, foi utilizado o **hook useSession()** da biblioteca **NextAuth**. Com este *hook*, a aplicação consegue saber se um utilizador está autenticado ou não, e alterar automaticamente os elementos apresentados na barra.

Com sessão ativa

Se houver uma sessão ativa (status === 'authenticated'), a barra mostra:

- o nome do utilizador;
- a sua **imagem de perfil** (caso exista);
- um menu com opções como "**Ver perfil**", "**Definições**" e "**Sair**".

```
{status === 'authenticated' && (  
  <span>{session?.user?.name}</span>  
  <img src={session.user.image} />  
  <button>Ver perfil</button>  
  <button onClick={handleSignOut}>Sair</button>  
)}
```

O botão de "**Sair**" chama a função *handleSignOut()*, que termina a sessão ativa.

Sem sessão ativa

Se o utilizador **não estiver autenticado**, a barra mostra dois links:

- "**criar conta**" (redireciona para o registo);
- "**login**" (leva à página de início de sessão).

```
{status !== 'authenticated' && (
  <>
    <a href="/page-register">Criar conta</a>
    <a href="/page-signin">Login</a>
  </>
)}
```

Este sistema torna a navegação **mais inteligente e adaptável**, mudando conforme o estado do utilizador — sem recarregar a página.

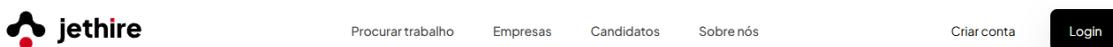


Figura 102 - Barra de navegação sem sessão



Figura 103 - Barra de navegação com sessão

Página de candidatos

A página de candidatos foi desenvolvida com o objetivo de permitir às empresas consultar todos os utilizadores registados na plataforma. Esta funcionalidade inclui **pesquisa em tempo real**, **ordenação por data de registo** e uma apresentação cuidada de cada candidato. O seu desenvolvimento exigiu uma **integração com a base de dados MongoDB**, construção de uma **API personalizada**, **gestão do estado da aplicação**, e uma atenção especial à **experiência do utilizador (UX)**.

Estrutura geral da página

A lógica da página é gerida pelo componente CandidateGrid, dividido em três partes:

- **cabeçalho com estatísticas**: mostra quantos candidatos estão registados;
- **zona de pesquisa e ordenação**: permite procurar por nome e ordenar por data;
- **lista de candidatos**: adaptável ao estado atual da aplicação (carregamento, resultados, ou vazio).

Os estados principais são definidos da seguinte forma:

```
const [users, setUsers] = useState([]);  
const [searchTerm, setSearchTerm] = useState("");  
const [sortOrder, setSortOrder] = useState("recent");
```

Estes estados controlam respetivamente a lista de utilizadores carregados, o termo de pesquisa atual, a ordem de visualização e se os dados estão ainda a ser carregados.

API de candidatos

A API associada (/api/candidates) foi construída como um **endpoint específico de leitura**, usando *mongoose* para aceder à coleção User na base de dados. A sua responsabilidade é **listar todos os utilizadores** que existem na plataforma:

```
if (req.method === 'GET') {  
  try {  
    const users = await User.find();  
    res.status(200).json(users);  
  } catch (error) {  
    res.status(500).json({ error: 'Erro ao procurar utilizadores' });  
  }  
}
```

Desta forma, garanto que esta API serve exclusivamente para **consultar dados**, mantendo a aplicação segura e bem estruturada.

Pesquisa em tempo real

Uma funcionalidade central nesta página é a **pesquisa dinâmica**. O utilizador pode escrever palavras-chave e, conforme digita, a lista de candidatos é filtrada automaticamente. Esta filtragem é feita por nome e **não diferencia maiúsculas de minúsculas**:

```
.filter((user) =>  
  user.name.toLowerCase().includes(searchTerm.toLowerCase())  
)
```

Além disso, o número de resultados é atualizado em tempo real, sendo apresentado de forma clara ao utilizador:

```
<span className="text-small text-showing">  
  A mostrar {filteredUsers.length} de {users.length} candidatos  
</span>
```

Ordenação por data de registo

Outro aspeto relevante é a **possibilidade de ordenar os candidatos** por data. O utilizador pode escolher entre "Mais recente" e "Mais antigo", com a ordenação aplicada ao **array** filtrado:

```
.sort((a, b) => {  
  const dateA = new Date(a.createdAt || a._id);  
  const dateB = new Date(b.createdAt || b._id);  
  return sortOrder === "recent" ? dateB - dateA : dateA - dateB;  
});
```

A seleção é feita através de um menu *dropdown*, com *feedback* visual para a opção ativa:

```
<li>  
  <a  
    className={`dropdown-item ${sortOrder === "recent" ? "active" : ""}`}  
    onClick={() => setSortOrder("recent")}>
```

Mais recente

```
</a>
</li>
```

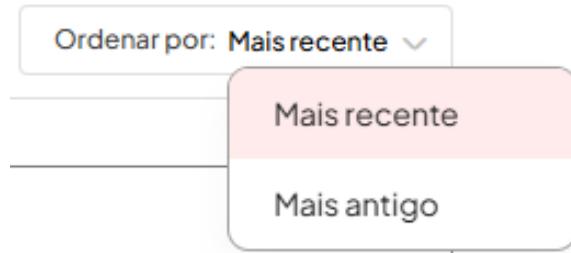


Figura 104 - Dropdown da página de candidatos

A mostrar 5 de 5 candidatos

Ordenar por: Mais recente

 Relatório À procura de emprego Sem descrição. Ver Perfil	 Lara Economista oi Cantanhede, Portugal Ver Perfil	 Tomé Almeida Full-Stack Developer Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed convallis mauris quis arcu ultrices, sed vestibulum nibh rh... HTML5 & CSS3 JavaScript React.js Photoshop Murtede, Portugal Ver Perfil	 Tome À procura de emprego Sem descrição. Ver Perfil
---	---	--	--

Figura 105 - Exemplo de pesquisa por "Mais recente"

Renderização condicional com mensagens de *fallback*

A apresentação da lista de candidatos utiliza o sistema de **renderização condicional com estado de carregamento**, já descrito na secção de **Elementos Gerais**.

Neste caso específico, a interface adapta-se automaticamente a **três cenários distintos**:

- **Enquanto os dados estão a ser carregados**, é apresentado um **spinner** com a mensagem “A carregar candidatos...”.



Figura 106 - Candidatos a carregar

- **Se não houver resultados** após a pesquisa, é mostrada uma mensagem como “Nenhum candidato encontrado.”

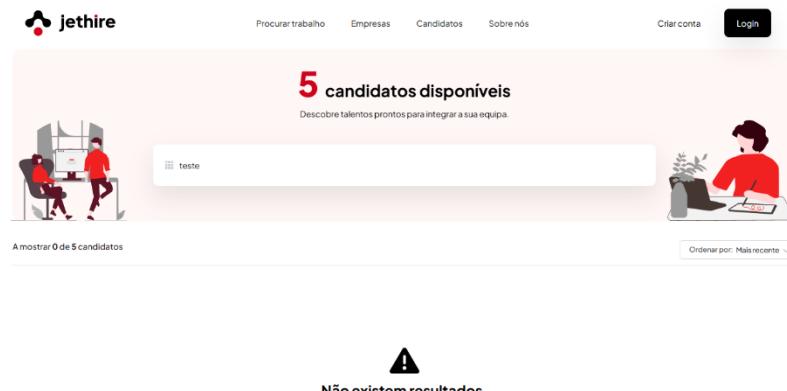


Figura 107 - Sem resultado de candidatos

- Quando existem candidatos válidos, é renderizada a lista de cartões com os dados reais.

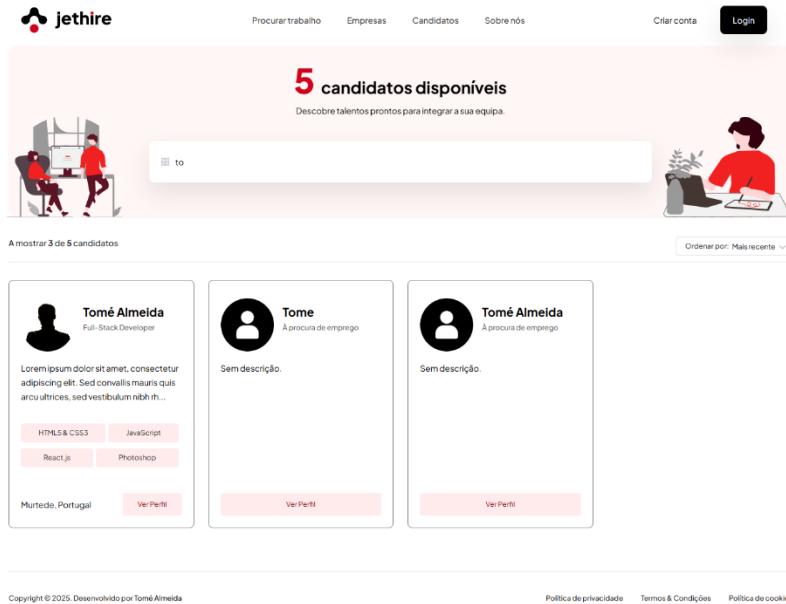


Figura 108 - Pesquisa de candidatos válida

Este tipo de renderização condicional garante uma experiência visual sempre adequada ao contexto, sem deixar o utilizador com ecrãs vazios ou dúvidas.

Perfil dinâmico e adaptável

Uma das partes mais relevantes da aplicação foi a criação de um **perfil de utilizador dinâmico**, que se adapta automaticamente ao conteúdo disponível na base de dados. Em vez de mostrar sempre a mesma estrutura fixa, a página de perfil **ajusta-se conforme os dados do utilizador**, garantindo uma melhor experiência e visual mais limpo.

Estrutura da página

A página está dividida em duas secções principais:

- **Mainbar** (conteúdo principal) – apresenta:
 - secção “**Sobre mim**”;
 - **competências profissionais**;

- **experiência profissional;**
- **educação;**
- projetos de **portfólio;**
- **Sidebar** (barra lateral) – inclui:
 - visão geral do perfil (anos de experiência, línguas, educação, etc.);
 - ligações para **LinkedIn**, **GitHub** ou **site pessoal**;
 - contactos e localização;
 - alerta de perfil incompleto (caso aplicável).

Esta separação permite manter a informação bem organizada e facilitar a leitura por empresas ou outros utilizadores.

Carregamento dinâmico com `getServerSideProps`

Para garantir que os dados aparecem logo ao abrir a página, utilizei a função `getServerSideProps()` do Next.js. Este método é executado no **servidor**, sempre que alguém acede ao perfil, e vai **buscar os dados à base de dados MongoDB** com base no ID do URL.

```
export async function getServerSideProps(context) {  
  const { id } = context.params;  
  await connectMongoDB();  
  const user = await User.findById(id).lean();  
  
  const result = checkProfileCompletion(user.profile);  
  return {  
    props: {  
      user: JSON.parse(JSON.stringify(user)),  
      profileComplete: result.complete,  
      profilePercentage: result.percentage  
    }  
  };  
}
```

Este método também contribui para melhorar o SEO e a performance da aplicação.

Renderização condicional

Como nem todos os perfis estão completos, implementei lógica condicional para mostrar apenas o conteúdo relevante. Por exemplo:

```
{mainPercentage !== 0 ? (
  <div>/* Secções como "Sobre mim", "Competências"... *</div>
) : (
  <div className="profile-noreviews">
    <h5>Este utilizador ainda não configurou o seu perfil...</h5>
  </div>
)}
```



Figura 109 - Perfil completo

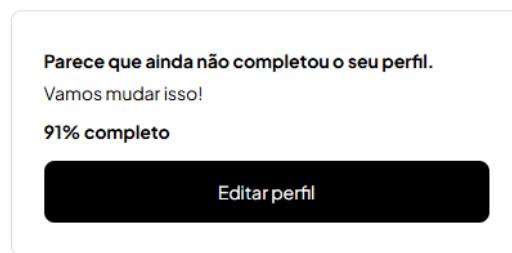


Figura 110 - Perfil por concluir

Além disso, o *layout* adapta-se automaticamente consoante a existência ou não de conteúdo na **sidebar**:

```
<div className={sidebarPercentage == 0
  ? "col-lg-12"
  : "col-lg-8"}>
```

Percentagem de preenchimento

Para incentivar o utilizador a completar o perfil, implementei funções que analisam se os campos obrigatórios estão preenchidos. Um exemplo:

```
const requiredFields = ["city", "country", "aboutMe", "skills"];
const filled = requiredFields.filter(field => profile[field]).length;
```

```
const percentage = Math.round((filled / requiredFields.length) * 100);
```

Depois, essa percentagem é apresentada na interface:

```
{!profileComplete ? (
    <p>{profilePercentage}% completo — ainda falta configurar o perfil.</p>
) : (
    <p>O perfil está completo!</p>
)}
```

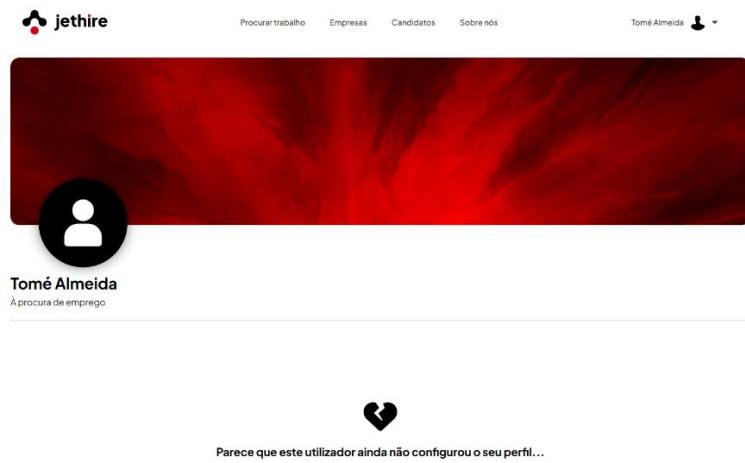


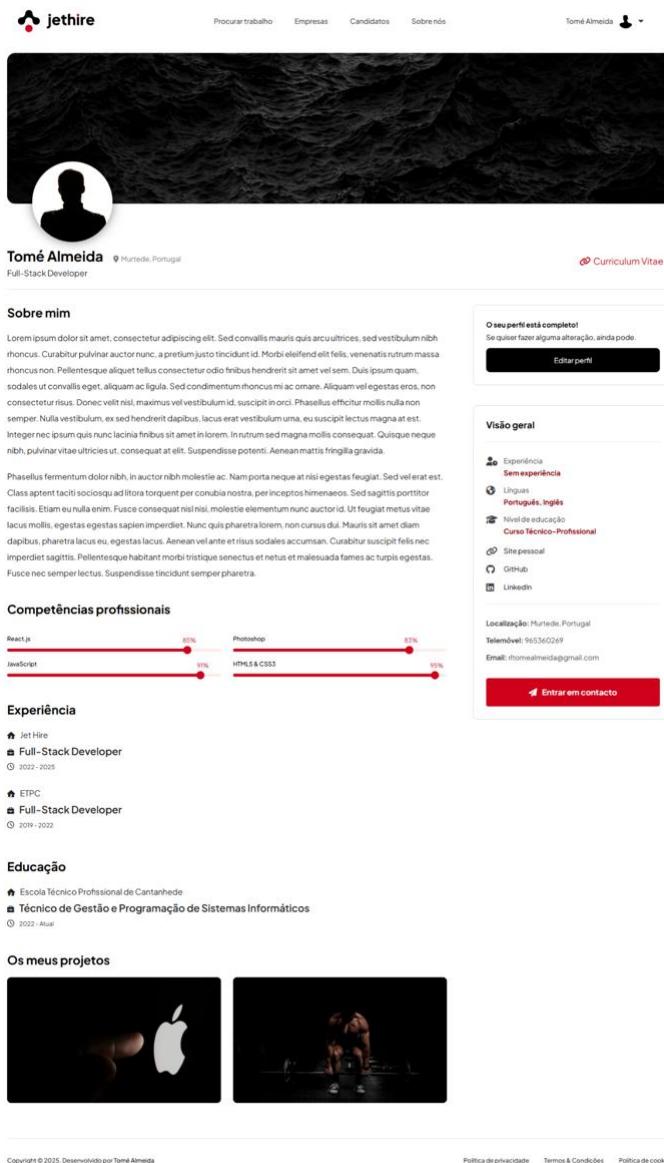
Figura 111 - Exemplo de perfil por concluir

Segurança e personalização

Por segurança, apenas o próprio utilizador pode editar o seu perfil. A verificação é feita com base no ID da sessão:

```
{session?.user?.id === user._id && (
    <a href="/edit">Editar perfil</a>
)}
```

Como os dados vão ser procurados pelos dados da sessão, o utilizador apenas consegue ver o seu próprio perfil e não tem hipótese de editar qualquer informação de outro utilizador a não ser que inicie sessão na conta do mesmo.



The screenshot shows a profile page for Tomé Almeida. At the top, there's a navigation bar with links for 'Procurar trabalho', 'Empresas', 'Candidatos', and 'Sobre nós'. A user icon for Tomé Almeida is also present. Below the navigation is a large dark rectangular area containing a placeholder image of a person's face. Underneath this is a circular profile picture of Tomé Almeida.

Tomé Almeida • Murtedo, Portugal
Full-Stack Developer

Sobre mim

A large block of placeholder text (Lorem ipsum) follows, detailing Tomé's experience and skills.

Competências profissionais

Tecnologia	Nível
React.js	80%
JavaScript	95%
Photoshop	85%
HTML5 & CSS3	95%

Experiência

- Jet Hire • Full-Stack Developer • 2022 - 2025
- ETPC • Full-Stack Developer • 2019 - 2022

Educação

- Escola Técnico Profissional de Cantanhede • Técnico de Gestão e Programação de Sistemas Informáticos • 2022 - Atual

Os meus projetos

Two thumbnail images are shown: one of a hand interacting with a glowing Apple logo, and another of a person working at a desk.

At the bottom of the page, there are footer links for 'Copyright © 2025. Desenvolvido por Tomé Almeida.', 'Política de privacidade', 'Termos & Condições', and 'Política de cookies'.

Figura 112 - Exemplo de perfil completo

Edição de perfil

A página de edição de perfil foi, sem dúvida, **uma das mais exigentes de todo o projeto**. A estrutura não veio de nenhum *template* — foi **desenvolvida do zero**, com o objetivo de dar ao utilizador total controlo sobre o seu perfil, numa interface clara, organizada e adaptável a qualquer ecrã.

Organização modular

O formulário foi dividido em vários componentes **React independentes**, como **BasicInfo**, **Education**, **Experience**, **Skills**, **Languages**, **Portfolio**, entre outros. Isto permitiu:

- separar responsabilidades e manter o código limpo;
- facilitar validações específicas por secção;
- permitir reaproveitamento de lógica, por exemplo entre **Experience** e **Education**.

Gestão do estado e validação

Todos os dados do utilizador são carregados ao iniciar a página, através de uma chamada protegida à API (getServerSession) com validação da sessão atual. O estado global do formulário é gerido com:

```
const [formData, setFormData] = useState<UserData>(user);
```

Cada alteração num campo é registada com:

```
const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  setFormData({ ...formData, [e.target.name]: e.target.value });
};
```

Ao submeter o formulário, é feita uma verificação das alterações antes de enviar os dados, comparando o novo estado com o original. Isto evita chamadas desnecessárias à API e melhora o desempenho da aplicação.

Portfólio com *upload presets*

O portfólio é uma das secções mais visuais e, ao mesmo tempo, **uma das mais técnicas**. Cada item inclui uma **imagem enviada para a cloud**, um **link** e uma **legenda**. Para tornar o processo mais eficiente, usei um **upload preset do Cloudinary**, que garante que todas as imagens:

- são enviadas para a pasta certa;
- são redimensionadas automaticamente;
- estão sempre associadas ao projeto certo.

```
const formData = new FormData();
formData.append("file", file);
formData.append("upload_preset", "portfolio");

await fetch("https://api.cloudinary.com/v1_1/.../upload", {
  method: "POST",
  body: formData,
});
```

Envio final e atualização da base de dados

Após todas as edições, os dados são enviados para a API com um **POST**, e o perfil é atualizado na base de dados.

Para garantir uma certa consistência, os campos que não sofreram alterações são mantidos, e os recursos antigos (como imagens substituídas) são **automaticamente removidos do Cloudinary** para não ocupar espaço desnecessário.

Página de empresas

A **página de empresas** tem como principal objetivo apresentar todas as **empresas ativas** registadas na plataforma, permitindo que os utilizadores possam **explorar o mercado** e conhecer potenciais entidades recrutadoras. Para além disso, permite que novas empresas possam **submeter o seu pedido de registo**, através de um formulário próprio.

Listagem de empresas e pesquisa em tempo real

Logo ao abrir a página, o sistema faz uma **chamada ao back-end** para obter a lista de todas as empresas com estado ativo. Esta consulta vai diretamente à base de dados MongoDB e retorna apenas os registos que tenham o campo *isActive* definido como verdadeiro. No *back-end*, o funcionamento é simples e direto:

```
const companies = await Company.find({ isActive: true });
```

Estes dados são depois enviados para o *front-end*, onde são apresentados visualmente através de **cartões individuais**, com informações como **nome, logótipo, área de atuação e localização**.

A funcionalidade de **pesquisa ao vivo** (live search) permite que o utilizador filtre esta lista instantaneamente, sem necessidade de novos pedidos ao servidor. À medida que se escreve no campo de pesquisa, o sistema filtra o *array* recebido inicialmente, comparando o termo com o **nome da empresa** ou com as suas **tags**:

```
company.name.includes(termo) || company.tags.includes(termo)
```

Este sistema torna a experiência muito **fluida**, garantindo uma resposta rápida e sem recarregamentos. Se não forem encontradas empresas, é apresentada uma **mensagem clara** ao utilizador.

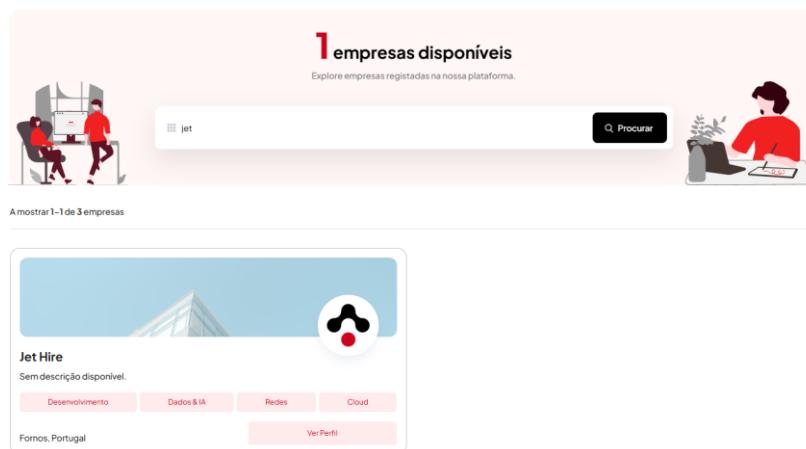


Figura 113 - Exemplo de pesquisa de empresa

Pedido de Registo de Empresa

Para empresas que ainda não estão na plataforma, existe a possibilidade de **submeter um pedido de registo**. Ao clicar no botão disponível na página, o utilizador é redirecionado para um **formulário de candidatura**, onde deve preencher informações básicas como:

- **nome da empresa;**
- **email de contacto;**

- **localização;**
- **pessoa responsável pelo registo.**

Após o envio do formulário, os dados são validados e enviados para o *back-end*, onde são guardados como um **pedido pendente**, utilizando um modelo próprio (*CompanyRequest*). Esta criação **não ativa imediatamente a empresa** — trata-se apenas de um **registo temporário**, que mais tarde será revisto e validado por um administrador.

```
await CompanyRequest.create({  
    name: data.name,  
    email: data.email,  
    location: data.location,  
    responsible: data.responsible,  
});
```

Este sistema garante um maior controlo sobre os registos e permite manter a plataforma restrita a **entidades legítimas e validadas**.

Página de detalhes de empresas

Ao clicar numa das empresas listadas na página principal, o utilizador é redirecionado para uma **página de perfil público**, onde pode consultar toda a **informação detalhada** sobre essa entidade. O objetivo desta página é apresentar de forma clara a **identidade da empresa**, reforçando a sua presença no mercado e permitindo que qualquer utilizador a conheça melhor.

Obtenção e apresentação dos dados

Assim que a página é carregada, o sistema faz um **pedido ao back-end** com o id da empresa, para obter os seus dados. O *back-end* procura esse registo na base de dados MongoDB com base no identificador único:

```
const company = await Company.findById(companyId);
```

O objeto devolvido contém todos os campos públicos da empresa, incluindo:

- **nome;**
- **imagem de capa;**
- **logótipo;**
- **slogan e descrição;**
- **localização** (cidade e país);
- **etiquetas (tags);**

Estes dados são recebidos no *front-end* e apresentados de forma estruturada e elegante.

Dados públicos

Esta página apresenta **apenas dados públicos** e não permite qualquer tipo de edição. A informação sensível (como membros da equipa, contactos internos, ou estatísticas) não é visível nem acessível a partir deste perfil — sendo reservada para a **área privada da empresa**, na *dashboard*.

Página de ofertas

A **página de ofertas** é o espaço onde os candidatos podem explorar todas as **oportunidades de emprego disponíveis** na plataforma. Esta página foi pensada para permitir uma **pesquisa rápida**, uma **navegação agradável** e um **controlo total sobre os resultados apresentados**.

Obtenção de ofertas

Logo ao carregar a página, é feito um **pedido ao back-end** para obter todas as ofertas ativas:

```
const offers = await Offer.find({ isActive: true });
```

Estes dados incluem as informações mais importantes como:

- **função (role);**
- **tipo de contrato** (tempo inteiro, part-time, estágio);

- **modalidade** (remoto, presencial, híbrido);
- **experiência requerida;**
- **intervalo salarial;**
- **tags e setor;**
- **dados da empresa associada.**

As ofertas são então renderizadas no *front-end* em **cartões visuais**, sendo todos os filtros e interações aplicados **no lado do cliente**.

Filtros avançados

Para oferecer uma experiência mais completa e ajustada às necessidades dos candidatos, a página de ofertas inclui um sistema de **filtros avançados**, que permite refinar os resultados de forma precisa. Estes filtros são aplicados **localmente** no *front-end*, com base nos dados recebidos inicialmente do *back-end*. Isto significa que **não é feita nenhuma chamada adicional ao servidor** — todos os filtros funcionam com o *array* já carregado.

O sistema foi pensado para ser **modular, combinável e eficiente**. O utilizador pode aplicar **um ou vários filtros em simultâneo**, e a lista de ofertas atualiza-se automaticamente.

Os filtros disponíveis são:

- **área de atuação** (ex: *Design*, Programação, *Marketing*, etc.);
- **tipo de contrato** (*fulltime*, *parttime*, *internship*);
- **modalidade** (*remote*, *presencial*, *hybrid*);
- **experiência exigida** (de < 1 ano até > 5 anos);
- **intervalo salarial mínimo e máximo;**
- **nível** (*junior*, *mid*, *senior*);
- **tags específicas** (como *React*, *UI*, *Node*, etc.);
- **menor de idade** (ou não);

Lógica de filtragem combinada

A filtragem é feita com uma lógica que verifica **todas as condições em simultâneo**, garantindo que só aparecem ofertas que correspondem a **todos os critérios ativos**. A estrutura da verificação segue este padrão:

```
const matchTags = selectedTags.every(tag => offer.tags.includes(tag));  
  
return (  
  matchTags && ...  
);  
});
```

Cada verificação individual é opcional — ou seja, se o utilizador **não selecionar esse filtro**, ele **não é aplicado**. Isto dá liberdade total ao utilizador para filtrar com base num ou vários critérios, e torna a pesquisa muito mais eficaz.

Exemplo prático de combinação de filtros

Se o utilizador selecionar:

- área: **Design**;
- tipo: **Part-time**;
- experiência: **1 a 2 anos**;
- tag: **Figma**.

A lógica irá procurar apenas ofertas onde:

- `offer.area === "design"`
- `offer.type === "parttime"`
- `offer.experience === "1_to_2"`
- e o array `offer.tags` inclui "figma"

Caso o utilizador adicione também um intervalo de salário, esse critério será também aplicado em conjunto.

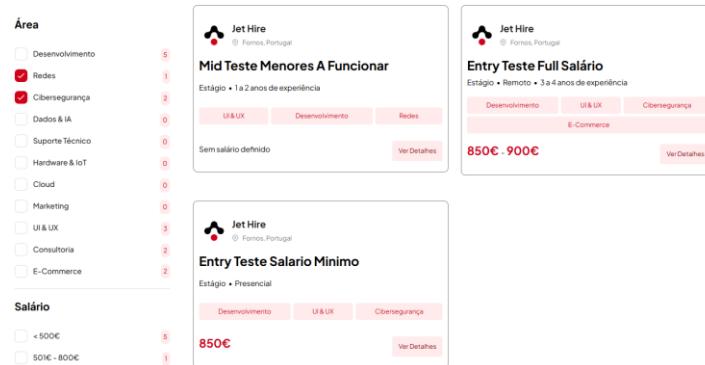


Figura 114 - Exemplo de filtragem combinada

Filtro “Mostrar apenas favoritos”

Uma das funcionalidades disponíveis na página de ofertas é o filtro “**Mostrar apenas favoritos**”, que permite ao utilizador **ver rapidamente as ofertas que marcou como favoritas**.

Este filtro utiliza os dados do próprio utilizador autenticado, mais concretamente o array `savedOffers`, que contém os identificadores (`_id`) das ofertas que ele guardou anteriormente.

Quando este filtro está ativo, a lógica de filtragem compara os IDs das ofertas listadas com os que estão no array de favoritos do utilizador:

```
const filtered = offers.filter((offer) =>
  user.savedOffers.includes(offer._id)
);
```

Desta forma, são apresentadas apenas as ofertas que o utilizador já guardou. Caso o filtro esteja desativado, todas as ofertas voltam a ser visíveis, respeitando os restantes critérios selecionados (tipo, área, experiência, etc.).



Figura 115 - A mostrar favoritos

Página de detalhes das ofertas

A página de detalhes de cada oferta inclui várias funcionalidades interativas no *front-end* que melhoraram significativamente a experiência do utilizador. Entre elas, destacam-se o **cálculo automático do tempo de publicação**, a gestão de **favoritos** e o sistema de **candidaturas com confirmação**.

Tempo de publicação

Para apresentar a data de forma mais humana e contextualizada, criei uma função que calcula automaticamente há quanto tempo a oferta foi publicada. O tempo é mostrado com expressões como:

- “**Publicado agora mesmo**”;
- “**Publicado há 5 minutos**”;
- “**Publicado há 2 horas**”;
- “**Publicado há 3 dias**”.

Este cálculo é feito no *front-end*, a partir da *createdAt* da oferta:

```
const created = new Date(offer.createdAt);
const now = new Date();
const diffMs = now - created;
const diffMin = Math.floor(diffMs / 60000);
const diffHr = Math.floor(diffMin / 60);
const diffDays = Math.floor(diffHr / 24);
```

 **Publicado há 8 dias**

Figura 116 - Exemplo do tempo de publicação

Candidaturas com modal de confirmação

Implementei um botão “**Candidatar-me**”, que verifica se o utilizador já se candidatou e atualiza o texto para “**Remover candidatura**”, caso aplicável. A candidatura ou remoção é confirmada num **modal visual** antes de ser executada, prevenindo cliques acidentais.


Candidatar-me

Figura 117 - Botão para se candidatar


Remover candidatura

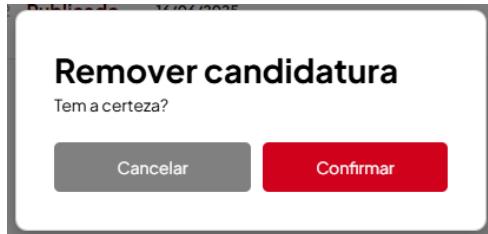
Figura 118 - Botão para remover a candidatura


Confirmar candidatura
 Quer mesmo candidatar-se a esta oferta?

Cancelar


Confirmar

Figura 119 - Modal de confirmação de candidatura


Remover candidatura
 Tem a certeza?

Cancelar


Confirmar

Figura 120 - Modal de confirmação de remoção

Favoritos com estrela dinâmica

Ao lado do botão de candidatura, inclui um botão de **favoritos** com ícone de estrela:

- **Estrela vazia:** a oferta ainda não está nos favoritos.
- **Estrela cheia:** a oferta já está guardada.

Com um clique, o utilizador pode adicionar ou remover a oferta da sua lista pessoal de favoritos. O estado visual muda instantaneamente, refletindo a ação.

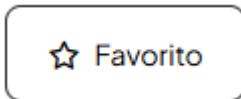

Favorito

Figura 121 - Botão para tornar favorito


Favorito

Figura 122 - Botão para remover favorito

Página de contactos

O formulário da página de contacto foi desenvolvido para permitir que qualquer utilizador envie uma **mensagem diretamente para a equipa da Jet Hire**. Para garantir a **fiabilidade e segurança dos dados**, o sistema realiza **validações obrigatórias no back-end**, antes de guardar qualquer informação.

Assim que o utilizador submete o formulário, é feito um pedido POST para o servidor com os seguintes campos:

- **name** – nome da pessoa que está a contactar;
- **company** – nome da empresa (opcional);
- **email** – endereço de email;
- **phone** – contacto telefónico (opcional);
- **message** – mensagem escrita.

No **back-end**, o sistema faz a **ligação à base de dados MongoDB** com `connectMongoDB()` e aplica uma verificação simples mas eficaz:

```
if (!name || !email || !message) {  
    return res.status(400).json({ error: "Preencha todos os campos obrigatórios." });  
}
```

Isto garante que **nome**, **email** e **mensagem** são sempre fornecidos. Os campos **company** e **phone** são opcionais e só serão guardados se existirem.

Se todos os campos obrigatórios estiverem presentes, a mensagem é **guardada numa coleção chamada Contact**, através do modelo `Contact`:

```
const newMessage = new Contact({ name, company, email, phone, message });  
await newMessage.save();
```

Jet AI

A página da Jet AI foi desenvolvida para permitir a integração de um **assistente virtual inteligente**, utilizando a **API da OpenAI (GPT)**. O objetivo deste assistente é **responder a dúvidas sobre o mercado de trabalho em informática**, ajudar o utilizador a **navegar na aplicação Jet Hire e dar conselhos práticos** relacionados com a área profissional.

Funcionamento geral

A comunicação entre o utilizador e o *bot* segue os seguintes passos:

- o utilizador escreve uma pergunta e clica em "Enviar";
- a mensagem é enviada para o servidor (/api/chatbot);
- o servidor envia a mensagem para a API da OpenAI com um *prompt* personalizado;
- o assistente gera uma resposta com base na pergunta e no contexto;
- a resposta é enviada de volta e apresentada na interface;

Envio de mensagem

No *front-end*, assim que o utilizador submete o formulário, é feito um *fetch* para a rota /api/chatbot, com o texto da mensagem:

```
const res = await fetch("/api/chatbot", {  
    method: "POST",  
    headers: { "Content-Type": "application/json" },  
    body: JSON.stringify({ message: input }),  
});
```

Se a resposta for válida, é adicionada à conversa com:

```
const data = await res.json();  
setMessages((prev) => [...prev, { sender: "bot", text: data.reply }]);
```

Lógica da API

No *back-end*, a rota /api/chatbot recebe a mensagem do utilizador e utiliza a biblioteca *openai* para comunicar com o modelo **GPT-4 ou GPT-3.5-turbo**.

Exemplo real de API (pages/api/chatbot.js):

```
import { Configuration, OpenAIApi } from "openai";  
  
const configuration = new Configuration({  
    apiKey: process.env.OPENAI_API_KEY,  
});  
const openai = new OpenAIApi(configuration);
```

```
export default async function handler(req, res) {
  if (req.method !== "POST") {
    return res.status(405).json({ error: "Método não permitido" });
  }

  const { message } = req.body;

  if (!message || message.trim() === "") {
    return res.status(400).json({ error: "Mensagem inválida" });
  }

  try {
    const completion = await openai.createChatCompletion({
      model: "gpt-3.5-turbo",
      messages: [
        {
          role: "system",
          content:
            "Estás a atuar como assistente virtual na plataforma Jet Hire. Responde a dúvidas sobre empregos em informática, ajuda os utilizadores a navegar na plataforma e dá conselhos úteis de forma simpática.",
        },
        { role: "user", content: message },
      ],
    });

    const reply = completion.data.choices[0].message.content;
    return res.status(200).json({ reply });
  } catch (err) {
    console.error("Erro ao comunicar com a OpenAI:", err);
    return res.status(500).json({ error: "Erro ao gerar resposta" });
  }
}
```

Comportamento inteligente

Graças ao uso da role: "system", o assistente tem um comportamento personalizado.

Pode:

- ajudar o utilizador a **procurar ofertas**;
- explicar a diferença entre áreas como **Front-End, Back-End** ou **Cibersegurança**;
- indicar como **editar o perfil, candidatar-se** ou **consultar empresas**;
- dar dicas para melhorar o **currículo, preparar entrevistas**, ou **começar a carreira**;

Exemplo de interação:

Utilizador: "Qual a melhor linguagem para começar no desenvolvimento web?"

Bot: "Uma excelente opção é começar com **JavaScript**, pois é essencial para o desenvolvimento **Front-End**. Depois, podes explorar frameworks como **React** ou **Next.js**, muito procuradas no mercado."

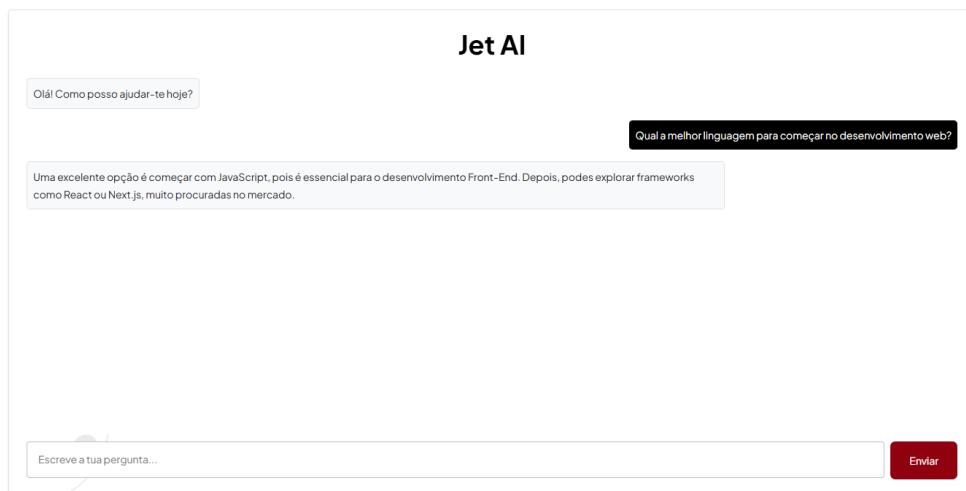


Figura 123 - Exemplo de interação com o Jet AI

Dashboard

A *dashboard* foi desenvolvida como uma área **restrita e funcional**, destinada exclusivamente a dois tipos de utilizadores: **administradores da aplicação e membros de empresas**. Todo o seu desenvolvimento teve como base três princípios fundamentais:

- **segurança de acesso;**
- **separação clara de permissões;**
- **modularidade na gestão.**

A *dashboard* foi desenvolvida com **Next.js 15**, utilizando a nova abordagem baseada em **App Router (app/)**, **React Server Components** e **TypeScript**. A interface é construída com **TailwindCSS**, garantindo responsividade e consistência visual. A autenticação é feita com **NextAuth**.

Tal como na página principal, a base de dados continua a ser o **MongoDB**, gerida com **Mongoose**. No entanto, para o **sistema de chat em tempo real**, foi integrada a **Firebase Realtime Database**, de forma a garantir **comunicação instantânea entre os membros da mesma empresa**.

Middleware de proteção

Para garantir que nenhuma página sensível fosse acedida sem autenticação, foi criado desde o início um **middleware global**:

```
const PUBLIC_PATHS = ['/signin', '/signup', '/invite'];

export async function middleware(req: NextRequest) {
  const { pathname } = req.nextUrl;

  const isPublic = PUBLIC_PATHS.some(path =>
    pathname === path || pathname.startsWith(path + '/')
  );
  if (isPublic) return NextResponse.next();
}
```

```
const token = await getToken({ req, secret: process.env.NEXTAUTH_SECRET });

if (!token) {
    return NextResponse.redirect(new URL('/signin', req.url));
}

return NextResponse.next();
}
```

Este sistema garante assim que todas as rotas da *dashboard* exigem uma **sessão ativa**, apenas as páginas */signin*, */signup*, */invite* são acessíveis livremente.

Estrutura de dados

Administradores

O modelo de administrador é simples, contendo apenas os campos essenciais à gestão:

```
const administratorSchema = new mongoose.Schema({
    user: String,
    name: String,
    email: String,
    password: String,
    role: { type: String, default: 'jethire-admin' },
    isActive: { type: Boolean, default: true },
    socials: [{ platform: String, url: String }]
});
```

Este utilizador **não tem funções públicas**, servindo exclusivamente para **moderar e validar pedidos**.

Empresas

O modelo da empresa é mais complexo e flexível:

- múltiplos membros com funções distintas (*admin*, *recruiter*, *manager*);
- estrutura hierárquica com equipa (*team*) e convites pendentes (*pending*);
- secções de descrição da empresa;
- estado de ativação e imagens institucionais.

```
const companySchema = new Schema({
  name: String,
  city: String,
  contact: { email: String, phone: String },
  team: [userSchema],
  pending: [pendingSchema],
  description: [descriptionSchema],
  tags: [String],
  isActive: { type: Boolean, default: true }
});
```

Este modelo permite **adaptar-se a qualquer empresa**, desde uma *startup* com 2 membros até uma equipa de RH completa.

Página de erro (404)

Para garantir uma experiência de navegação mais cuidada e profissional, foi criada uma página de erro personalizada para a *dashboard* da Jet Hire. Esta página surge sempre que o utilizador tenta aceder a uma rota que não existe ou que foi removida.

A estrutura baseia-se num fundo limpo com elementos gráficos, utilizando **imagens distintas para modo claro e escuro**, o que assegura consistência com o resto da interface.

A mensagem exibida é direta, mas acolhedora. Logo abaixo, é apresentado um botão visível para regressar à página principal da *dashboard*, permitindo ao utilizador retomar rapidamente a navegação normal.

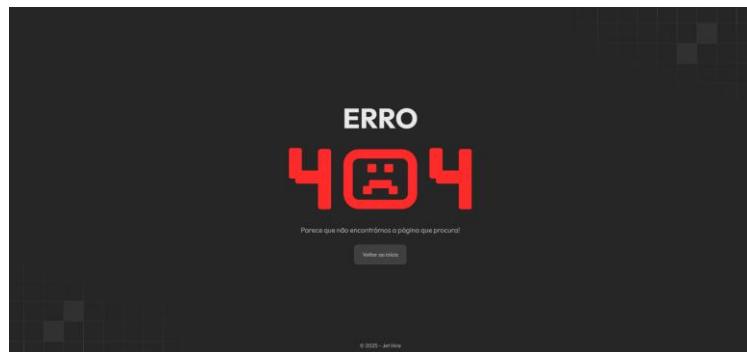


Figura 124 - Página 404

Barra lateral

A **barra lateral** é o principal componente de navegação dentro da *dashboard*. Foi desenvolvida para ser **modular, dinâmica e segura**, adaptando o conteúdo apresentado ao tipo de utilizador que está autenticado.

Objetivos da *sidebar*

A barra lateral tem como objetivo:

- **organizar o acesso às páginas principais da aplicação;**
- **restringir o conteúdo visível com base na role do utilizador;**
- **facilitar a navegação com submenus e ícones;**
- **adaptar-se a diferentes tamanhos de ecrã (responsiva).**

Funcionamento base

A estrutura da *sidebar* é composta por uma lista de itens (*navItems*), onde cada entrada representa um **link direto** ou um **menu com subitens**. Cada item pode conter:

- um **ícone**;
- um **nome**;
- um **caminho (path)**;
- e opcionalmente: uma lista de **roles permitidas**.

O código verifica automaticamente o tipo de utilizador através da sessão (`session.user.role`) e mostra apenas os itens que ele pode ver.

Sistema de permissões

Cada item da *sidebar* pode ser restrito a determinadas **funções (roles)**. Estas são as funções existentes:

- **jethire-admin** – administrador da plataforma;
- **admin** – administrador da empresa;
- **manager** – gestor de recrutamento;

- **recruiter** – recrutador da equipa.

A lógica de verificação é feita com:

```
const hasAccess = (roles?: Role[]) => {
  if (!roles) return true;
  return roles.includes(role);
};
```

Se um item **não tiver roles definidos**, é considerado **público dentro da dashboard**.

Caso contrário, só será visível para as roles permitidas.

Exemplo: declarar quem pode ver o quê

A visibilidade de cada item é definida diretamente nos dados do menu. Aqui estão alguns exemplos práticos:

```
{
  name: "Caixa de entrada",
  path: "/inbox",
  icon: <Inbox />,
  roles: ["jethire-admin"]
}
{
  name: "Empresas",
  icon: <Building2 />,
  roles: ["jethire-admin"],
  subItems: [
    { name: "Lista", path: "/company/list", roles: ["jethire-admin"] },
    { name: "Registros", path: "/company/requests", roles: ["jethire-admin"] }
  ]
}
```

Este sistema torna extremamente fácil **adicionar ou limitar o acesso a cada secção da dashboard**, bastando definir as funções autorizadas.

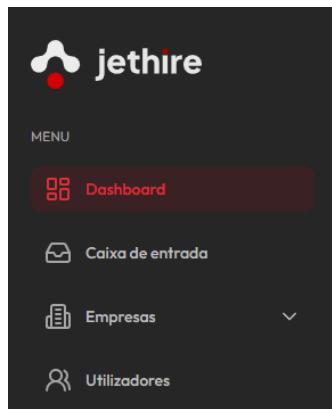


Figura 125 - Barra lateral de administrador

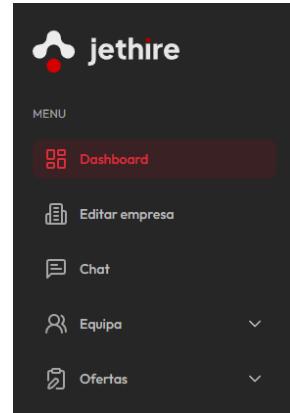


Figura 126 - Barra lateral de empresa

Estrutura do menu

Cada secção é gerada com a função `renderMenuItem(...)`, que percorre a lista e mostra:

- links diretos (como “`Dashboard`”);
- botões que abrem **submenus com animação**, como:
 - **empresas** → lista / registos;
 - **equipa** → ver / adicionar;
 - **ofertas** → ver / adicionar.

Submenus dinâmicos

Se o item tiver `subItems`, a `sidebar` calcula **automaticamente a altura** do submenu com base no conteúdo, para aplicar animações suaves ao abrir e fechar:

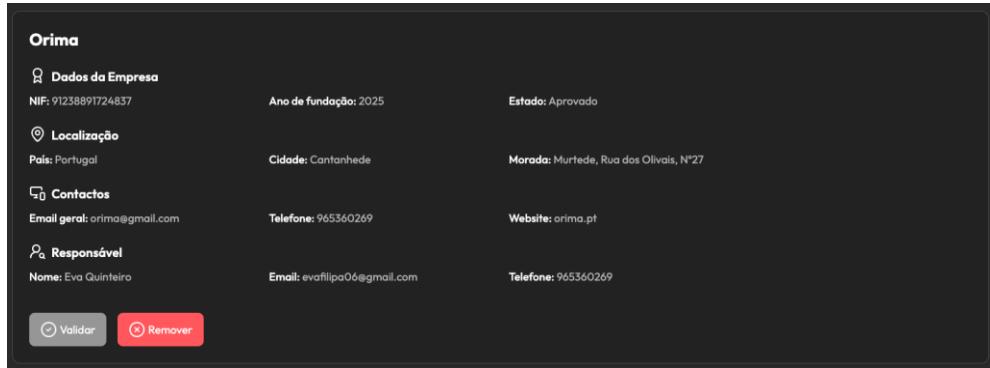
```
setSubMenuHeight((prevHeights) => ({
  ...prevHeights,
  [key]: subMenuRefs.current[key]?.scrollHeight || 0,
}));
```

Esta lógica garante uma navegação **fluida e visualmente apelativa**, sem quebras ou recortes abruptos.

Registo de empresas

O registo de uma nova empresa inicia-se com o preenchimento de um **formulário de candidatura**. Após o envio, os dados são guardados na base de dados como um pedido pendente.

Esse pedido é apresentado na *dashboard* como no seguinte exemplo:



The screenshot shows a registration form for a company named 'Orima'. The form is divided into sections: 'Dados da Empresa' (NIF: 9123889724837, Ano de fundação: 2025, Estado: Aprovado), 'Localização' (País: Portugal, Cidade: Cantanhede, Morada: Murtede, Rua das Olivas, N°27), 'Contactos' (Email geral: orima@gmail.com, Telefone: 965360269, Website: orima.pt), and 'Responsável' (Nome: Eva Quinteiro, Email: evafilipa06@gmail.com, Telefone: 965360269). At the bottom are two buttons: 'Validar' (in grey) and 'Remover' (in red).

Figura 127 - Pedido de registo de empresa

Os administradores podem então validar ou recusar o pedido através da *dashboard*.

Integração com o Resend

O **Resend** foi usado para enviar **emails automáticos** e profissionais.

O que é o Resend?

O **Resend** é uma plataforma especializada no envio de emails transacionais. Foi utilizada neste projeto para automatizar o envio de **tokens de acesso** e **convites para membros de equipa**, de forma segura e profissional.

Preparação

Para utilizar o Resend foi necessário:

- criar uma conta;
- associar o domínio da aplicação à conta Resend;
- inserir a **API Key** no ficheiro .env da aplicação.

Nota: O envio de emails só funciona quando a aplicação está alojada num domínio válido, e não localmente.

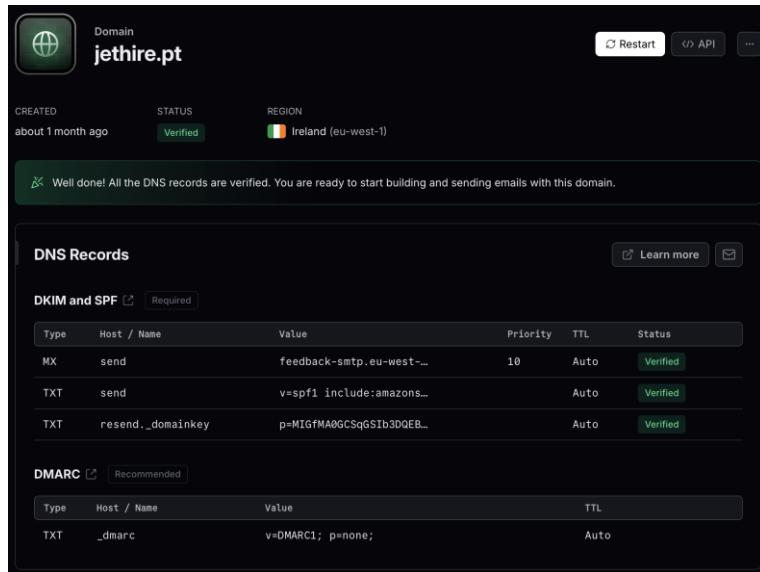


Figura 128 - Domínios do Resend

Processo de validação

Quando um administrador **valida uma empresa**, é gerado automaticamente um **token de ativação**, e enviado um email com esse *token* para o responsável da empresa. Esse email permite completar o registo através de uma página exclusiva */signup*.

Geração do *token*

```
const rawToken = Math.random().toString(36).substring(2);
const hashedToken = await bcrypt.hash(rawToken, 10);
company.activationToken = hashedToken;
```

Envio do email com o componente TokenEmail

```
await resend.emails.send({
  from: process.env.RESEND_FROM!,
  to: company.responsibleEmail,
  subject: `A sua empresa ${company.companyName} foi validada!`,
  react: TokenEmail({ token: rawToken }),
});
```

Conteúdo do email

O componente TokenEmail foi criado com `@react-email/components`, apresentando:

- logótipo da aplicação;
- *token* visualmente destacado;
- link direto para a página de registo (`/signup`);
- instruções de utilização.

```
<Section style={codeBox}>  
  <Text style={confirmationCodeText}>{token}</Text>  
</Section>
```



Aqui está o seu token!

A empresa que registou foi validada por um administrador. Utilize o seguinte token para a registar na nossa dashboard.

z85mw2rhqe

Pode utilizar o token neste link: <https://dashboard.jethire.pt/signup/>

Se não tentou registar nenhuma empresa, pode ignorar este email ou contactar-nos para perceber o que aconteceu.

Jet Hire

©2025 Jet Hire. Todos os direitos reservados.

Figura 129 - Exemplo de mail de token de acesso

Registo de empresa com *token*

Após a empresa receber o seu **token de acesso** por email (após validação do pedido por parte de um administrador), é possível **finalizar o processo de registo** diretamente na aplicação.

Este registo consiste na criação das **credenciais do administrador principal da empresa**, que será o primeiro membro da equipa a aceder à conta da organização.

Interface de registo

A página de registo apresenta um **formulário simples e direto**, com os seguintes campos:

- **token de acesso;**
- **palavra-passe;**
- **confirmação da palavra-passe;**
- aceitação dos **termos e condições**.

Visualmente, a página foi construída com base no estilo da plataforma, dividida em duas colunas: uma com o **formulário funcional** e outra com o **logótipo da aplicação** e um **slogan institucional**, garantindo uma consistência visual e confiança ao utilizador.

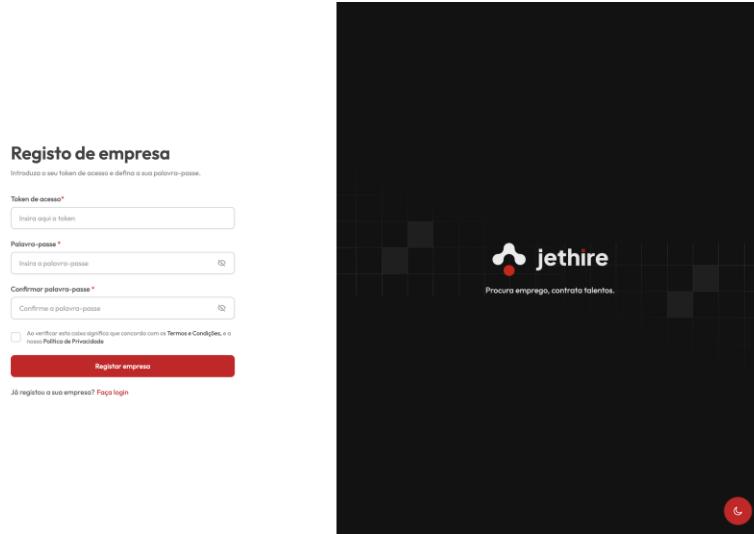


Figura 130 - Página de registo de empresas

Criação da conta e geração automática do nome de utilizador

Assim que o formulário é submetido, é feita uma chamada a uma **API interna** responsável por:

- **verificar a validade do token inserido;**
- **associar o novo utilizador à empresa previamente registada (dados preenchidos anteriormente);**

- **gerar um nome de utilizador automaticamente**, com base no nome da empresa e no nome do administrador.

Para este processo, foi implementado um sistema inteligente de geração de nomes de utilizador:

- primeiro, uma função remove os **acentos, espaços e caracteres especiais**, tornando o texto limpo e uniforme:

```
function removeAccents(str: string): string {  
    return str  
        .normalize("NFD")  
        .replace(/[\u0300-\u036f]/g, "")  
        .toLowerCase()  
        .replace(/\s+/g, "");  
}
```

- em seguida, é construído o nome de utilizador no seguinte formato: [empresa]-[nome]. Por exemplo, o nome de utilizador para a empresa **Jet Hire** com o administrador **Tomé Almeida** será: **jethire-tomealmeida**.

Se esse nome já estiver em uso dentro da empresa, o sistema verifica a existência de duplicados e gera automaticamente uma variação com numeração sequencial, como por exemplo: **jethire-tomealmeida1**.

Confirmação e envio de email com credenciais

Quando a conta é criada com sucesso, é enviado automaticamente um **email de confirmação** com os dados de acesso do novo administrador. Este email contém:

- **o nome de utilizador gerado;**
- informação de que a conta foi registada com sucesso;
- instruções para aceder à plataforma.

Este processo assegura uma **experiência simples, automática e segura** tanto para o utilizador como para a equipa de administração da aplicação.



Bem-vindo à Jet Hire, Jet Hire!

A sua empresa foi registada com sucesso na nossa plataforma. Pode agora aceder à sua conta com os seguintes dados:

Utilizador: jethire-tomealmeida
Palavra-passe: tome

Aceda à sua conta através do seguinte link:
<https://dashboard.jethire.pt/signin>

Se não tentou registrar nenhuma empresa, pode ignorar este email ou contactar-nos para perceber o que aconteceu.

Jet Hire

©2025 Jet Hire. Todos os direitos reservados.

Figura 131 - Exemplo de email de credenciais

A partir deste momento, o administrador da empresa pode **aceder** normalmente à aplicação, usando os dados de acesso enviados, e dar início à gestão da sua conta.

Convite de novos membros para a equipa

Depois de uma empresa concluir o processo de ativação através do *token*, o administrador principal ganha acesso total à conta da organização. Uma das funcionalidades mais importantes disponíveis de imediato é a **possibilidade de convidar novos membros para integrarem a equipa da empresa**.

Interface de gestão de membros

Para isso, foi criada uma página dedicada à **gestão da equipa**, onde o administrador pode:

- **convidar novos membros** através de um formulário;
- **definir a função de cada utilizador**, como por exemplo: recrutador, gestor ou administrador;
- visualizar uma **lista de todos os convites pendentes**.

A interface foi desenhada para ser clara, moderna e funcional. A lista de convites mostra-se em forma de tabela, com **filtros por nome, email e função**, permitindo uma navegação simples mesmo em equipas maiores.

	Nome		Email	Recrutador (recomendado)		+ Adicionar
	Nome		Email			
Emanuel Oliveira		emanuel.oliveira@glacier.pt		Recrutador	Expirado	
Tomé Almeida		ritomealmeida@gmail.com		Recrutador	Ativo	
Teste		emanuel.oliveira@glacier.pt		Recrutador	Expirado	

Figura 132 - Listagem de convites

Validação e expiração de convites

Cada convite enviado tem uma **validade de 1 hora**. Se não for aceite dentro desse prazo, o mesmo é automaticamente marcado como **expirado**, o que obriga a reemitir um novo convite.

Antes de enviar o convite, a aplicação executa uma **verificação automática** para garantir:

- que **não existe já um utilizador registado** com aquele email;
- que **não existe um convite pendente e ainda válido** para o mesmo endereço.

Isto evita duplicações e garante a consistência dos dados da equipa da empresa.

Envio do convite por email

Quando todas as validações são ultrapassadas, é enviado automaticamente um **email personalizado** ao utilizador convidado, com:

- **o nome da empresa;**
- **a função para a qual foi convidado;**
- **um link com o token de convite;**
- **a informação sobre a validade do convite.**

Este link permite ao convidado criar a sua conta associada à empresa de forma simples e segura, bastando inserir uma palavra-passe e confirmar os dados enviados.

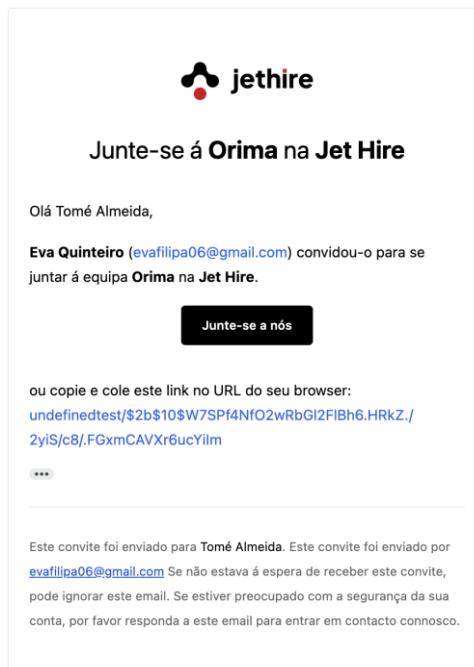


Figura 133 - Exemplo de email de convite

Criação de conta de convidado

Depois de o convite ser enviado por email, o utilizador convidado recebe um link com um **token de acesso exclusivo**. Este link leva-o diretamente para uma página de registo simplificada, específica para **aceitação de convites**.

Interface de convite

A página apresenta um formulário onde o utilizador apenas precisa de confirmar a palavra-passe.

Todo o processo foi planeado para ser **rápido, simples e seguro**, sem necessidade de preencher dados adicionais — visto que as informações principais (como nome, email e função) já tinham sido definidas previamente no convite.

A interface segue a mesma linha visual da aplicação, dividida em dois blocos: o formulário à esquerda e o *branding* da aplicação à direita.

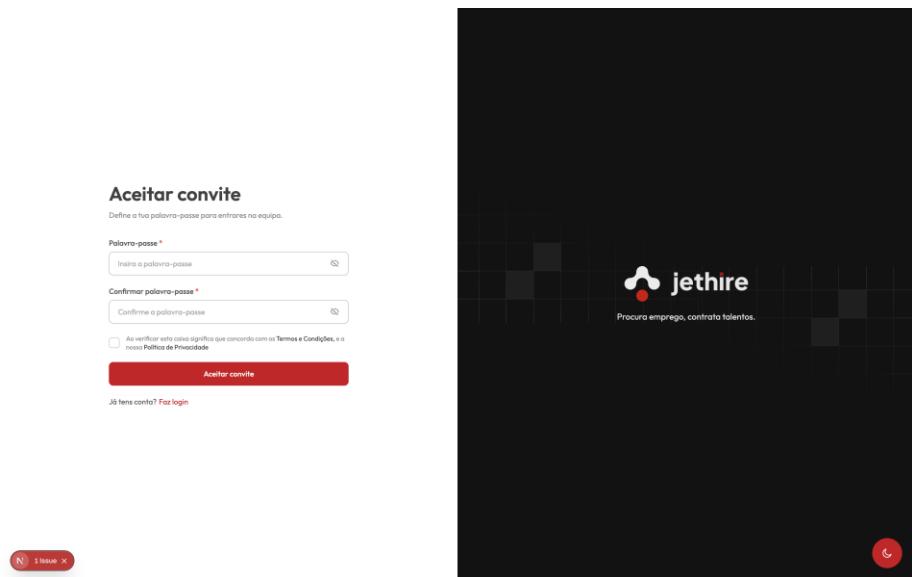


Figura 134 - Página de convite

Registo com base no token

Quando o utilizador submete o formulário:

- o sistema **valida o token** presente na URL (que estava incluído no link do email);
- se o *token* for válido e ainda estiver dentro do prazo de 1 hora, o registo é concluído com sucesso;
- o utilizador é adicionado à **equipa da empresa**, com a função atribuída no momento do convite;
- tal como no registo do administrador, é gerado **automaticamente um nome de utilizador único**, com base no nome e no da empresa;
- por fim, é enviado um **email de confirmação** com as credenciais finais e instruções de acesso.

Visualização da equipa

Depois de convidar os membros, o administrador precisa de conseguir **monitorizar quem faz parte da empresa** de forma clara e imediata.

Para isso, foi desenvolvida uma **página dedicada à listagem da equipa**, onde são apresentados todos os membros associados à empresa.

Interface de membros da equipa

A página apresenta um conjunto de **cartões individuais**, onde cada cartão representa um membro da equipa.

O *layout* é **limpo**, com destaque para a informação **essencial** de cada utilizador, permitindo ao administrador ter uma visão rápida da composição da equipa.

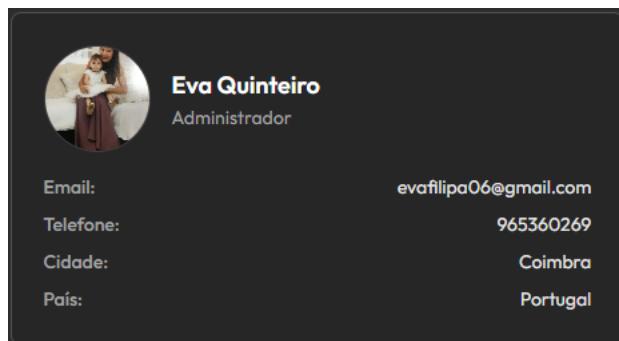


Figura 135 - Exemplo de membro de equipa

API de apresentação de membros da equipa

Por trás desta listagem existe uma lógica simples, mas eficaz:

- a aplicação identifica **quem está com sessão iniciada**, através do **ID da sessão ativa**;
- esse ID é usado para identificar a **empresa a que o utilizador pertence**;
- com base nisso, é feita a listagem dos **membros da empresa atual**, apresentando apenas os utilizadores da mesma organização.

Esta abordagem garante que os dados apresentados são **seguros e privados**, e que cada empresa só vê a sua própria equipa.

Edição de perfil de empresa

A funcionalidade de **edição de perfil da empresa** permite que cada organização mantenha o seu perfil sempre **atualizado, completo e com identidade visual própria**. Esta área é fundamental para garantir que a presença da empresa na plataforma é profissional, clara e atrativa para possíveis candidatos.

Estrutura geral da página

A interface está organizada em blocos temáticos, apresentados num formulário contínuo.

Cada componente foi modularizado para facilitar a manutenção e escalabilidade do código:

Tabela 2 - Estrutura geral da página de empresas

Logótipo	Upload da imagem representativa da empresa
Foto de fundo	Imagen de capa utilizada no perfil público
Informações básicas	Nome, <i>slogan</i> , país, cidade
Contactos	Morada completa, email institucional, número de telefone
Regime remoto	Indicação se a empresa permite ou não trabalho remoto
Tags	Áreas de atuação selecionáveis
Descrições	Blocos livres com título e texto para apresentar a empresa

Carregamento inicial dos dados

A primeira ação ao entrar na página é **detetar o utilizador autenticado** e carregar os dados da sua empresa, através do ID contido na sessão:

```
const res = await fetch('/api/company/profile/get/${session.user.id}', { method: "GET" });
```

A API (GET) procura a empresa onde o membro pertence, e retorna os dados do perfil:

```
const company = await Company.findOne({ "team._id": id }).select("-team - pending").lean();
```

O `select("-team -pending")` é fundamental aqui: **protege a privacidade** da equipa da empresa, evitando que os dados sensíveis sejam expostos neste *endpoint*.

Comparação de alterações

Antes de enviar alterações para o servidor, a aplicação avalia se foram feitas **modificações reais**. Esta comparação é feita manualmente com base nos valores iniciais:

```
if (info.name !== originalData.info.name) return true;  
  
if (!arraysEqual(tagsInfo.tags, originalData.tagsInfo.tags)) return true;  
  
if (banner !== null) return true;
```

Este cuidado evita chamadas desnecessárias à API e melhora a experiência do utilizador.

Gestão de imagens com o Cloudinary

Quando o logótipo ou o *banner* são alterados:

- a imagem anterior é removida da **Cloudinary**, através da função `deleteFromCloudinary(public_id);`
- a nova imagem é convertida em Buffer e enviada para a **Cloudinary**;
- a empresa é atualizada com os dados da nova imagem (incluindo `secure_url` e `public_id`).

```
if (logoFile && logoFile.size > 0) {  
  
    if (company.logo?.public_id) {  
  
        await deleteFromCloudinary(company.logo.public_id);  
  
    }  
  
    const buffer = Buffer.from(await logoFile.arrayBuffer());  
  
    const uploaded = await uploadToCloudinary(buffer);  
  
    company.logo = uploaded;  
  
}
```

Este sistema garante que **não há ficheiros órfãos na Cloudinary**, e que o armazenamento é usado de forma otimizada.

Organização das descrições

A secção de **descrições** foi pensada para oferecer flexibilidade às empresas. Cada bloco é composto por um título e um texto.

No código, estas descrições são armazenadas como um vetor:

```
const [descriptions, setDescriptions] = useState<{ title: string; text: string }[]>([]);
```

E enviadas à API como JSON:

```
formData.append("descriptions", JSON.stringify(descriptions));
```

No MongoDB, são guardadas num *subschema* de descrição com `_id: false`, o que significa que **não têm um identificador próprio**, mas fazem parte diretamente do objeto da empresa.

Atualização final do perfil

Após submeter as alterações, a API atualiza todos os campos da empresa:

```
company.name = name;  
company.slogan = slogan;  
company.city = city;  
company.country = country;  
company.remote = remote;  
company.contact = contacts;  
company.tags = tags;  
company.description = descriptions;
```

E guarda tudo com:

```
await company.save();
```

No final, o utilizador recebe *feedback* sobre a operação e pode verificar visualmente as alterações feitas através do **link direto para o perfil público**:

```
<Link href={"https://jethire.pt/company/${originalData?.id}">aqui</Link>
```

Segurança

- as alterações só são permitidas a utilizadores com **sessão iniciada**;
- o ID da sessão é usado para garantir que o utilizador **pertence** à empresa que está a **editar**;
- a comunicação com o **Cloudinary** é feita com autenticação segura e controlo de duplicações.

Edição de perfil de utilizador

A funcionalidade de **edição de perfil de utilizador** permite que **membros de empresas** e **administradores** editem os seus dados pessoais, garantindo que a informação associada a cada conta está sempre correta, atualizada e sincronizada com o sistema.

Lógica de carregamento de perfil

Quando a página é carregada, é feito um pedido à API com o **ID do utilizador autenticado**. Esse pedido é protegido — se o ID da sessão não corresponder ao ID no *endpoint*, o acesso é bloqueado:

```
if (!session || session.user.id !== id) {  
    return NextResponse.redirect(new URL("/unauthorized", request.url));  
}
```

Depois, a API tenta localizar o utilizador:

- Primeiro, procura-o como membro de uma empresa:

```
const company = await Company.findOne({ "team._id": id }, { "team.$": 1, name: 1 });
```

- Se não encontrar, tenta como administrador:

```
const admin = await Admin.findById(id).select("-password");
```

Dados apresentados na interface

A interface divide-se em três secções principais, cada uma representada por um componente modular:

Tabela 3 - Dados de perfil de utilizador

Componente	Campos editáveis
<i>UserMetaCard</i>	Foto de perfil (upload de imagem)
<i>UserInfoCard</i>	Nome, função, redes sociais, telefone
<i>UserAddressCard</i>	Cidade, país



Figura 136 - UserMetaCard

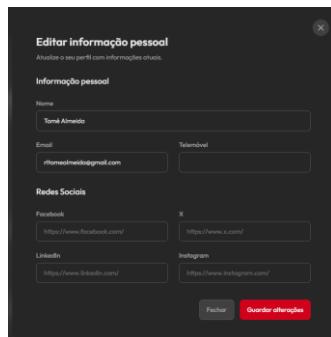


Figura 137 - UserInfoCard



Figura 138 - UserAddressCard

Estes componentes recebem o objeto **user** como **prop** e são responsáveis por exibir e permitir a edição dos respetivos campos.

Atualização dos dados

Cada grupo de campos é atualizado por chamadas a APIs específicas:

Morada

```
PUT /api/profile/edit/address/:id
{
  "city": "Lisboa",
  "country": "Portugal"
}
```

Contactos e redes sociais

```
PUT /api/profile/edit/contact/:id
{
  "phone": "912345678",
  "socials": [
    { "platform": "LinkedIn", "url": "https://linkedin.com/in/exemplo" }
  ]
}
```

Foto de perfil

O upload da imagem é tratado via FormData, e segue este fluxo:

- a imagem é convertida para Buffer;
- se já existir uma imagem antiga, é removida da **Cloudinary**;
- a nova imagem é enviada para a **Cloudinary**;
- o URL seguro (*secure_url*) e o ID público (*public_id*) são guardados no utilizador.

```
if (admin.logo.public_id) {
  await deleteFromCloudinary(admin.logo.public_id);
}

const uploadResult = await uploadToCloudinary(buffer);
admin.logo = {
  secure_url: uploadResult.secure_url,
  public_id: uploadResult.public_id
};
```

Segurança

Todas as rotas:

- verificam a validade do ID;
- garantem que apenas o **próprio utilizador pode editar os seus dados**;

- adaptam a lógica consoante o tipo de utilizador;
- evitam o carregamento de campos sensíveis como a palavra-passe.

Benefícios da modularização

Cada secção da edição foi dividida em componentes React (*UserMetaCard*, *UserInfoCard*, etc.), o que oferece várias vantagens:

- código mais **limpo e reutilizável**;
- facilidade de **manutenção** e extensão **futura** (ex: adicionar secções);
- separação clara entre **lógica e visualização**.

Visualização de empresas

A funcionalidade de **visualização de empresas** permite listar publicamente todas as empresas ativas na plataforma, exibindo os dados mais relevantes de forma clara, organizada e responsiva.

Isto oferece aos utilizadores uma visão geral das organizações registadas, incentivando a interação, candidatura ou simples consulta.

API de listagem

O *endpoint* de listagem (/api/company/list) é responsável por fornecer os dados de todas as empresas presentes na base de dados:

```
const company = await Company.find().select("-pending -team");
```

A query **exclui** os campos **sensíveis**:

- *pending*: convites e pedidos de adesão;
- *team*: lista dos membros da empresa.

Este *endpoint* devolve apenas os dados públicos relevantes (nome, *slogan*, localização, contactos, etc.), garantindo **privacidade e eficiência**.

Interface de apresentação

Cada empresa é representada por um **cartão visual** (CompanyCard), com os seguintes elementos:

Tabela 4 - Apresentação de empresas

Campo	Descrição
Logótipo	Imagen da empresa (ou ícone por defeito)
Nome	Nome completo da empresa
Slogan	Frase de apresentação
Área	Setor de atuação (ex: Desenvolvimento)
Tipo de trabalho	Regime (presencial, remoto, híbrido)
Cidade / País	Localização geográfica
Endereço	Morada física da empresa
Email	Email institucional
Telefone	Número de contacto
Website	Link externo da empresa

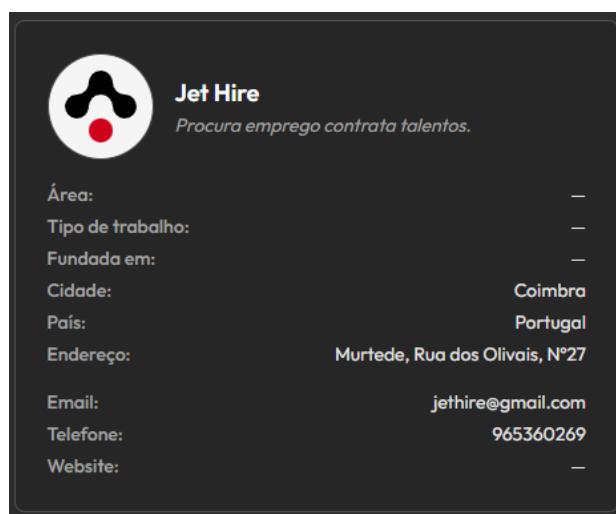


Figura 139 - Exemplo de empresa

Criação de oferta

A funcionalidade de **criação de ofertas** foi pensada para permitir que as empresas registadas na plataforma possam publicar oportunidades de forma rápida, organizada e com total controlo sobre os dados inseridos.

Esta funcionalidade inclui:

- um **formulário interativo e modular**, dividido por categorias de campos;
- um sistema de **validação automática** e integração com a sessão do utilizador;
- o envio dos dados para a base de dados MongoDB, através de uma **API protegida**.

Estrutura do formulário

A página *AddOffer* está dividida em vários componentes React reutilizáveis, o que facilita a manutenção e clareza do código:

- **FunctionAndLevel**: função e nível da oferta;
- **TypeAndExperience**: tipo de contrato e anos de experiência;
- **SalaryRange**: salário mínimo e máximo;
- **Tags**: tecnologias e regime remoto;
- **Descriptions**: blocos descritivos com título e texto;
- **MinorFriendly**: indica se a oferta está disponível para menores;

O estado do formulário é controlado com um único **useState**:

```
const [formData, setFormData] = useState({  
  role: "",  
  level: "",  
  type: "",  
  experience: "",  
  salaryMin: "",  
  salaryMax: "",  
  remote: "",  
  tags: []},
```

```
description: [],
isMinorFriendly: false,
});
```

Cada campo individual é atualizado através da função **handleChange**, enquanto as **tags** e descrições têm lógica própria para permitir múltiplos elementos de forma dinâmica.

Envio dos dados com validação de sessão

Quando o formulário é submetido, é feito um pedido POST para a API /api/offers/create:

```
const res = await fetch("/api/offers/create", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify(formData),
});
```

Esta API valida primeiro a **sessão do utilizador autenticado** com `getServerSession(authOptions)`, garantindo que só membros de empresas podem criar ofertas.

A seguir, identifica a empresa associada ao utilizador:

```
const company = await Company.findOne({ "team.email": session.user.email });
```

Se for encontrada, é criada uma nova oferta com os dados do formulário:

```
const newOffer = await Offer.create({
  ...rest,
  salary: {
    salaryMin: Number(salaryMin),
    salaryMax: Number(salaryMax),
  },
  company: company._id,
});
```

Os campos `salaryMin` e `salaryMax` são agrupados dentro de um único objeto `salary`, de acordo com o *schema* da base de dados.

Confirmação visual

No final do processo, é apresentado um **modal de confirmação** que indica ao utilizador que a oferta foi criada com sucesso:

```
<Modal isOpen={isOpen} onClose={onClose}>  
  <p>A oferta foi criada com sucesso!</p>  
</Modal>
```

Este *feedback* positivo melhora a experiência do utilizador e evita ações duplicadas.

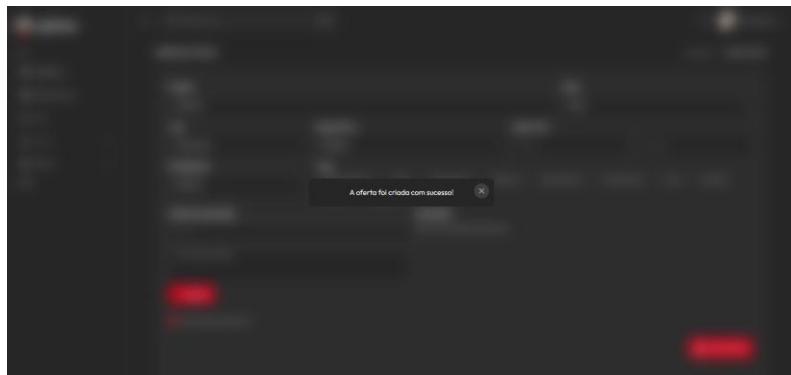


Figura 140 - Exemplo de confirmação visual

Visualização e gestão de ofertas

A funcionalidade de **visualização de ofertas** permite às **empresas** aceder a todas as oportunidades que já publicaram na plataforma. Além da listagem, é possível **pesquisar ofertas, ativar ou desativar, e consultar os candidatos** associados a cada uma delas.

Filtro e pesquisa inteligente

Ao entrar na página, a aplicação faz um pedido à API para **obter todas as ofertas** da empresa associada à sessão atual:

```
const res = await fetch("/api/offers/list");
```

Os dados são guardados num `useState`, e aplicam-se filtros em tempo real através de uma função `normalize`, que remove **acentos** e converte tudo para **minúsculas**, permitindo uma **pesquisa mais flexível**.

Ativação e remoção de ofertas

Cada oferta pode ser **ativada** ou **removida** temporariamente com um simples clique no ícone de **lixo**. Isso envia um pedido PATCH à API com o novo estado:

```
fetch('/api/offers/status/${id}', {
  method: "PATCH",
  body: JSON.stringify({ isActive: !currentStatus }),
});
```

Na interface, a oferta ativa é marcada com um **badge verde** ("Ativo") e a removida com um **badge vermelho** ("Removido").



Figura 141 - Exemplo de oferta ativa e oferta removida

Visualização de candidatos

Cada **oferta publicada** pela empresa inclui um **botão de acesso direto** à lista de candidatos que se candidataram a essa vaga. Ao clicar, o utilizador é redirecionado para uma página dedicada, onde pode visualizar, de forma clara e organizada, todos os **perfis associados** àquela oferta.

A listagem mostra os seguintes dados para cada candidato:

- **nome completo;**
- **cargo pretendido;**
- **localização** (cidade e país);
- **email;**
- **telefone;**
- **imagem de perfil;**
- **ligação direta ao perfil completo.**

A interface foi desenvolvida para ser **intuitiva, rápida e agradável**, facilitando a leitura dos dados mais relevantes. Os candidatos são apresentados em **cartões individuais**, e é possível fazer **pesquisa por nome, localização, contacto ou cargo**.

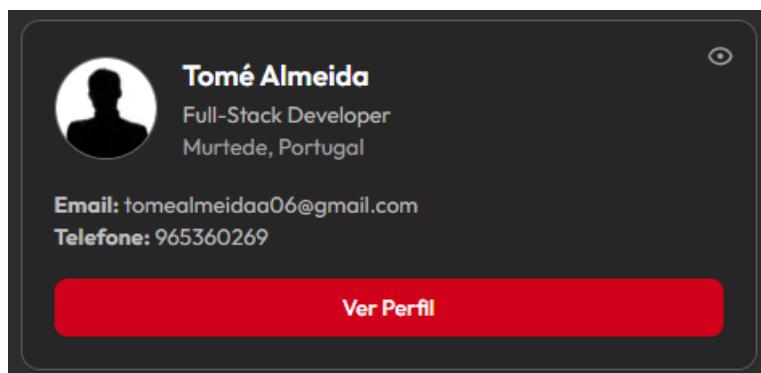


Figura 142 - Apresentação de candidatos

Marcação de candidatos como "visto"

Uma funcionalidade muito útil para recrutadores é a possibilidade de **marcar candidatos como "vistos"**, permitindo distinguir os candidatos que já foram analisados dos que ainda estão por rever.

Na interface, esta ação é feita através de um **ícone de olho** no canto superior do cartão do candidato. Internamente, ao clicar, o sistema envia os dados da oferta e do candidato e executa a seguinte lógica:

```
const offer = await Offer.findById(offerId);
const candidate = offer.candidates.find(
  (c) => c.user.toString() === userId
);
candidate.isSeen = !candidate.isSeen;
await offer.save();
```

Ou seja, o sistema:

- **vai buscar a oferta correspondente** na base de dados;
- **procura o candidato** específico dentro do **array** de candidaturas;
- **inverte o valor** do campo *isSeen* (de *true* para *false* ou vice-versa);
- **guarda a alteração** na base de dados MongoDB.

Atualização visual imediata

Depois de atualizado na base de dados, o estado do candidato é também **atualizado no front-end**, sem necessidade de recarregar a página. Isto é feito com um simples `setState`:

```
setCandidates((prev) =>
  prev.map((c) =>
    c._id === candidate._id ? { ...c, isSeen: data.isSeen } : c
  )
);
```

A interface mostra uma **sobreposição com a palavra "Visto"**, que desaparece ao passar o rato por cima do cartão — tornando o processo visualmente claro e funcional para o utilizador.

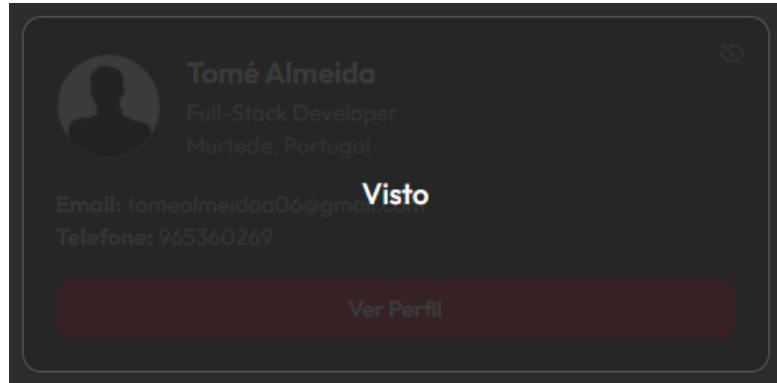


Figura 143 - Candidato "visto"

Registo de *logs*

Para garantir maior controlo sobre as ações realizadas na plataforma e permitir uma eventual **auditoria interna**, foi implementado um sistema de **logs automáticos**. Este sistema regista eventos importantes ao longo do tempo, como **alterações de perfil**, **criação de ofertas** ou **operações realizadas por administradores**.

Estrutura do *log*

Cada registo de *log* é armazenado na base de dados através de um **modelo Mongoose** dedicado, com os seguintes campos:

```
{
  message: String,      // Descrição do evento
  type: "admin" | "company" | "user" | "system",
  userId: ObjectId,    // Associado à entidade envolvida (opcional)
  level: "info" | "warn" | "error", // Nível de gravidade
  createdAt: Date       // Gerado automaticamente
}
```

O campo *type* permite identificar o **tipo de entidade que originou o evento**, e o *level* indica a **importância ou urgência** do mesmo. Todos os registos incluem também a **data e hora da criação**.

Criação de *logs* automáticos

Para facilitar a utilização, foi criada uma função genérica chamada *logEvent()* que pode ser usada em qualquer parte da aplicação:

```
export async function logEvent({ message, type, userId, level = "info" }) {  
    await Log.create({ message, type, userId, level });  
}
```

Esta função aceita quatro parâmetros:

- **message**: descrição do que aconteceu;
- **type**: tipo de entidade (*company*, *admin*, etc.);
- **userId**: opcional, identifica o autor do evento;
- **level**: grau de gravidade (por defeito, *info*).

Exemplo prático

Um exemplo concreto de utilização é o registo de quando uma empresa atualiza o seu perfil:

```
await logEvent({  
    message: `O perfil da empresa "${company.name}" foi atualizado.`,
    type: "company",
    userId: id,
    level: "info",
});
```

Este evento fica **guardado na base de dados**, permitindo um histórico claro de ações por parte de cada empresa.

Privacidade e segurança

O sistema de *logs* foi pensado para **respeitar a privacidade dos dados**. Não são armazenadas informações sensíveis como senhas, emails completos ou conteúdos de mensagens — apenas **descrições resumidas**, *userId* e *type*, suficientes para **identificar o evento** sem comprometer dados pessoais.

Página inicial da dashboard

A página inicial da *dashboard* apresenta um **resumo visual e estatístico** da atividade na plataforma. O conteúdo varia consoante o tipo de utilizador (empresa ou administrador), mas a estrutura principal é semelhante em ambos os casos.

Estatísticas

Na parte superior da página são apresentados **três blocos informativos**, também chamados de **cards**, com os seguintes dados:

- **total de ofertas publicadas;**
- **total de candidatos registados;**
- **total de empresas ativas.**

Estes valores são obtidos dinamicamente através de contagens da base de dados, por exemplo:

```
const totalOffers = await Offer.countDocuments();
const totalUsers = await User.countDocuments();
const totalCompanies = await Company.countDocuments({ isActive: true });
```

No caso de empresas, os dados são filtrados para apresentar apenas as **ofertas da própria empresa** e os **candidatos que se candidataram às suas vagas**.

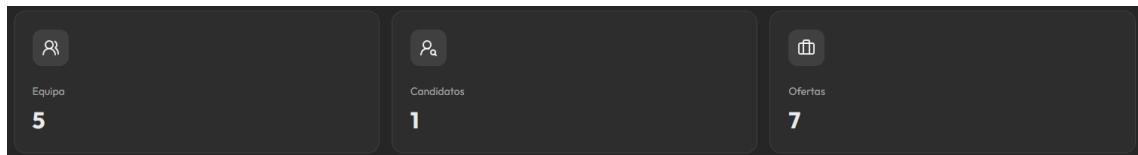


Figura 144 - Estatísticas da dashboard

Gráficos de barras

A seguir aos *cards*, são apresentados **dois gráficos de barras verticais**, que variam consoante o tipo de utilizador:

Para administradores:

- **distribuição de ofertas por modalidade** (remoto, híbrido, presencial);
- **distribuição de contratos** (tempo inteiro, part-time, estágio).

Para empresas:

- **número de candidatos por nível de experiência;**
- **número de ofertas por tipo de contrato.**

Os dados são agrupados diretamente na base de dados:

```
const stats = await Offer.aggregate([
  { $group: { _id: "$type", count: { $sum: 1 } } }
]);
```

Cada grupo é transformado num conjunto de **barras coloridas**, com títulos e legendas interativas. Estes gráficos facilitam a **comparação direta entre categorias**.



Figura 145 - Gráfico de barras da dashboard

Gráfico de linhas

No caso de empresas, o painel inclui também um **gráfico de linhas duplo**, que mostra a evolução **mensal** dos seguintes elementos:

- **candidatos recebidos;**
- **ofertas criadas;**

Este gráfico é construído com base nas datas de criação (*createdAt*) de cada documento. A lógica agrupa os registo por mês e conta quantos existem:

```
[  
  { month: "jan", candidates: 12, offers: 4 },  
  { month: "feb", candidates: 7, offers: 6 },  
  ...  
]
```

As duas linhas têm cores distintas e o gráfico permite **acompanhar o crescimento da atividade da empresa ao longo do tempo.**

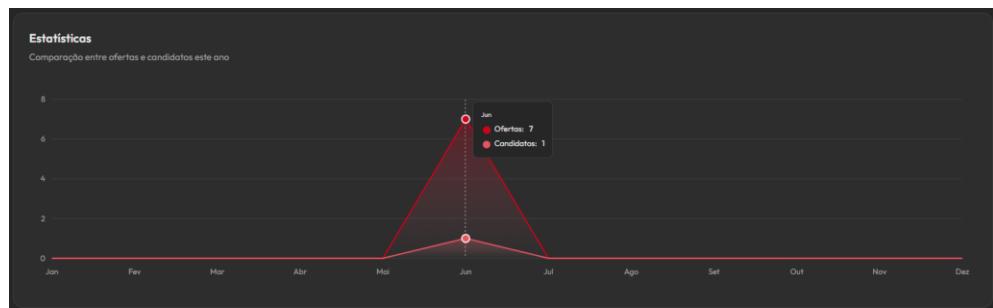


Figura 146 - Gráfico de linhas

Tabela de *logs*

Em vez do gráfico de linhas, os administradores veem uma **tabela interativa com os últimos *logs* registrados no sistema.**

Cada linha da tabela mostra:

- A **mensagem do evento**
- O **tipo** (*admin*, *company*, etc.)
- O **nível** (*info*, *warn*, *error*)
- A **data e hora do registo**

Os dados vêm diretamente da base de dados de *logs*:

```
const logs = await Log.find().sort({ createdAt: -1 }).limit(10);
```

Esta tabela permite aos administradores **monitorizar ações importantes em tempo real**, como criação de ofertas, edições de perfis, ou validações feitas.

Registo de atividades do sistema		
Estado	Mensagem	Data
Info	A empresa "Jet Hire" criou uma nova oferta: Teste nova	25/06/2025, 02:51:47
Info	Novo membro João Bita (101175pinkitjooos@gmail.com) adicionado à empresa "Jet Hire".	24/06/2025, 15:57:09
Info	Convite enviado para 101175pinkitjooos@gmail.com na empresa "Jet Hire".	24/06/2025, 15:56:23
Info	Novo membro Rúben Mendes (rubencm77@gmail.com) adicionado à empresa "Jet Hire".	24/06/2025, 10:43:20
Info	Convite enviado para rubencm77@gmail.com na empresa "Jet Hire".	24/06/2025, 10:42:41
Info	Novo membro João Teixeira (joaoandrielevateixeira@gmail.com) adicionado à empresa "Jet Hire".	24/06/2025, 03:06:15
Info	Convite enviado para joaoandrielevateixeira@gmail.com na empresa "Jet Hire".	24/06/2025, 03:04:26
Info	Perfil da empresa "Jet Hire" foi atualizado.	23/06/2025, 17:47:10

Figura 147 - Tabela de logs

Live chat com Firebase

Para permitir **comunicação em tempo real** entre os membros de cada empresa, foi implementado um sistema de **chat interno**, baseado no **Firebase Realtime Database**. Esta foi a única parte da aplicação onde o Firebase foi utilizado, por ser uma solução leve, escalável e com suporte automático à **sincronização em tempo real**.

Porquê Firebase?

Inicialmente, foi considerada a utilização de soluções como Socket.IO, mas o Firebase destacou-se por:

- eliminar a necessidade de servidores próprios para WebSocket;
- permitir **escuta de dados em tempo real** com baixo esforço;
- ser facilmente integrado com aplicações React/Next.js.

No contexto deste projeto, o Firebase foi utilizado **apenas para o Live Chat**, recorrendo à funcionalidade **Realtime Database**.

Organização do Realtime Database

Cada empresa tem uma **sala de chat exclusiva**, identificada pelo seu *companyId*. A estrutura é:

```
companies/
[companyId]/
  messages/
```

```
[messageId]: {  
    senderId,  
    name,  
    avatar,  
    text,  
    time  
}
```

Desta forma, **cada empresa tem um histórico privado** de mensagens, e os membros só conseguem aceder à sua própria sala.

Leitura de mensagens em tempo real

Sempre que o utilizador entra no *chat*, é ativado um *listener* que fica a escutar novas mensagens dentro da sala da empresa:

```
const messagesRef = ref(db, `companies/${companyId}/messages`);
```

```
const listener = onChildAdded(messagesRef, (snapshot) => {  
    const message = snapshot.val();  
    callback(message);  
});
```

Este método (*onChildAdded*) garante que **todas as novas mensagens são recebidas automaticamente**, sem necessidade de atualizações manuais ou recarregamentos.

Ao sair da página, o *listener* é removido com *off(...)*, para evitar consumo desnecessário de recursos.

Envio de mensagens

Ao enviar uma mensagem, os dados são enviados para o Firebase através da função *push*:

```
const messagesRef = ref(db, `companies/${companyId}/messages`);
```

```
await push(messagesRef, {  
    senderId: session.user.id,  
    name: session.user.name,  
    avatar: session.user.image,  
    text: messageText,  
    time: new Date().toLocaleTimeString([], { hour: "2-digit", minute: "2-digit" })  
});
```

Cada mensagem inclui:

- **id do remetente;**
- **nome;**
- **imagem de perfil;**
- **texto;**
- **hora formatada.**

Integração com a sessão

O *companyId* da sala de chat é obtido com base na **sessão do utilizador autenticado**.

Para isso, é feita uma chamada a um *endpoint* interno que devolve a empresa a que o utilizador pertence.

```
const res = await fetch("/api/chat/session");  
const data = await res.json();  
setCompanyId(data.companyId);
```

Isto garante que:

- o utilizador **entra automaticamente na sala correta;**
- não é necessário inserir IDs manualmente;
- a experiência é **simples e segura.**

Interface visual

A interface foi desenhada com foco em **clareza e usabilidade**, seguindo o estilo da Jet Hire:

- as mensagens do próprio utilizador são **alinhadas à direita** com fundo vermelho;
- as mensagens dos colegas aparecem **à esquerda**, com imagem e nome;
- a hora de envio é apresentada em baixo, com formatação local;
- existe um **campo de input com botão de envio**, que só é ativado quando a sessão e o *companyId* estão carregados.

As mensagens são apresentadas em blocos individuais, com classes diferentes consoante o lado:

```
{msg.sender === "me" ? "justify-end" : "justify-start"}
```

E com estilos distintos para cores, avatar e posicionamento.

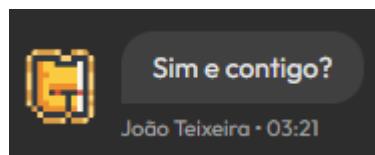


Figura 148 - Mensagem de outro utilizador

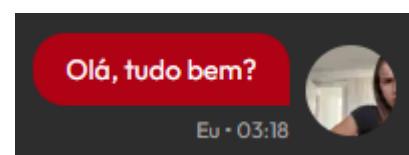


Figura 149 - Mensagem própria

Testes e validação da aplicação

Durante o desenvolvimento da Jet Hire, procurei garantir não só o bom funcionamento técnico de cada funcionalidade, mas também a qualidade da experiência para o utilizador final. Para isso, realizei testes manuais em várias fases e pedi a diferentes pessoas que experimentassem a aplicação e dessem *feedback* sincero.

Testes técnicos

Todos os principais componentes da aplicação foram testados manualmente ao longo do desenvolvimento. Isto incluiu funcionalidades como:

- criação e edição de perfis;
- submissão de candidaturas;
- registo e autenticação com NextAuth;
- envio e remoção de imagens no Cloudinary;
- comunicação em tempo real com Firebase;
- atualização de dados através da API com ligação MongoDB.

Verifiquei o comportamento da aplicação em diferentes resoluções de ecrã (telemóvel, tablet e *desktop*), testei em navegadores como Chrome e Firefox, e acompanhei a *performance* com ferramentas como a consola do *browser* e o Postman.

Validação com utilizadores

Para além dos testes técnicos, pedi a familiares, amigos e ao meu colega de estágio que utilizassem a plataforma como se fossem utilizadores reais. Este tipo de validação permitiu-me identificar pequenos erros e melhorar aspetos importantes da interface.

Com base no *feedback* que recebi, fiz várias melhorias, como:

- alteração das mensagens de erro e sucesso para serem mais claras;
- melhoria na legibilidade de certos botões e cores em modo escuro;
- ajustes nas animações e carregamento de componentes dinâmicos (ex.: candidatos e empresas);

- reforço da indicação de estados (por exemplo: *loading*, sem resultados, confirmação visual).

Este processo de validação foi fundamental para tornar a Jet Hire mais intuitiva, rápida e agradável para qualquer utilizador.

Conclusão

O desenvolvimento da Jet Hire foi uma experiência extremamente enriquecedora a vários níveis. Mais do que um simples projeto técnico, este trabalho representou um verdadeiro desafio pessoal, onde pude aplicar, consolidar e expandir os conhecimentos adquiridos ao longo do curso, desde a conceção da ideia até à implementação completa de duas aplicações distintas.

Ao longo do projeto, enfrentei diversos obstáculos técnicos e pessoais. Houve momentos de frustração, decisões difíceis e muito tempo investido em resolver erros que, à primeira vista, pareciam impossíveis de ultrapassar. Mas foi precisamente nesses momentos que mais cresci — aprendi a ser resiliente, a procurar soluções com espírito crítico e a confiar mais nas minhas capacidades. Sinto que evoluí não só como programador, mas também como pessoa.

A Jet Hire é hoje uma aplicação funcional, completa e realista, que pode ser utilizada por utilizadores e empresas reais. O projeto está online, disponível com domínio próprio, e integra uma vasta gama de funcionalidades modernas: sistema de autenticação seguro, base de dados estruturada com MongoDB, edição de perfil com upload de ficheiros para a *cloud*, sistema de favoritos, filtros dinâmicos, visualização de dados em tempo real, *dashboards* interativos, e até um chat interno para empresas. Cada funcionalidade foi pensada com um propósito concreto e desenvolvida com atenção ao detalhe, sempre com a experiência do utilizador como prioridade.

Este projeto permitiu-me trabalhar com tecnologias atuais como **Next.js**, **TypeScript**, **MongoDB**, **Firebase**, **TailwindCSS**, entre outras. Mais importante do que a parte técnica, ajudou-me a perceber o que é realmente esperado num ambiente profissional de desenvolvimento — planeamento, organização, modularização, documentação, testes e clareza na tomada de decisões.

Acima de tudo, a Jet Hire representa o culminar de três anos de formação, de dedicação, de erros e aprendizagens. É um reflexo direto do tipo de profissional que quero ser: **curioso, empenhado, prático e orientado para soluções reais**. Olho para este projeto com orgulho, não só pelo que consegui concretizar, mas porque sei que deixei nesta aplicação o melhor de mim em cada linha de código.

Termino este ciclo com a certeza de que estou preparado para o próximo. A Jet Hire foi o meu maior desafio até agora, mas acredito que é apenas o início de um percurso que quero continuar a construir com paixão, responsabilidade e ambição.

Webgrafia

ZacsTech. 2023. *How to install Github Desktop on Ubuntu 22.04.* Acedido em outubro de 2024: https://www.youtube.com/watch?v=ki0MuWceGk4&ab_channel=ZacsTech

UXTools. 2023. *How I make UI color palletes.* Acedido em novembro de 2024: https://www.youtube.com/watch?v=yYwEnLYT55c&ab_channel=UXTools

Heroicons. 2024. *Beautiful hand-crafted SVG icons, by the makers of Tailwind CSS.* Acedido em novembro de 2024: <https://heroicons.com/>

GTCoding. 2023. *Step-by-Step Guide: Adding Google Authentication with NextAuth in Next.js 13 and MongoDB.* Acedido em novembro de 2024: <https://www.youtube.com/watch?v=6N3Rumo-c3s>

Clerk. 2024. *Password-based authentication in Next.js.* Acedido em dezembro de 2024: <https://clerk.com/blog/password-based-authentication-nextjs>

HeyNode. 2020. *Salt and hash passwords with bcrypt.* Acedido em dezembro de 2024: <https://heynode.com/blog/2020-04/salt-and-hash-passwords-bcrypt/>

Rushabh Dabhade. 2023. *Next.js Demystified: User Authentication with NextJS & MongoDB.* Acedido em dezembro 2024: https://medium.com/@Rushabh_/nextjs-demystified-user-authentication-with-nextjs-mongodb-2a0e1e697526
<https://themeselection.com+14medium.com+14mongodb.com+14>

Vincent Delitz. 2024. *How to implement Password-Based Authentication in Next.js.* Acedido em dezembro 2024: <https://www.corbado.com/blog/nextjs-password-corbado.com>

Meet Patel. 2024. *How to Link Users Across Different Authentication Providers in Next.js?* Acedido em dezembro 2024:
mongodb.com+15mongodb.com+15corbado.com+15

DustinA. 2024. *Tailwind not applied in Next.js project.* Acedido em dezembro 2024:
udemy.com+12stackoverflow.com+12reddit.com+12

Mongoose. 2025. *Guide.* Acedido em janeiro de 2025:
<https://mongoosejs.com/docs/guide.html>

Mongoose. 2025. *Subdocuments.* Acedido em janeiro de 2025:
<https://mongoosejs.com/docs/subdocs.html>

React. 2025. *useState.* Acedido em janeiro de 2025:
<https://react.dev/reference/react/useState>

Firebase. 2025. *Integrate Firebase with a Next.js app.* Acedido em janeiro de 2025:
<https://firebase.google.com/codelabs/firebase-nextjs>

DeadSimpleChat. 2025. *Firebase Chat App Tutorial.* Acedido em janeiro de 2025:
<https://www.deadsimplechat.com/blog/firebase-chat-app-tutorial/>

Mindbowser. 2024. *Creating a Chat Application Using Firebase Realtime Database.* Acedido em janeiro de 2025: <https://www.mindbowser.com/firebase-realtime-database-chat-app/>

Dev.to. 2024. *Create a dashboard using Next.js, Tailwind CSS, and Chart.js.* Acedido em fevereiro de 2025: <https://dev.to/niranthal/build-a-dashboard-using-nextjs-tailwind-css-and-chartjs-3i3j>

Next.js Blog. 2025. *Next.js 15 App Router Fundamentals.* Acedido em março de 2025:

<https://nextjs.org/blog/next-15>

FreeCodeCamp. 2025. *Create a fullstack app with Next.js 15 and MongoDB.* Acedido em março de 2025: <https://www.freecodecamp.org/news/nextjs-mongodb-fullstack-app/>

Pagepro. 2024. *Next.js Performance Optimization in 9 Steps.* Acedido em abril de 2025:

<https://pagepro.co/blog/next-js-performance-optimization-in-9-steps/>

Blazity. 2025. *The Expert Guide to Next.js Performance Optimization.* Acedido em abril

de 2025: <https://blazity.com/blog/the-expert-guide-to-nextjs-performance-optimization>

Notion. 2025. *Organizing Fullstack Projects with Next.js.* Acedido em abril de 2025:

<https://notion.so/fullstack-nextjs-best-practices>

MongoDB Docs. 2025. *Schema Design Best Practices.* Acedido em abril de 2025:

<https://www.mongodb.com/docs/manual/core/data-modeling-introduction/>

CodeWithChris. 2025. *Handle API Errors Gracefully in Fullstack Apps.* Acedido em

maio de 2025: <https://codewithchris.com/error-handling-best-practices/>

Netlify Blog. 2025. *Deploying Multi-Page Apps with Subdomains.* Acedido em maio de

2025: <https://www.netlify.com/blog/2020/05/21/deploying-subdomains-with-netlify/>

StackOverflow. 2025. *Best Way to Implement Search Filters in React.* Acedido em

junho de 2025: <https://stackoverflow.com/questions/65192639/how-to-implement-multiple-search-filters-react>