



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 Cantanheda

Técnico de Gestão e Programação de Sistemas Informáticos

Prova de Aptidão Profissional: Jet Hire

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

Cantanheda

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.

Índice Geral

Introdução (Português)	1
Introduction (Inglês).....	3
Planeamento	5
Fundamentação	5
Recursos	6
Hardware	6
Software.....	6
Metodologia	8
Atividades.....	9
Fase de Definição	9
Fase de Desenvolvimento	9
Fase de Manutenção.....	10
Cronograma.....	10
Instalações.....	11
Visual Studio Code	11
MongoDB Compass.....	12
Node.js (node / npm / npx).....	13
Github Desktop	14
Inkscape	16
Krita.....	17
Postman	18
Sincronização entre dois computadores	20
Criação do repositório.....	20
Push.....	21
Pull	23
Sitemap	24
Projeto.....	26
Instalação do template.....	26
Título	Erro! Marcador não definido.
Logótipo	28

Escolha da paleta de cores.....	28
Cor principal	28
Cores secundárias	30
Cores neutras	31
Criação da paleta no projeto.....	31
Edição do template	32
Rodapé	32
Barra de navegação.....	33
Página principal.....	33
Página de ofertas	42
Página de empresas	44
Página de início de sessão.....	45
Página de registo.....	46
Página de apresentação de perfil.....	47
Página de edição de perfil.....	51
Segurança	61
Criação de um ambiente seguro	61
MongoDB.....	62
Criação do cluster.....	62
Acesso á base de dados.....	64
Mongoose.....	68
Instalação do Mongoose	68
Ligação á base de dados com Mongoose.....	68
Schemas	69
Subschemas.....	70
Criação de um modelo Mongoose	71
Envio de dados para a base de dados	73
Desenvolvimento funcional da aplicação	74
Gestão de formulários: useState, handleChange e handleSubmit	74
Registo de utilizadores	76
Início de sessão	77
React Hook – useSession()	78
Sessão partilhada.....	78
signIn & signOut.....	79
Google Cloud	80

Credenciais	85
Criação de um token JWT	86
Barra de navegação funcional.....	88
Perfil dinâmico	90
Estrutura da página.....	91
Recolha de dados com getServerSideProps	91
Renderização condicional e layout responsivo.....	93
Percentagem de Preenchimento	94
Segurança e personalização.....	95
Edição de perfil	97
Atualização dos dados em tempo real.....	97
Modularização do formulário	97
Preparação do Cloudinary	98
Upload Presets do Cloudinary	100
Upload de ficheiros com Cloudinary.....	100
Portfólio com gestão de imagens	101
Segurança e permissões	102
Página de candidatos	102
Estrutura geral da página.....	102
Carregamento dos dados com feedback visual	103
API de candidatos	104
Pesquisa em tempo real	105
Ordenação por data de registo	105
Renderização condicional com mensagens de fallback	107
Página de empresas	109
Registo de empresas.....	109
Estrutura geral da página de candidatos	110
Criação da dashboard	111
Middleware	111
Criação de modelos de empresa e de administrador	113
Início de sessão	118
Interface de Login	118
Configuração da Autenticação (Next-Auth).....	119
Registo de empresa com token.....	131
Interface de registo.....	131
Criação da conta e geração automática do nome de utilizador	132
Confirmação e envio de email com credenciais	132
Convite de novos membros para a equipa	133

Interface de Gestão de Membros	133
Validação e Expiração de Convites	134
Envio do Convite por Email.....	134
Criação de conta de convidado	135
Interface de convite	135
Registo com base no token.....	136
Visualização da equipa	137
Interface de membros da equipa	137
API de apresentação de membros da equipa	138
Edição de perfil de empresa.....	138
Estrutura geral da página.....	138
Carregamento inicial dos dados	139
Comparação de alterações	139
Gestão de imagens com o Cloudinary	140
Organização das descrições	140
Atualização final do perfil	141
Segurança	141
Edição de perfil de utilizador	142
Lógica de carregamento de perfil	142
Dados apresentados na interface	142
Atualização dos dados	143
Segurança	145
Benefícios da Modularização	145
Visualização de empresas	145
API de Listagem	145
Interface de apresentação.....	146
Carregamento de dados	147
Conclusão	149
Webgrafia.....	150

Índice de Figuras

Figura 1 - Metodologia Kanban	8
Figura 2 - Cronograma	10
Figura 3 - Instalação Visual Studio Code: Passo 1	11
Figura 4 - Instalação Visual Studio Code: Passo 2	11
Figura 5 - Extensões	12
Figura 6 - Instalação MongoDB Compass: Passo 1	12
Figura 7 - Instalação MongoDB Compass: Passo 2	13
Figura 8 - sudo apt install nodejs	13
Figura 9 - sudo apt install npm	13
Figura 10 - Verificação do npm	14
Figura 11 - Instalação GitHub Desktop: Passo 1	15
Figura 12 - Instalação GitHub Desktop: Passo 2	15
Figura 13 - Instalação GitHub Desktop: Passo 3	16
Figura 14 - Instalação do Inkscape: Passo 1	16
Figura 15 - Instalação do Inkscape: Passo 2	17
Figura 16 - Instalação do Krita: Passo 1	18
Figura 17 - Instalação do Krita: Passo 2	18
Figura 18 - Instalação do Postman: Parte 1	19
Figura 19 - Instalação do Postman: Parte 2	19
Figura 20 - Instalação do Postman: Parte 3	20
Figura 21 - Sincronização entre dois computadores: Passo 1	21
Figura 22 - Sincronização entre dois computadores: Passo 2	21
Figura 23 - Sincronização entre dois computadores: Passo 3	22
Figura 24 - Sincronização entre dois computadores: Passo 4	22
Figura 25 - Sincronização entre dois computadores: Passo 5	23
Figura 26 - Sincronização entre dois computadores: Passo 6	23
Figura 27 - Sincronização entre dois computadores: Passo 7	24
Figura 28 - Sincronização entre dois computadores: Passo 8	24
Figura 29 – Sitemap principal	25
Figura 30 - Download do template	26
Figura 31 - Instalação dos node_modules	26
Figura 32 - npm run dev	27
Figura 33 - Projeto executado	27
Figura 34 - Logótipo	28
Figura 35 - Escolha da cor principal	28

Figura 36 - Escolha do tom claro.....	29
Figura 37 - Escolha do tom escuro.....	29
Figura 38 - Criação de um gradiente claro	29
Figura 39 - Criação de um gradiente escuro	30
Figura 40 - Vermelho.....	30
Figura 41 - Amarelo.....	31
Figura 42 - Azul.....	31
Figura 43 - Verde	31
Figura 44 - Cores neutras.....	31
Figura 45 - Exemplo de variáveis globais	32
Figura 46 - Rodapé inicial.....	32
Figura 47 - Rodapé final	33
Figura 48 - Barra de navegação inicial	33
Figura 49 - Barra de navegação final	33
Figura 50 - Destaque principal inicial.....	34
Figura 51 - Destaque principal final	34
Figura 52 - Slider de categorias inicial.....	37
Figura 53 - Slider de categorias final	37
Figura 54 - Como funciona inicial	37
Figura 55 - Como funciona final.....	38
Figura 56 - Ofertas mais recentes inicial	38
Figura 57 - Ofertas mais recentes final.....	39
Figura 58 - Procurar trabalho inicial.....	39
Figura 59 - Procurar trabalho final	40
Figura 60 - Candidatos recentes inicial.....	40
Figura 61 - Candidatos recentes final	41
Figura 62 - Cria o teu perfil inicial	41
Figura 63 - Cria o teu perfil final	41
Figura 64 - Cabeçalho da página de ofertas inicial	42
Figura 65 - Cabeçalho da página de ofertas final	42
Figura 66 - Filtro avançado: Área	43
Figura 67 - Filtro avançado: Salário.....	43
Figura 68 - Filtro avançado: Modalidade	43
Figura 69 - Filtro avançado: Experiência	43
Figura 70 - Filtro avançado: Oferta Publicada	43
Figura 71 - Filtro avançado: Tipo de Trabalho	43

Figura 72 - Oferta de emprego inicial	44
Figura 73 - Oferta de emprego final.....	44
Figura 74 - Apresentação de empresas inicial.....	44
Figura 75 - Apresentação de empresas final	45
Figura 76 - Página de início de sessão inicial.....	45
Figura 77 - Página de início de sessão final	46
Figura 78 - Página de registo inicial	46
Figura 79 - Página de registo final.....	47
Figura 80 - Banner inicial.....	47
Figura 81 - Banner final	48
Figura 82 - Descrição final.....	48
Figura 83 - Competências iniciais.....	48
Figura 84 - Competências finais	49
Figura 85 - Experiência inicial	49
Figura 86 - Experiência final.....	49
Figura 87 - Educação inicial	50
Figura 88 - Educação final.....	50
Figura 89 - Portfólio normal	50
Figura 90 - Portfólio com hover	50
Figura 91 - Barra lateral inicial.....	51
Figura 92 - Barra lateral final	51
Figura 93 - Primeira secção de edição de perfil.....	52
Figura 94 - Segunda secção de edição de perfil.....	52
Figura 95 - Terceira secção da edição de perfil.....	53
Figura 96 - Quarta secção do registo de perfil.....	55
Figura 97 - Exemplo de adição de dados no registo de perfil	60
Figura 98 - Portfólio na edição de perfil	61
Figura 99 - Exemplo de portfólio na edição de perfil.....	61
Figura 100 - Quinta secção do registo de perfil	61
Figura 101 - Página inicial MongoDB	63
Figura 102 - Criação de projeto no MongoDB	63
Figura 103 - Criação de um cluster no MongoDB	64
Figura 104 - Criação de administrador no MongoDB.....	65
Figura 105 - Escolha de aplicação no MongoDB	65
Figura 106 - Ligação ao cluster no MongoDB.....	66
Figura 107 - Página inicial MongoDB Compass	67

Figura 108 - Interface do MongoDB Compass.....	67
Figura 109 - Dependências Mongoose.....	68
Figura 110 - Exemplo de erro no registo	77
Figura 111 - Criação de projeto na GCC	80
Figura 112 - Dashboard da GCC.....	81
Figura 113 - Criação do token OAuth: Parte 1.....	81
Figura 114 - Criação do token OAuth: Parte 2.....	82
Figura 115 - Criação do token OAuth: Parte 3.....	82
Figura 116 - Criação do token OAuth: Parte 4.....	83
Figura 117 - Criação do token OAuth: Parte 5.....	83
Figura 118 - Criação do token OAuth: Parte 6.....	84
Figura 119 - Barra de navegação sem sessão	90
Figura 120 - Barra de navegação com sessão	90
Figura 121 - Verificação de perfil completa	95
Figura 122 - Verificação de perfil por concluir.....	95
Figura 123 - Exemplo de perfil por preencher.....	95
Figura 124 - Exemplo de perfil completo	96
Figura 125 - Criação de conta no Cloudinary	98
Figura 126 - Página de "API Keys"	99
Figura 127 - Exemplo de "A carregar candidatos"	104
Figura 128 - Exemplo de contador de apresentação de utilizadores	105
Figura 129 - Exemplo de "dropdown" na aplicação	106
Figura 130 - Candidatos ordenados por "Mais Recente"	106
Figura 131 - Candidatos ordenados por "Mais Antigo"	107
Figura 132 - Candidatos a carregar.....	107
Figura 133 - Sem resultados	108
Figura 134 - Resultado da pesquisa de candidatos	108
Figura 135 - Placeholder do formulário do registo de empresas.....	109
Figura 136 - Formulário de registo de empresas	109
Figura 137 - Cabeçalho da página de empresas	110
Figura 138 - Secção 1 da barra lateral da página de empresas.....	110
Figura 139 - Secção 2 da barra lateral da página de empresas.....	110
Figura 140 - Apresentação das empresas	111
Figura 141 - Página de início de sessão da dashboard inicial	119
Figura 142 - Página de início de sessão da dashboard final.....	119
Figura 143 - Exemplo de cartão de registo de empresa	123

Figura 144 - Domínio verificado no Resend	124
Figura 145 - API key do Resend.....	124
Figura 146 - Exemplo de email de token de acesso	125
Figura 147 - Interface de registo de empresa.....	131
Figura 148 - Exemplo de email de registo de empresa.....	133

Índice de Tabelas

Introdução (Português)

Para a minha Prova de Aptidão Profissional (PAP), escolhi desenvolver a “Jet Hire”, uma plataforma online para oferta e procura de emprego. A escolha deste projeto foi inspirada numa experiência pessoal que vivi durante as férias de verão. Na altura, precisei de procurar um trabalho para juntar dinheiro, mas encontrei várias dificuldades em encontrar vagas de forma prática e acessível. Isso fez-me perceber a importância de uma ferramenta que torne este processo mais simples tanto para quem procura como para quem oferece emprego.

Durante o desenvolvimento do projeto, optei por direcionar a plataforma apenas para a área de TI, visto que, é a área que frequento e que pretendo seguir e ao ter noção desta problemática no mundo real, decidi tornar este projeto uma espécie de solução para um problema atual.

A “Jet Hire” será uma aplicação prática e moderna, que permitirá a ligação entre pessoas à procura de trabalho e empresas com vagas disponíveis. Além disso, esta plataforma incluirá funcionalidades úteis, como a possibilidade de adicionar anúncios para trabalhos temporários, incluindo vagas de verão para jovens maiores de 16 anos, garantindo que possam candidatar-se com o apoio de um responsável.

A decisão de desenvolver este projeto com tecnologias como **Next.js**, **MongoDB**, **Bootstrap**, **Mongoose** e **NextAuth** deve-se ao facto de querer explorar ferramentas que são muito utilizadas no mercado atual. Para mim, esta é uma excelente oportunidade de consolidar os conhecimentos que adquiri ao longo do curso e ao mesmo tempo criar uma solução que pode ajudar outras pessoas na mesma situação que eu enfrentei.

A plataforma terá várias funcionalidades para os dois tipos principais de utilizadores. Para quem procura emprego, será possível criar um perfil detalhado com competências e experiências, publicar anúncios com os serviços que oferece e responder diretamente a vagas publicadas pelas empresas. Para as empresas, estas poderão criar anúncios de emprego e até formular questionários personalizados para facilitar o processo de seleção. Adicionalmente, será criado um sistema de backoffice para que moderadores possam supervisionar e garantir a segurança e a qualidade do conteúdo da plataforma.

O projeto será desenvolvido em três fases. Na primeira fase, fiz uma pesquisa detalhada sobre as ferramentas e funcionalidades que seriam necessárias para criar a aplicação.

Depois disso, planeei as principais funcionalidades e desenhei *mockups* simples para organizar as ideias. Na segunda fase, comecei a construir o projeto, implementando as funções principais e corrigindo erros à medida que surgiam. Na terceira e última fase, ajustarei os detalhes finais, realizarei testes com utilizadores reais e recolherei feedback para melhorar a plataforma antes da sua entrega.

A “Jet Hire” não é apenas um projeto técnico, mas também um reflexo do meu crescimento ao longo do curso e da minha vontade de criar algo útil e inovador. Acredito que este trabalho não só mostra o que aprendi, mas também demonstra como pretendo usar esses conhecimentos no futuro.

Introduction (Inglês)

For my Professional Aptitude Project (PAP), I decided to develop “Jet Hire,” an online platform for finding and offering jobs. The idea for this project came from a personal experience during the summer holidays. At that time, I needed to find a job to save money, but I had a hard time finding opportunities in an easy and practical way. This made me realise how important it is to have a tool that makes the process simpler for both job seekers and employers.

While working on the project, I chose to focus the platform on the IT area, as it is the field I am studying and want to pursue in the future. Knowing this is a real problem for many people, I decided to make this project a way to solve it.

“Jet Hire” will be a modern and easy-to-use application that connects people looking for jobs with companies offering positions. The platform will also include helpful features, such as allowing temporary job postings, including summer jobs for people aged 16 and older, with parental approval if needed.

I decided to develop this project using technologies like Next.js, MongoDB, Bootstrap, Mongoose, and NextAuth because these tools are widely used in the job market. For me, this is a great chance to practise what I have learned during my course while creating a solution that could help others who are in the same situation I was in.

The platform will have different features for its two main user groups. Job seekers can create detailed profiles with their skills and experience, post ads about the services they offer, and apply directly to job postings from companies.

Companies, on the other hand, will be able to create job postings and design customised questionnaires to help with the hiring process. A back-office system will also be created so that moderators can monitor the platform and ensure everything is running smoothly and safely.

The project will be developed in three main stages. In the first stage, I researched the tools and features needed for the application and planned the main functions. I also created simple mockups to organise my ideas. In the second stage, I started building the project, adding the main features and fixing any problems that came up. In the third and final stage, I will focus on making improvements, testing the platform with real users, and collecting feedback to make it even better before delivering it.

“Jet Hire” is not just a technical project. It also shows how much I have grown during my course and reflects my passion for creating something useful and innovative. I believe this project not only demonstrates what I have learned but also shows how I plan to use this knowledge in the future.

Planeamento

Fundamentação

Razões que levaram à escolha do tema.

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; • Adquirir conhecimentos na Linguagens (Quais?) • Aplicar os conhecimentos adquiridos ao longo dos três anos do curso. • Explorar novas tecnologias utilizadas para o desenvolvimento da PAP. • Colocar a aplicação na Play Store. • 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+

- Arc

Requisitos Mínimos

- Sistema Operativo – Windows 10 19H1+ ou macOS 13+

- Google Chrome

Requisitos Mínimos

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

- Krita

Requisitos Mínimos

- Sistema Operativo – Windows 8.1+
- 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+

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.

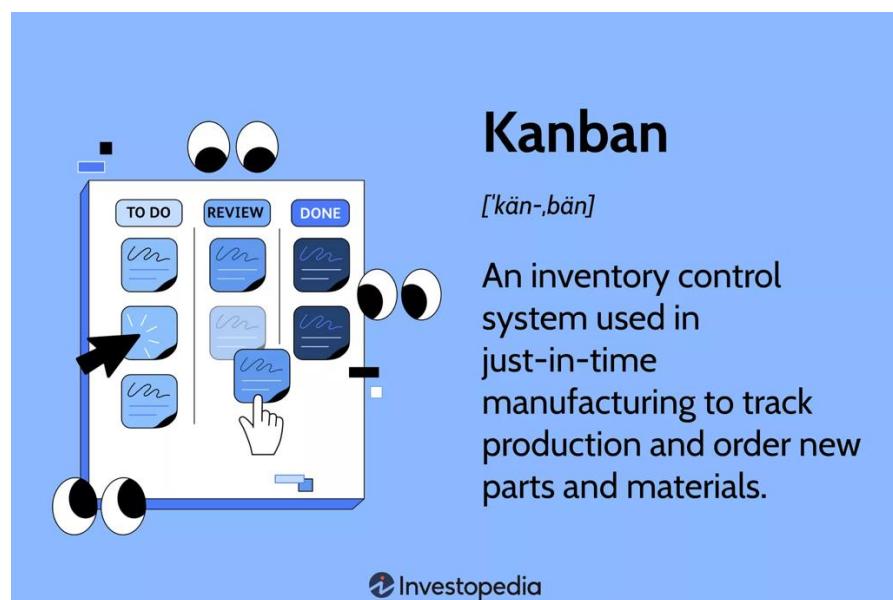


Figura 1 - Metodologia Kanban

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

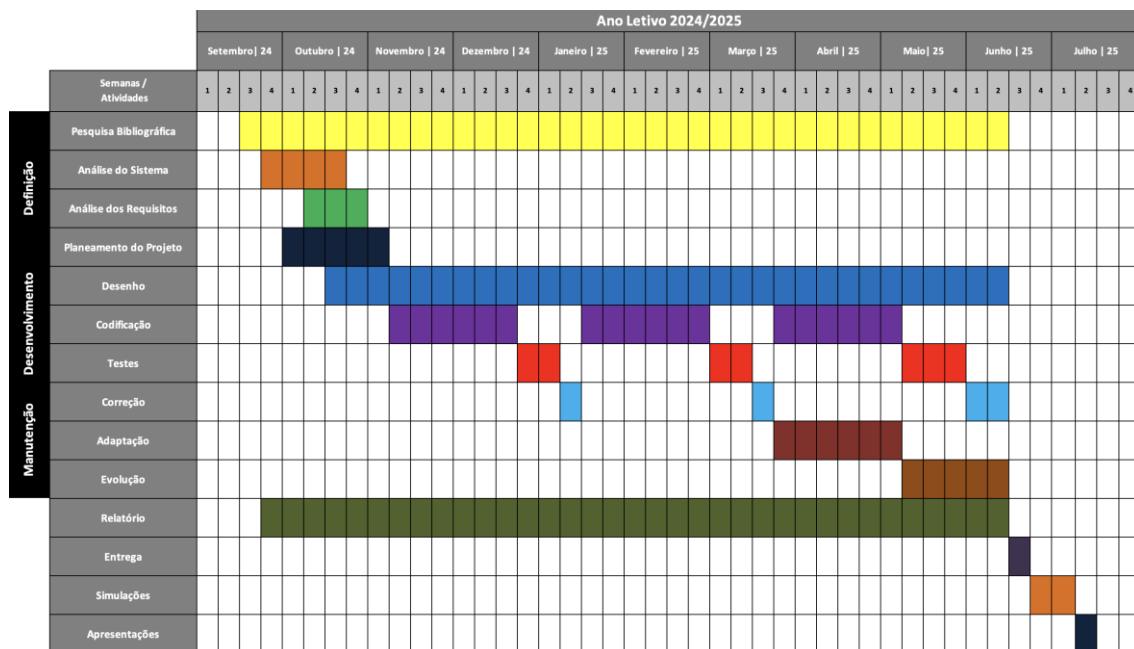


Figura 2 - Cronograma

Instalações

Visual Studio Code

Para começar a instalação do Visual Studio Code é preciso fazer *download* do instalador através do [site oficial](#).

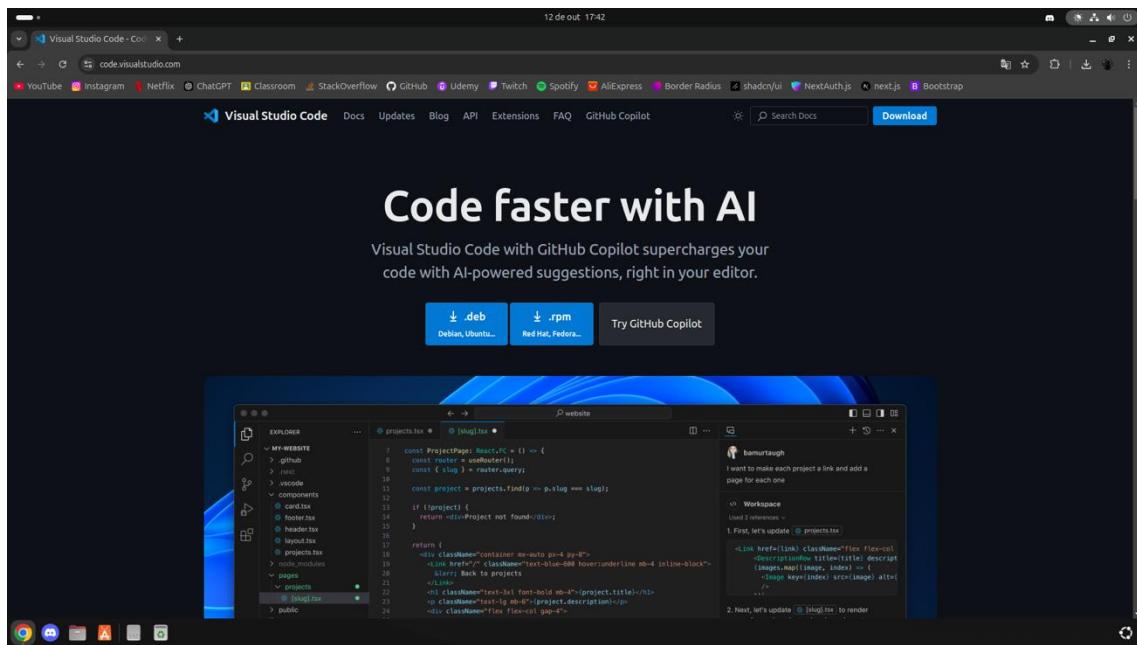


Figura 3 - Instalação Visual Studio Code: Passo 1

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

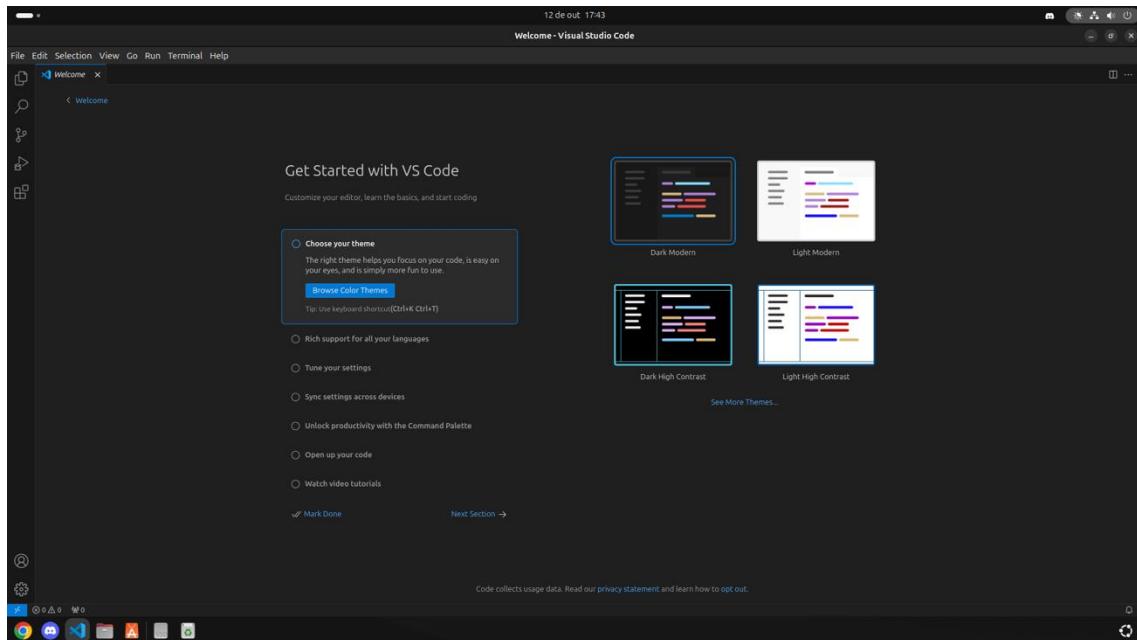


Figura 4 - Instalação Visual Studio Code: Passo 2

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

- **Auto Rename Tag**
- **Error Lens**
- **Min Theme**
- **MongoDB for VS Code**
- **Symbols**

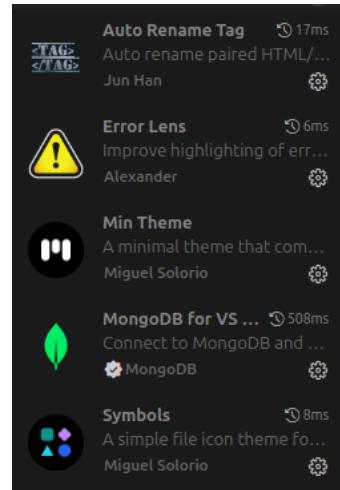


Figura 5 - Extensões

MongoDB Compass

Tal como o VS Code é necessário fazer *download* do instalador através do [site oficial](#).

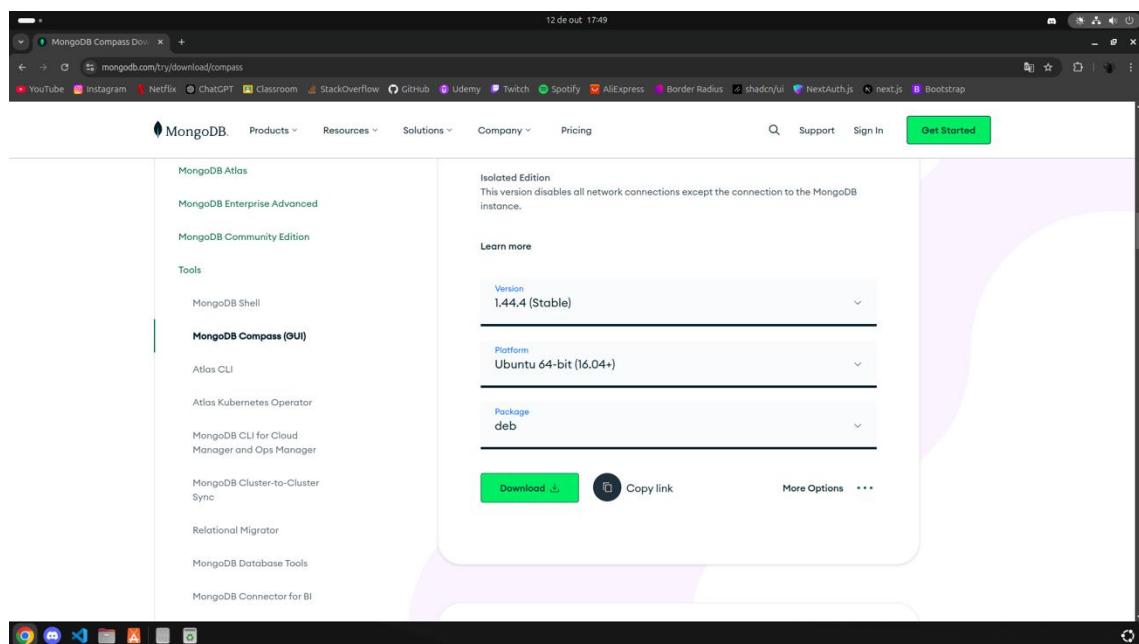


Figura 6 - Instalação MongoDB Compass: Passo 1

De seguida, é só executar o instalador.

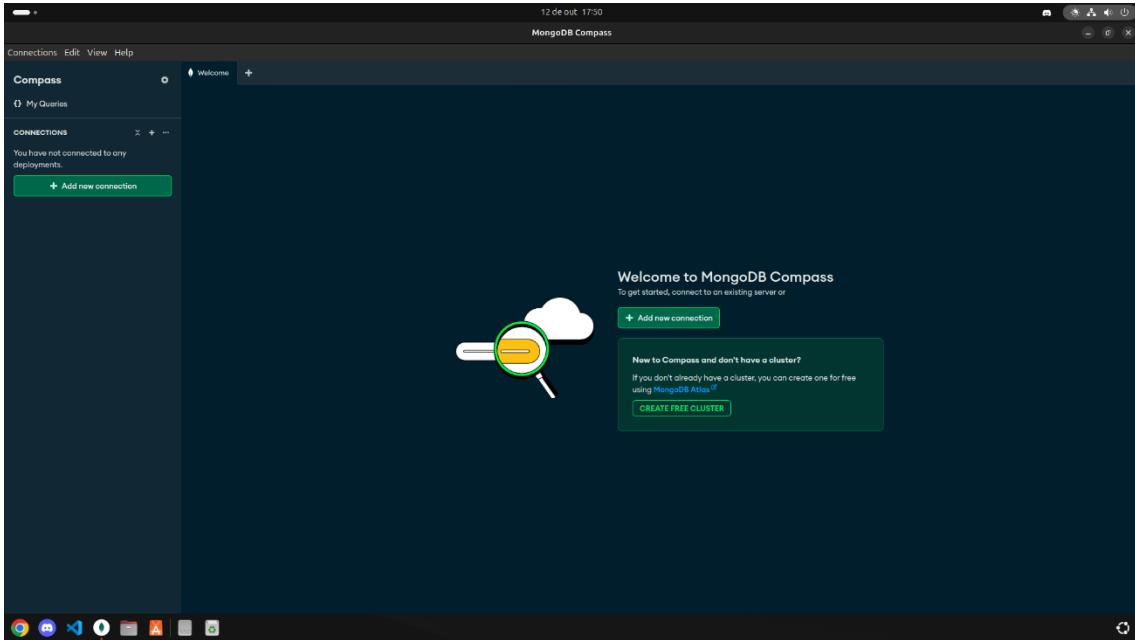
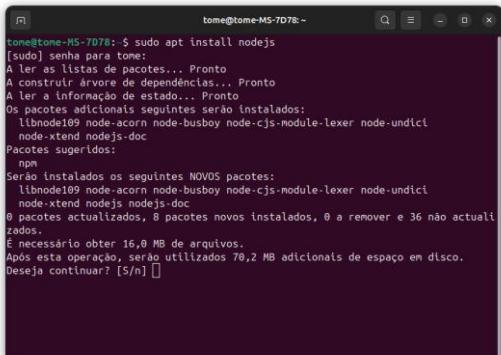


Figura 7 - Instalação MongoDB Compass: Passo 2

Node.js (node / npm / npx)

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

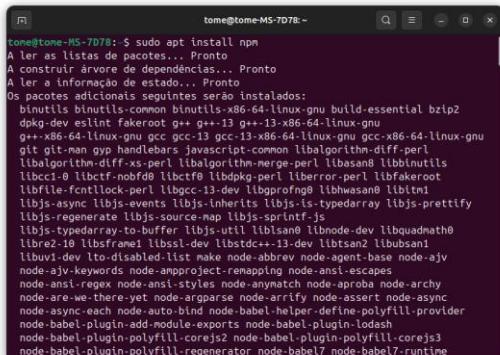
sudo apt install nodejs



```
tome@tome-MS-7D7B: $ 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 8 - *sudo apt install nodejs*

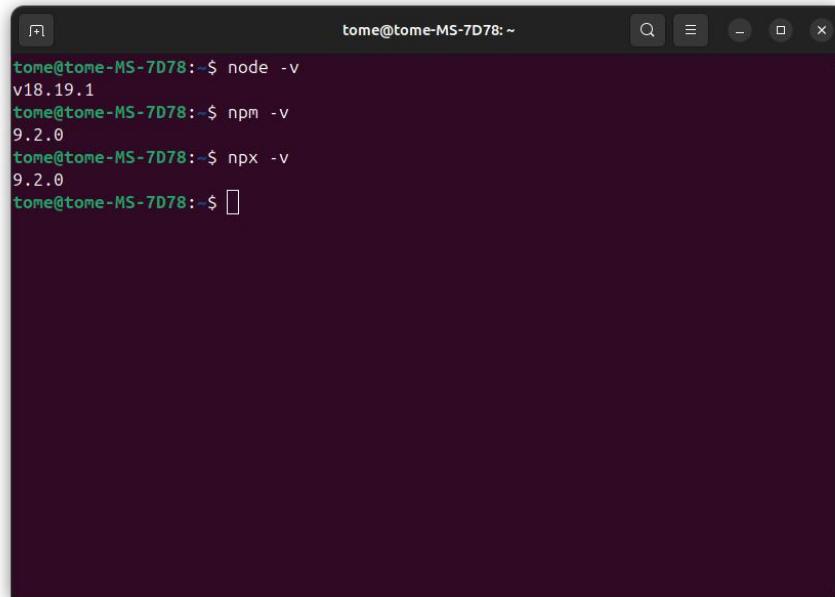
sudo apt install npm



```
tome@tome-MS-7D7B: $ sudo apt install npm
[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:
  binutils binutils-common binutils-x86-64-linux-gnu build-essential bztp
  dpkg-dev eslintr fakroot g++ g++-13 g++-13-x86_64-linux-gnu
  g++-x86_64-linux-gnu gcc gcc-13 gcc-13-x86_64-linux-gnu gcc-x86_64-linux-gnu
  git git-man gnu-handbook javascript-common libalgorithm-diff-perl
  libalgorithm-diff-xs-perl libalgorithm-merge-perl libasan8 libbinutils
  libcc1-0 libctf-nobfd0 libcrt0 libdpkg-perl liberror-perl libfakeroot
  libfile-fcntllock-perl libgcc-13-dev libgprofng libhwسان libitm
  libjs-async libjs-events libjs-inherits libjs-is-typedarray libjs-prettify
  libjs-regenerate libjs-source-map libjs-sprintf.js
  libjs-typedarray-to-buffer libjs-util libtsan libnode-dev libquadmath0
  libtsan-10 libtsan1 libtsan1-dev libtsan1-13-dev libtsan2 libubsan1
  libubsan1-dev libubsan1-13-dev libubsan1-13-dev libubsan2 libubsan2
  libubsan2-13-dev libubsan2-13-dev libubsan3 libubsan3-13-dev libubsan4
  node-ajv node-ajv5 node-ansi-escapes node-aniso-regex node-ansi-styles node-anymatch node-aproba node-archy
  node-are-we-there-yet node-aparse node-arrify node-assert node-async
  node-async-each node-auto-blind node-babel-helper node-babel-polyfill-provider
  node-babel-plugin-add-module-exports node-babel-plugin-lodash
  node-babel-plugin-polyfill-corejs2 node-babel-plugin-polyfill-corejs3
  node-babel-plugin-polyfill-regenerator node-babel7 node-babel7-runtime 
```

Figura 9 - *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 10 - 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
```

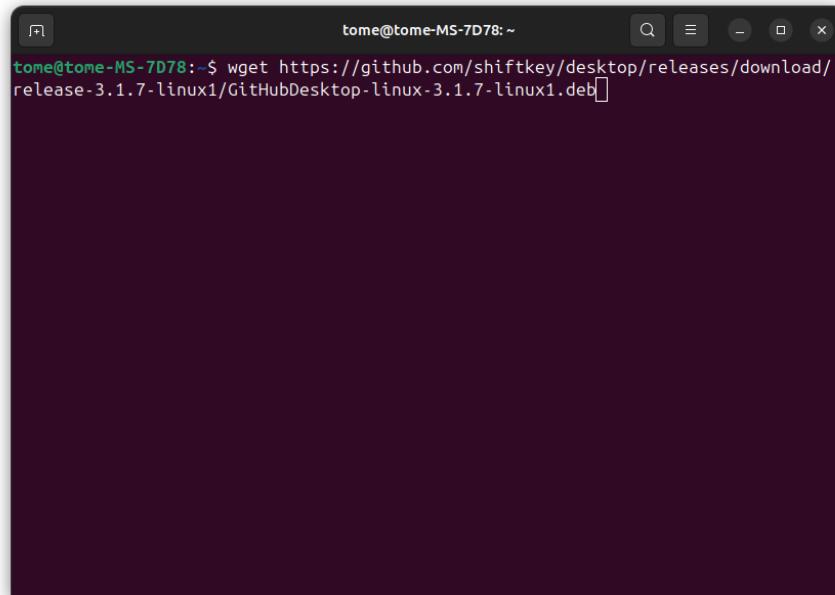


Figura 11 - Instalação GitHub Desktop: Passo 1

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

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

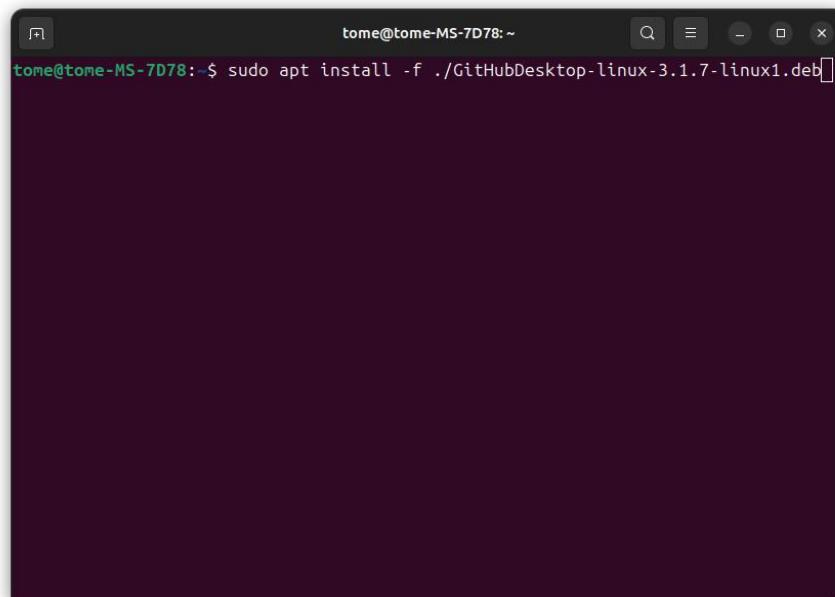


Figura 12 - Instalação GitHub Desktop: Passo 2

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

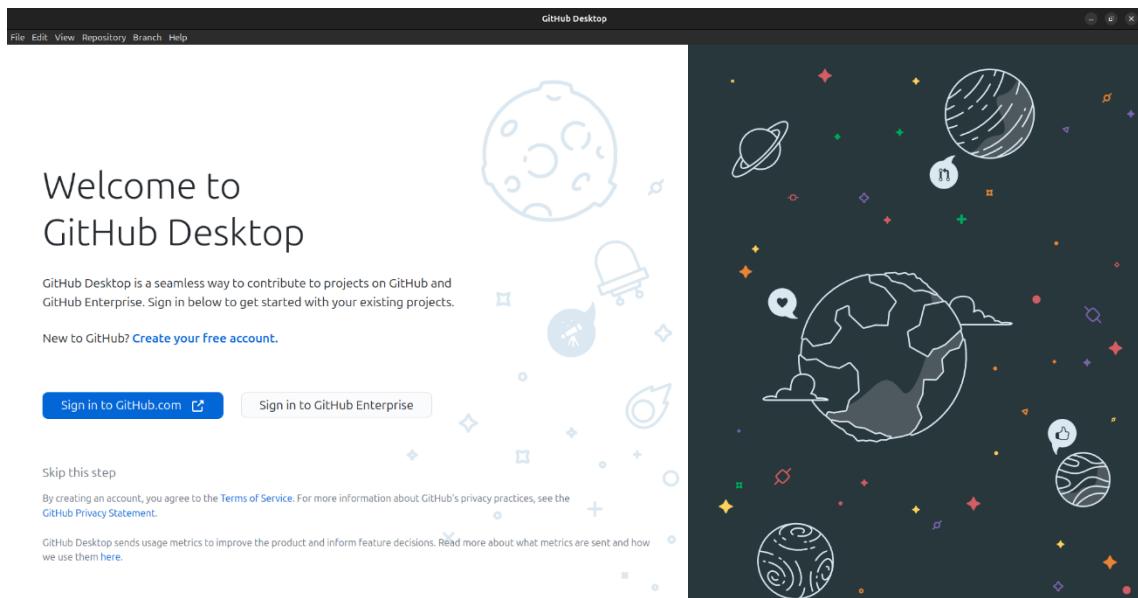


Figura 13 - Instalação GitHub Desktop: Passo 3

Inkscape

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

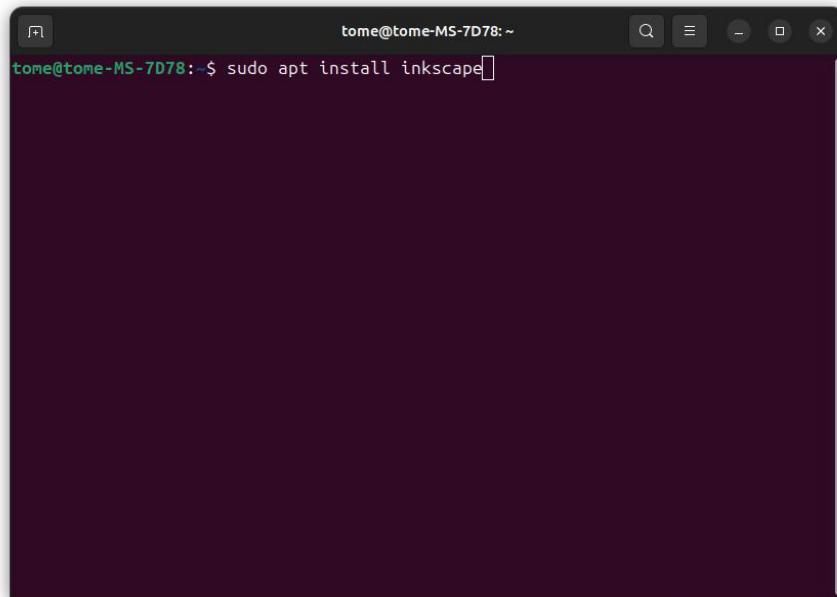


Figura 14 - Instalação do Inkscape: Passo 1

A instalação está concluída e pronta a ser utilizada.

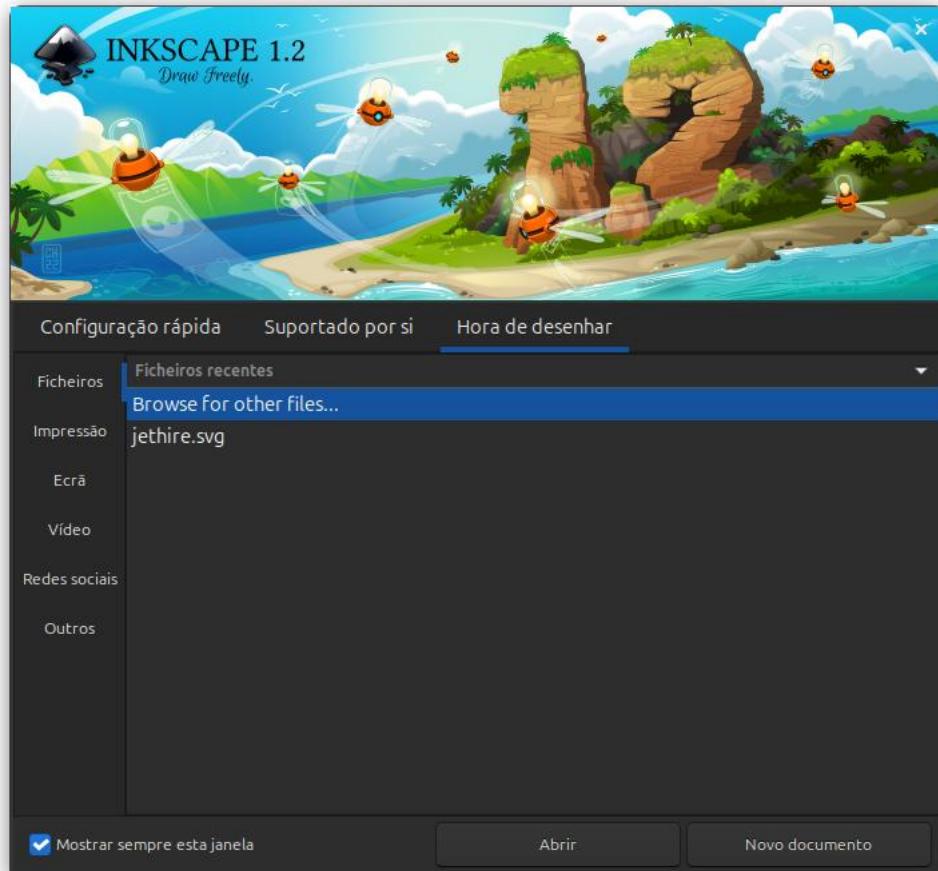


Figura 15 - Instalação do Inkscape: Passo 2

Krita

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

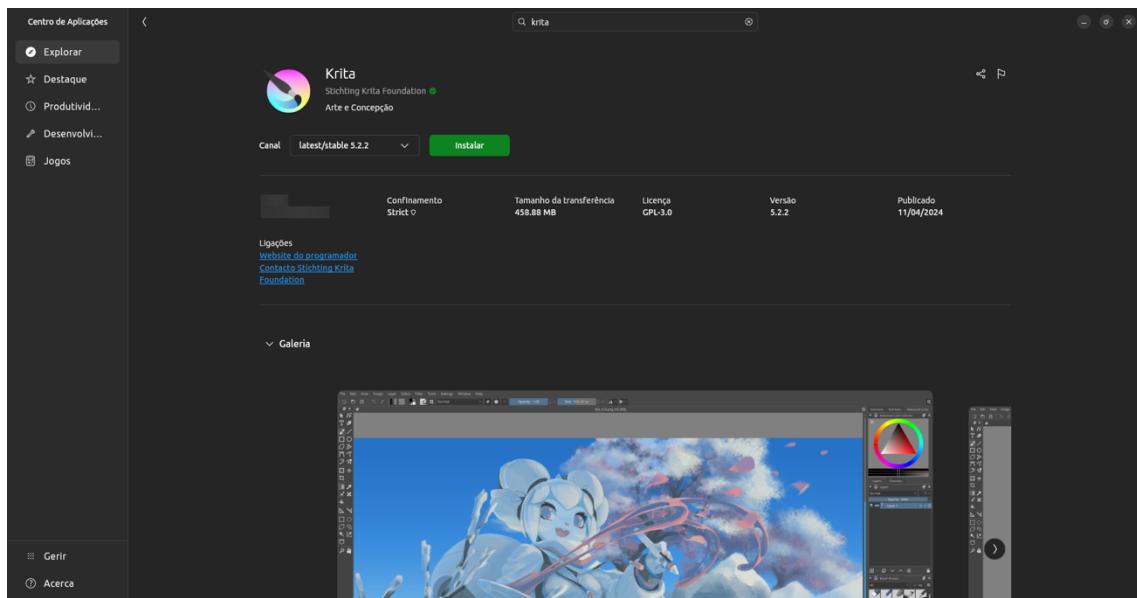


Figura 16 - Instalação do Krita: Passo 1

A aplicação está pronta a ser utilizada.



Figura 17 - Instalação do Krita: Passo 2

Postman

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

Download Postman

Download the app to get started using the Postman API Platform today. Or, if you prefer a browser experience, you can try the web version of Postman.

The Postman app

Download the app to get started with the Postman API Platform.

[Mac Intel Chip](#) [Mac Apple Chip](#)

By downloading and using Postman, I agree to the [Privacy Policy](#) and [Terms](#).

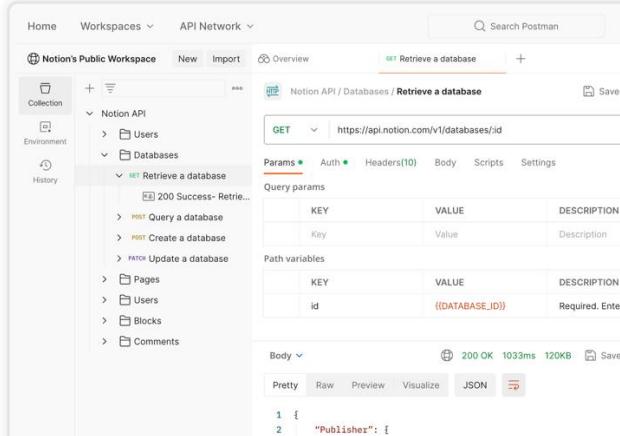
[Release Notes →](#)

Not your OS? Download for Windows ([x64](#)) or Linux ([x64, arm64](#))

Postman on the web

Access the Postman API Platform through your web browser. Create a free account, and you're in.

[dl.pstmn.io/download/latest/osx_arm64](#) [Try the Web Version](#)



The screenshot shows the Postman web interface. At the top, there are tabs for Home, Workspaces, and API Network. Below that, a search bar says "Search Postman". The main area shows a workspace titled "Notion's Public Workspace". A list of collections is on the left, including "Notion API", "Users", "Databases", "Pages", "Comments", and "Blocks". Under "Databases", there is a request for "Retrieve a database" with a status of "200 Success - Retriev...". The request details show a GET method, URL https://api.notion.com/v1/databases/:id, and a query parameter "id" with value "{{DATABASE_ID}}". The response body is shown in JSON format:

```

1 {
2   "published": {
3     ...
4   }
5 }
  
```

Figura 18 - Instalação do Postman: Parte 1

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

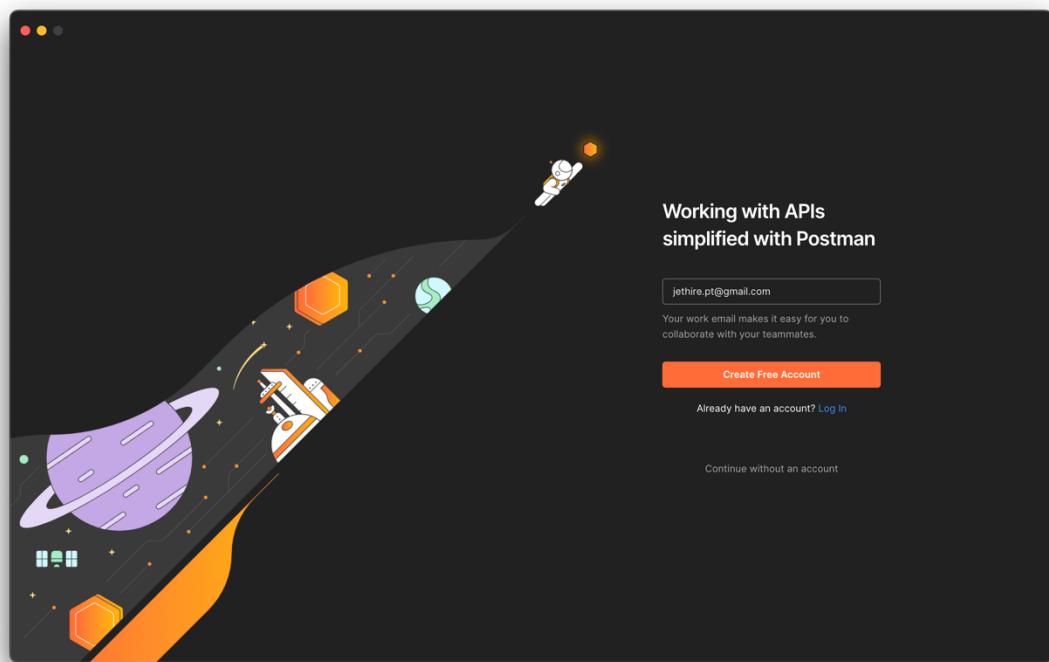


Figura 19 - Instalação do Postman: Parte 2

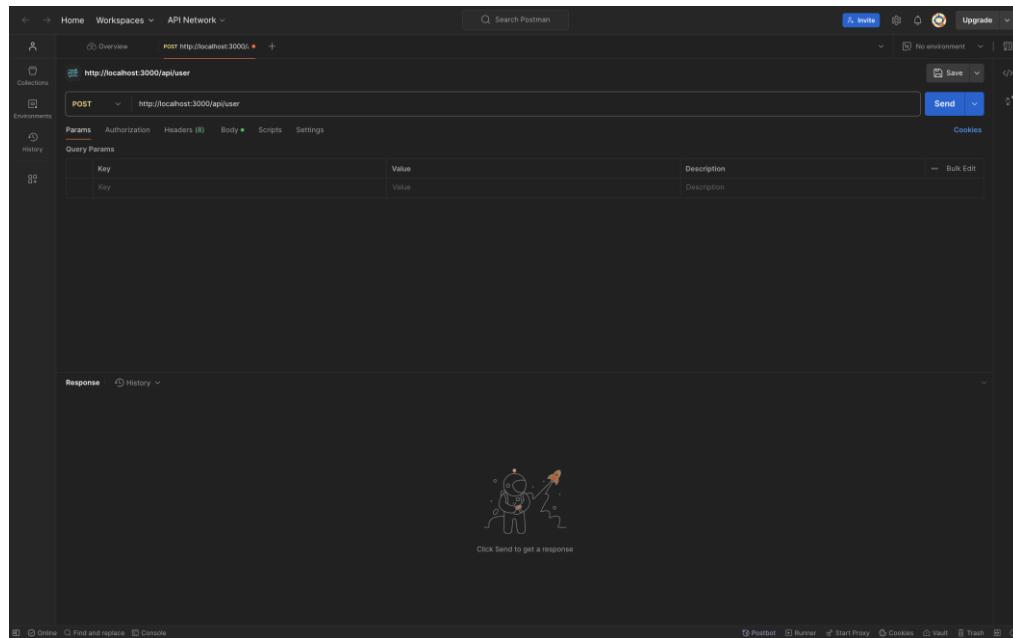


Figura 20 - Instalação do Postman: Parte 3

Sincronização entre dois computadores

Como utilizei dois computadores para a realização projeto utilizei o GitHub Desktop para fazer esta sincronização.

Criação do repositório

Para começar, é preciso criar o respetivo repositório onde se vai guardar todos os ficheiros:

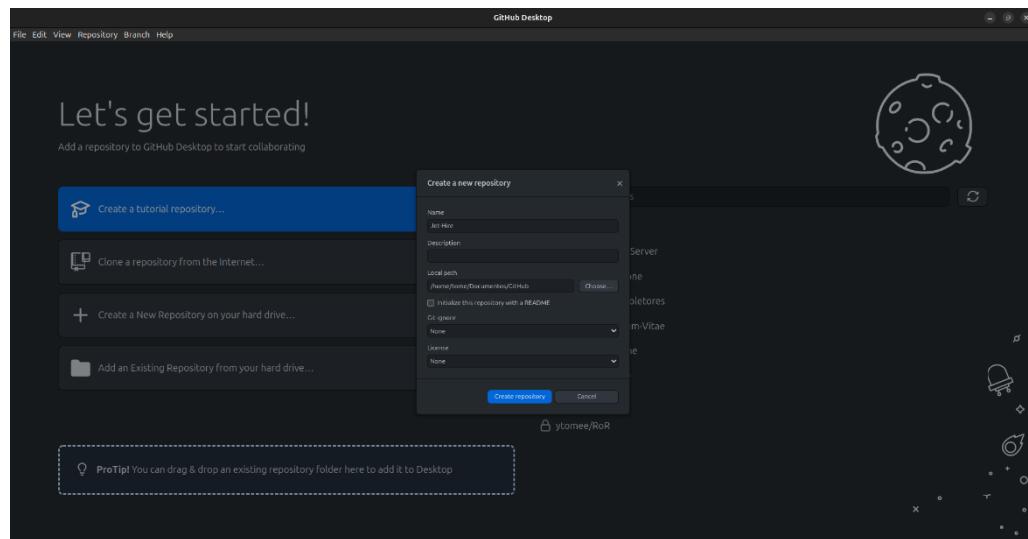


Figura 21 - Sincronização entre dois computadores: Passo 1

Depois da criação do repositório é preciso publicá-lo:

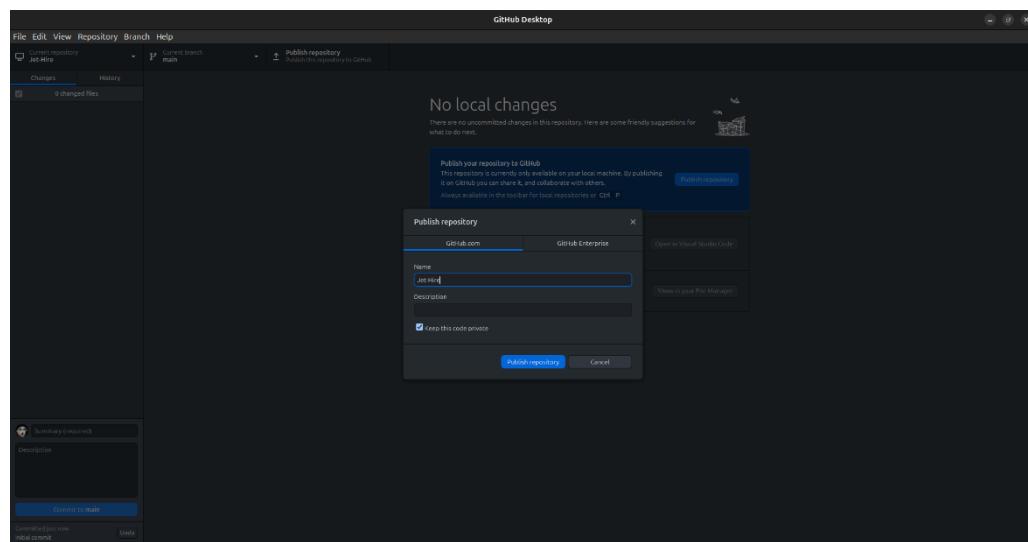


Figura 22 - Sincronização entre dois computadores: Passo 2

Push

O Github vai criar uma pasta dentro da pasta Documentos do computador e é só preciso passar os ficheiros para a mesma. De seguida, volta-se á aplicação e verifica-se as alterações. No fim, dá-se *commit*, que vai fazer todas as alterações na *cloud*.

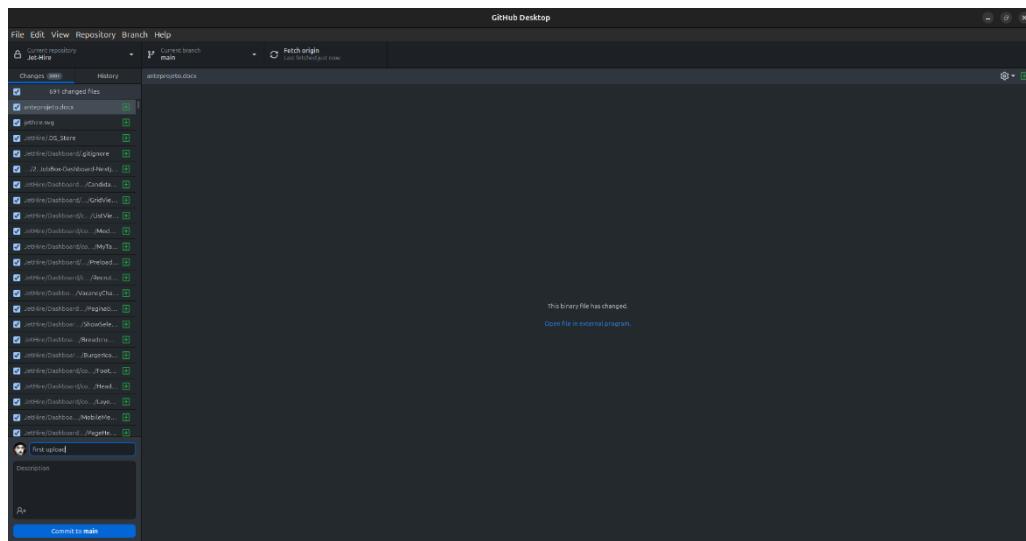


Figura 23 - Sincronização entre dois computadores: Passo 3

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

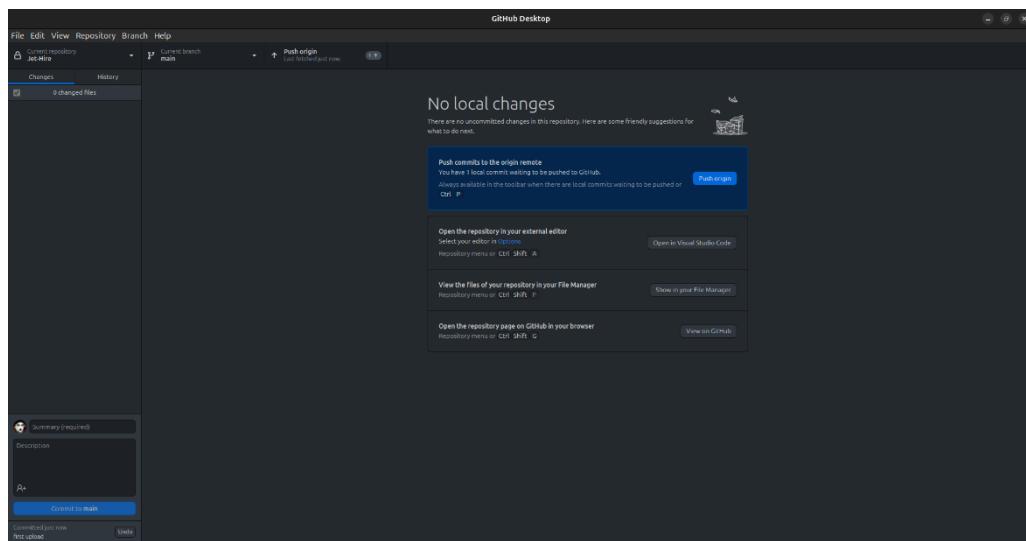


Figura 24 - Sincronização entre dois computadores: Passo 4

Para confirmar se o repositório foi criado corretamente pode-se confirmar pelo browser:

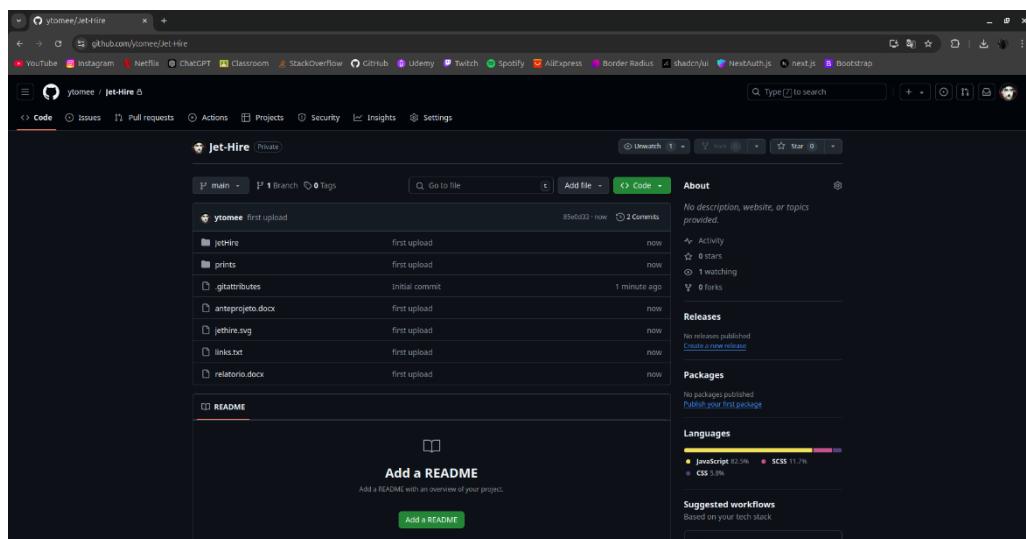


Figura 25 - Sincronização entre dois computadores: Passo 5

Pull

Caso o repositório ainda não exista no computador, tem que se clonar o mesmo.

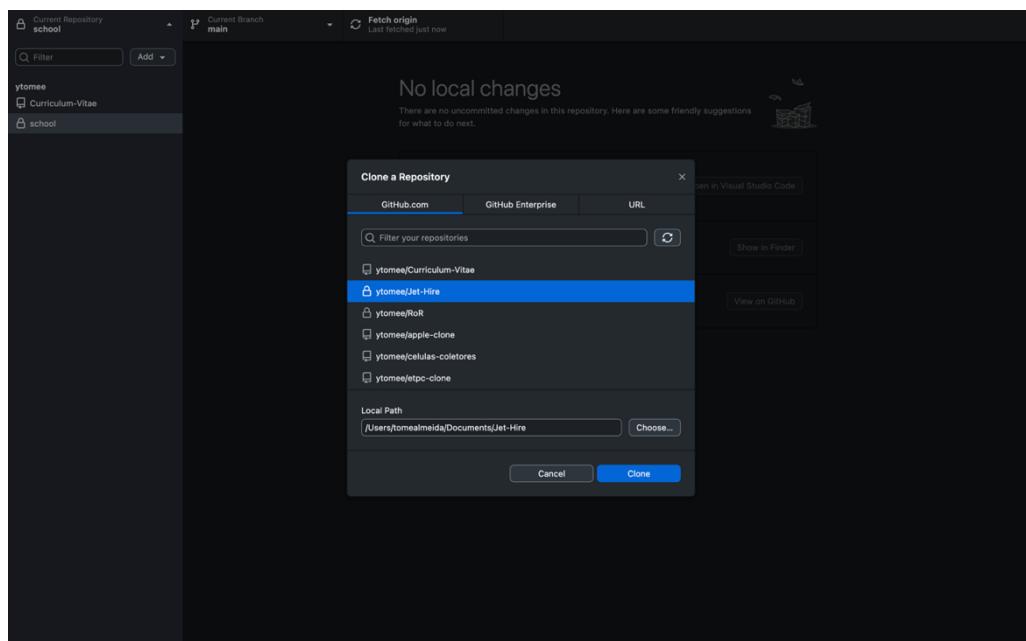


Figura 26 - Sincronização entre dois computadores: Passo 6

Se o repositório já existir, dá-se *fetch* para verificar se existem alterações na pasta, caso existam, dá-se *push* para voltar a alterar os dados:

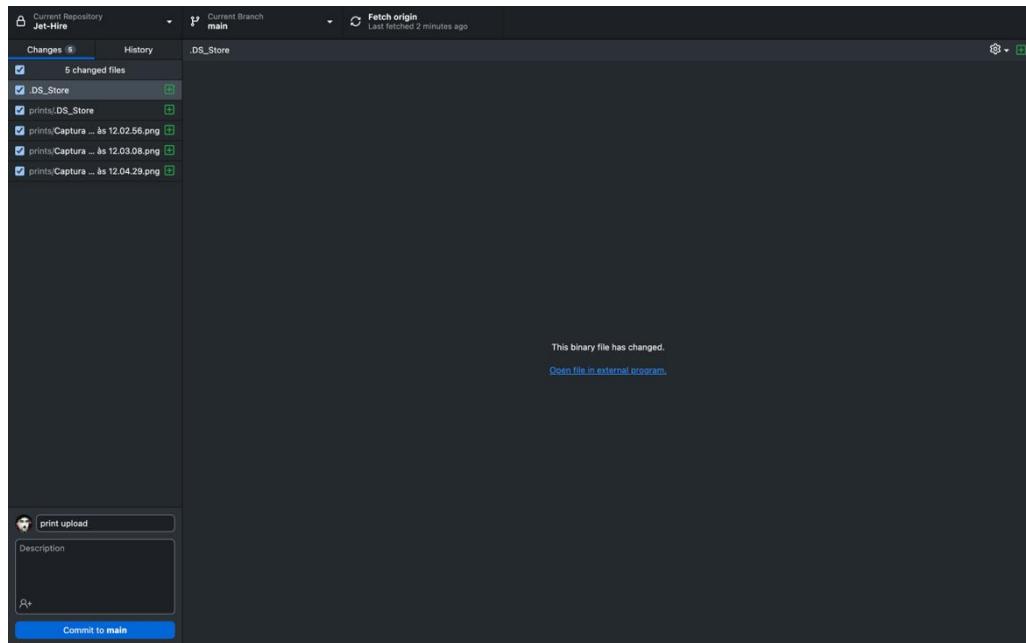


Figura 27 - Sincronização entre dois computadores: Passo 7

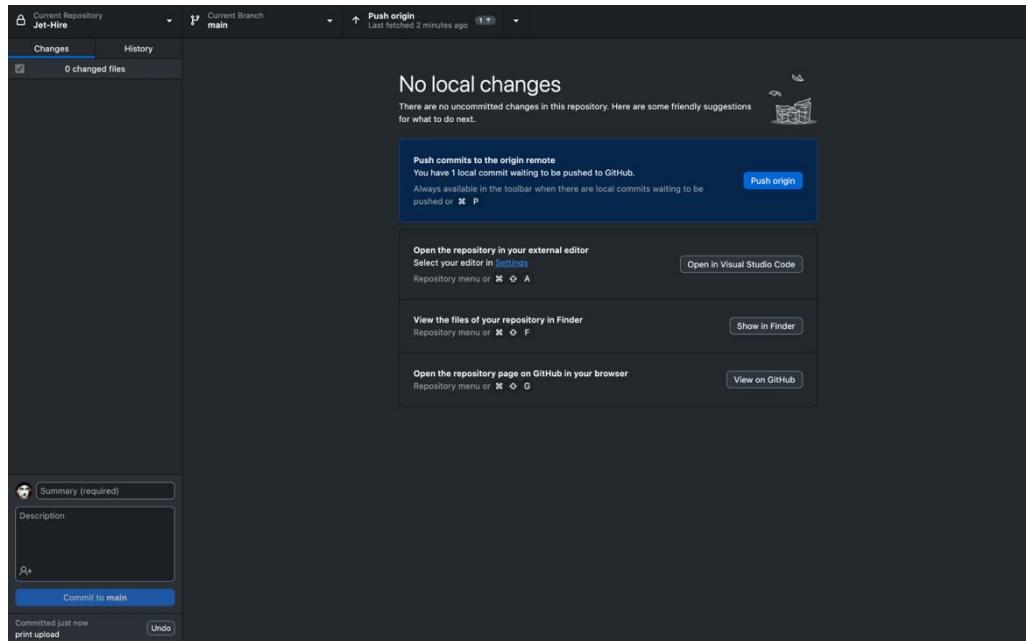


Figura 28 - Sincronização entre dois computadores: Passo 8

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.

Para uma melhor organização, optei por criar dois *sitemaps* separados, um para as páginas principais e outro para a *dashboard* geral.

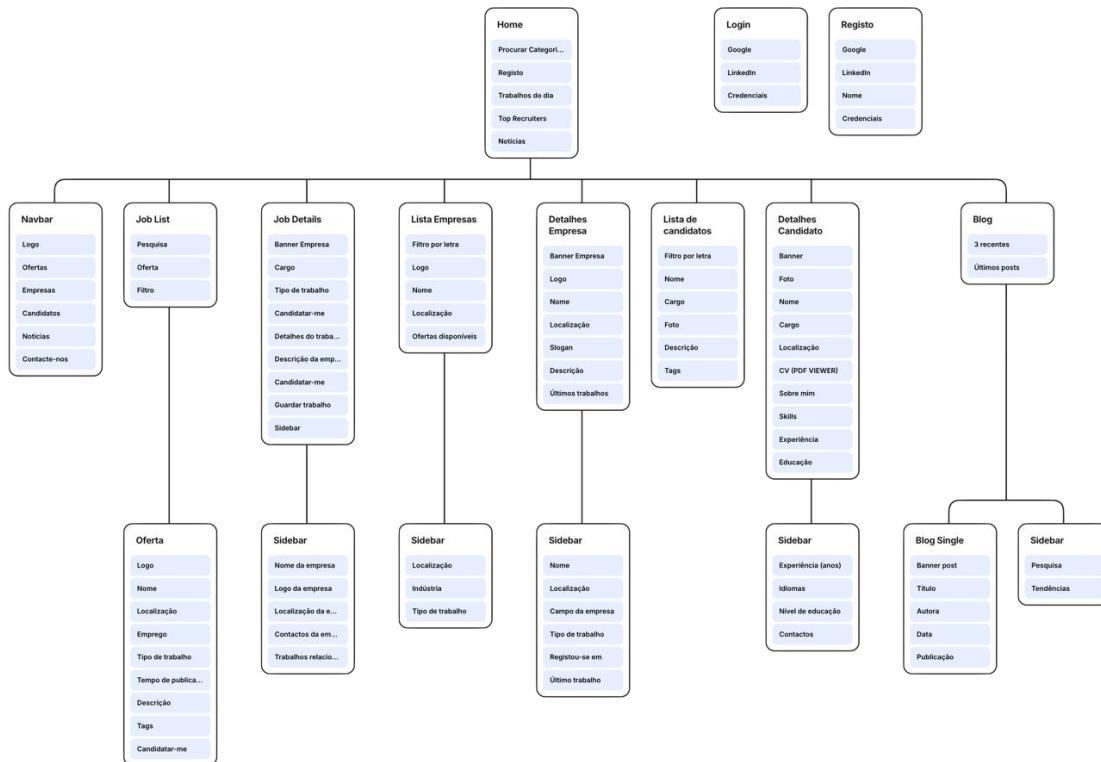


Figura 29 – Sitemap principal

Projeto

Instalação do template

Após ter escolhido o template, o próximo passo é o *download*.

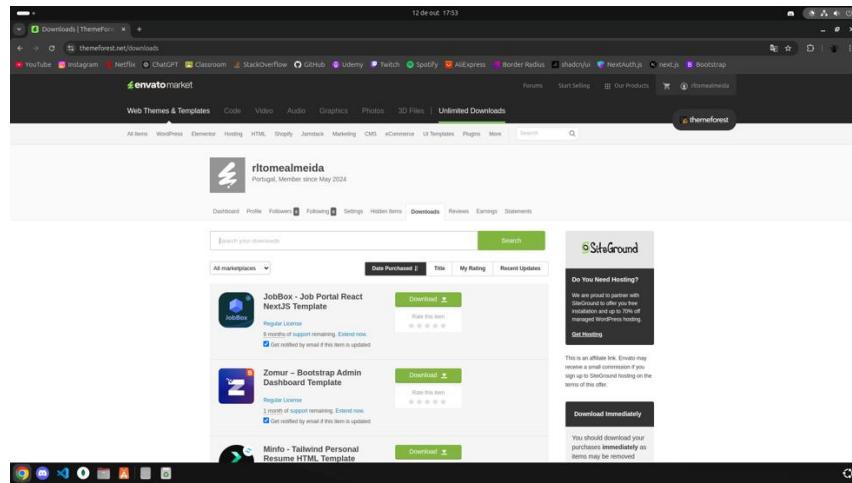


Figura 30 - Download do template

De seguida, abre-se a pasta anteriormente transferida através do Visual Studio Code. Cria-se um novo terminal e acede-se á pasta específica através do comando “cd [Nome da pasta]”. Quanto estivermos dentro da pasta passamos á instalação dos *node_modules* e do ficheiro *package-lock.json* através do comando “npm install”.

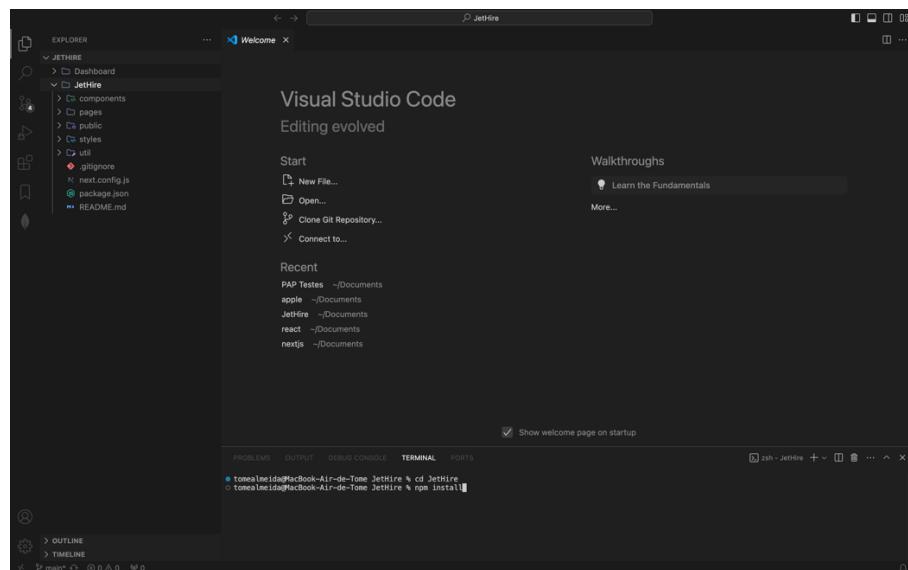


Figura 31 - Instalação dos node_modules

Quando a instalação tiver terminado, o projeto está pronto a ser executado através do comando “npm run dev”.

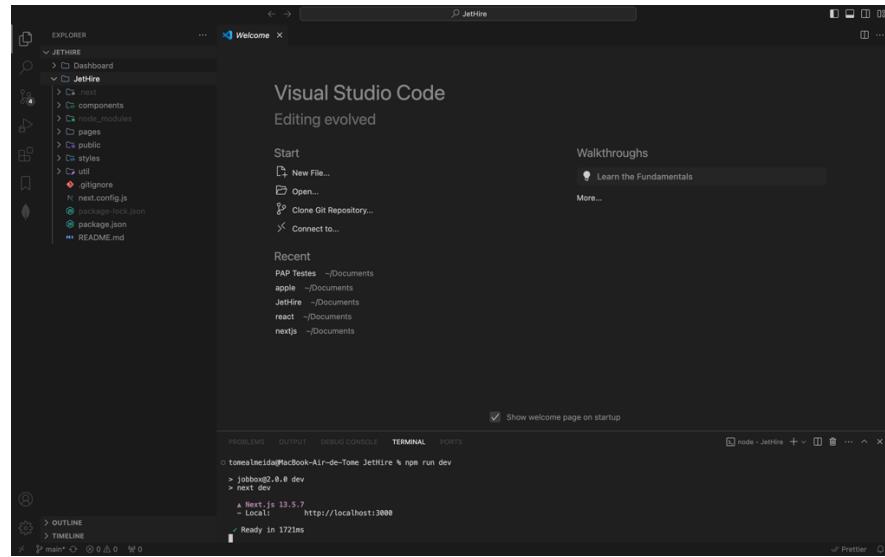


Figura 32 - npm run dev

Ao abrir o link apresentado no terminal, temos acesso ao projeto em tempo real.



Figura 33 - Projeto executado

Logótipo

Decidi criar o logótipo de uma forma completa, mas como o meu projeto consiste na inserção de novos elementos numa empresa, separei um dos quatro elementos para destacar o elemento sozinho que vai posteriormente integrar a empresa para a complementar.

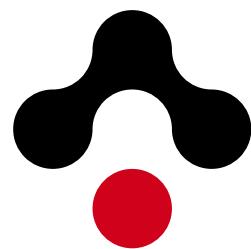


Figura 34 - Logótipo

Escolha da paleta de cores

Cor principal

A escolha da paleta de cores do meu projeto foi feita através de um guião sobre a estrutura de cores da interface de websites. Utilizei o espaço de cor HSB (*Hue, Saturation, Brightness*) para conseguir criar os tons mais escuros e mais claros. Comecei por escolher a minha cor principal que foi o vermelho.



Figura 35 - Escolha da cor principal

Depois da escolha da cor é preciso definir os diferentes tons da cor. O método que utilizei foi escolher o tom mais escuro e mais claro antes dos intermediários.

Para o tom mais claro aumentei a luminosidade e diminuí a saturação.

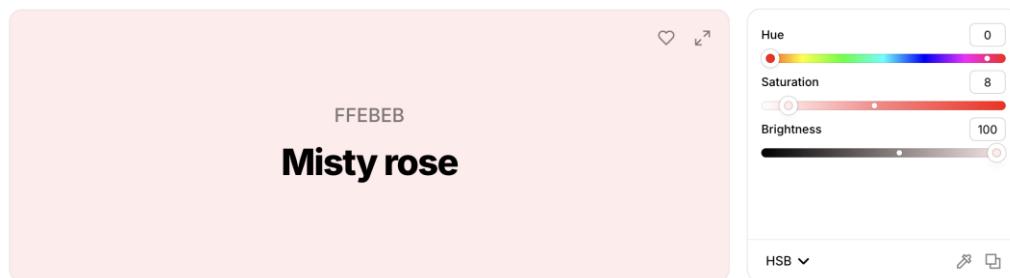


Figura 36 - Escolha do tom claro

Para o tom mais escuro, o processo é o inverso. Diminui-se a luminosidade e aumenta-se a saturação.

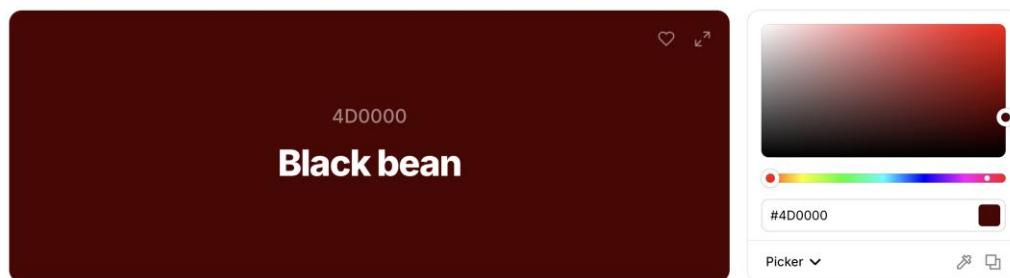


Figura 37 - Escolha do tom escuro

Depois de ter estes três tons, utilizei uma ferramenta que gera gradientes entre duas cores. Primeiro do tom mais claro até à cor principal.

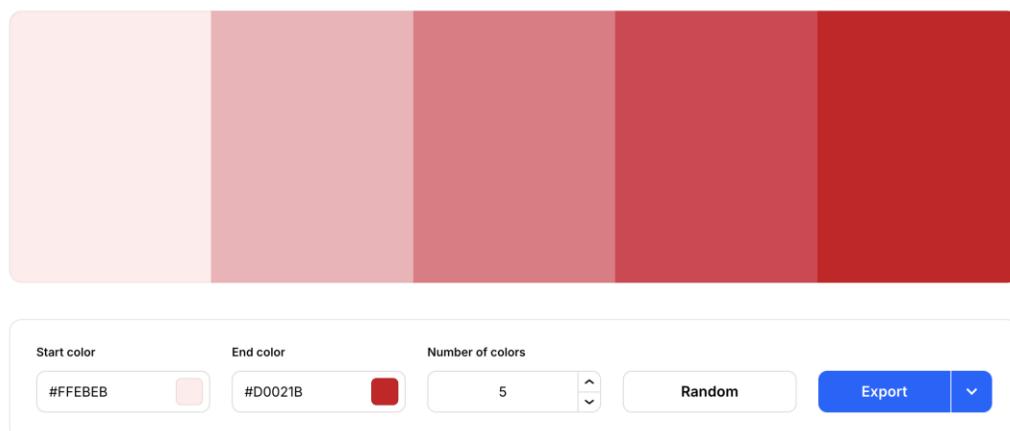


Figura 38 - Criação de um gradiente claro

De seguida, da cor principal até ao tom mais escuro.

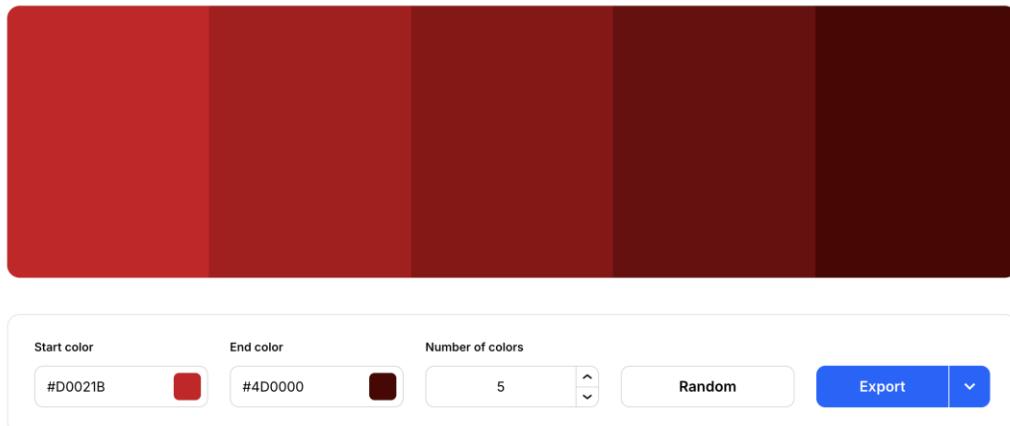


Figura 39 - Criação de um gradiente escuro

E assim, tenho a paleta de cores principais do projeto.



Figura 40 - Vermelho

Cores secundárias

As cores secundárias vão ser utilizadas em avisos, *pop-ups*, informações, etc. Para estas cores optei pelo amarelo para avisos, azul para informações, verde para confirmações e o vermelho novamente para erros. Repete-se todo o processo da cor principal.



Figura 41 - Amarelo



Figura 42 - Azul

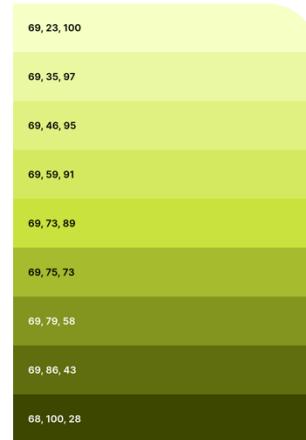


Figura 43 - Verde

Cores neutras

As cores neutras são as cores que vão acompanhar a cor principal ao longo do site. Estas cores vão desde o branco até ao preto. Novamente o mesmo processo para esta paleta.

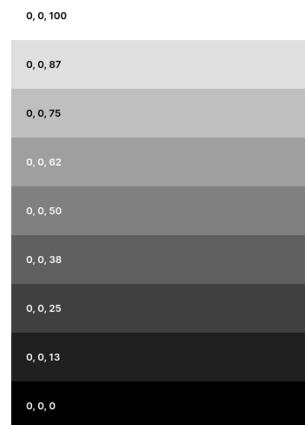


Figura 44 - Cores neutras

Criação da paleta no projeto

Para adicionar todas as cores no projeto é preciso adicioná-las como variáveis globais. É necessário criar um root dentro do ficheiro CSS e adicionar todas as cores.

```

1  :root {
2    /* BRAND COLOR */
3    --red-100: #FFECEB;
4    --red-200: #F3B1B7;
5    --red-300: #E87783;
6    --red-400: #DC3C4F;
7    --red-500: #D0021B;
8    --red-600: #AF0214;
9    --red-700: #8F010E;
10   --red-800: #6E0107;
11   --red-900: #4D0000;
12 }

```

Figura 45 - Exemplo de variáveis globais

Quando for pretendido utilizar alguma destas cores, basta utilizar a propriedade `var` dentro do ficheiro CSS. De seguida, está o exemplo:

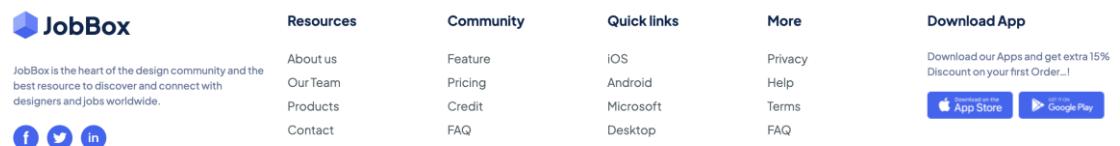
```
a {
  color: var(--red-900);
}
```

Edição do *template*

Após as alterações anteriores, comecei a editar o *template* para a estrutura planeada no *sitemap* e para alterar para a paleta de cores escolhida.

Rodapé

Na edição do rodapé, apenas removi o primeiro bloco por achar desnecessário, editei o *copyright*, e traduzi as políticas e os termos.



The footer contains the following sections:

- JobBox**: Includes the JobBox logo and a brief description: "JobBox is the heart of the design community and the best resource to discover and connect with designers and jobs worldwide." Below it are three social media icons for Facebook, Twitter, and LinkedIn.
- Resources**: Links to About us, Our Team, Products, and Contact.
- Community**: Links to Feature, Pricing, Credit, and FAQ.
- Quick links**: Links to iOS, Android, Microsoft, and Desktop.
- More**: Links to Privacy, Help, Terms, and FAQ.
- Download App**: Buttons for the App Store and Google Play.
- Copyright**: Copyright © 2022. JobBox all right reserved.
- Links**: Privacy Policy, Terms & Conditions, and Security.

Figura 46 - Rodapé inicial

Figura 47 - Rodapé final

Barra de navegação

Para a barra de navegação, comecei por alterar o logótipo e logo de seguida a estrutura dos botões.



Figura 48 - Barra de navegação inicial



Figura 49 - Barra de navegação final

Página principal

Secção “Destaque principal”

A página principal está dividida em várias secções, sendo o **destaque principal** a primeira delas. Esta secção apresenta o slogan da Jet Hire acompanhado por imagens que ilustram o propósito da plataforma. Para destacar ainda mais, foi escolhido um gradiente entre preto e vermelho como fundo para as imagens.

Além disso, os botões foram ajustados para melhorar a navegação: um direciona o utilizador para uma secção seguinte da página, enquanto o outro o leva à página Sobre Nós.

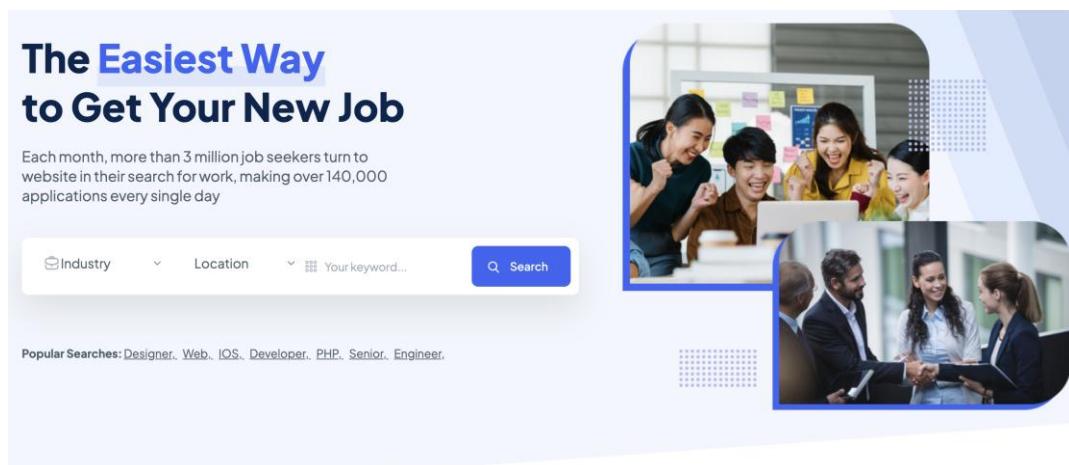


Figura 50 - Destaque principal inicial



Figura 51 - Destaque principal final

Secção “Slider de categorias”

Para manter uma estrutura minimalista à vista do utilizador quando este abre a plataforma, dentro do *viewport*, só adicionei um *slider* para as categorias que dividem as diferentes ofertas.

Alterei também, a forma como adiciono cada objeto ao *slider* utilizando uma estrutura **JSON-like** onde para cada um defino o seu ícone, o seu título e a sua descrição. Importei os ícones do website Heroicons onde apenas é necessário clicar no botão de copiar o código do *svg* para o adicionar em cada objeto.

```
const data = [
```

```
{
  icon: (
    <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24"
strokeWidth={1.5} stroke="currentColor">
      <path strokeLinecap="round" strokeLinejoin="round" d="m6.75 7.5 3 2.25-3
2.25m4.5 0h3m-9 8.25h13.5A2.25 2.25 0 0 0 21 18V6a2.25 2.25 0 0 0 0-2.25-
2.25H5.25A2.25 2.25 0 0 0 3 6v12a2.25 2.25 0 0 0 2.25 2.25Z" />
    </svg>
  ),
  title: "Desenvolvimento",
  description: "Software, Web, Jogos, etc..."
},
(...),
];
}
```

Para este bloco, foi necessário editar a biblioteca **Swiper.js**. Comecei por adicionar um **loop** para o slider se repetir, defini o número de slides visíveis para 4 com o **slidesPerView** para deixar o espaço mais “limpo” e organizado, ativei o **autoplay** para tornar tudo mais dinâmico, alterei a velocidade do *slider* com a propriedade **speed** e **delay**.

Para ter um design responsivo, defini **breakpoints** dentro do *slider* onde defino a resolução pretendida e defino a quantidade de objetos visíveis e o espaço entre eles. De seguida, está apresentado a função responsável pelo *slider* de categorias.

```
const CategorySlider = () => {
  return (
    <>
      <div className="swiper-container swiper-group-5">
        <Swiper
          slidesPerView={4}
          spaceBetween={30}
          autoplay={{
            delay: 2200,
            disableOnInteraction: false,
          }}
          speed={3000}
          loop={true}
          breakpoints={{
            320: {
              slidesPerView: 1,
            }
          }}
        </Swiper>
      </div>
    </>
  );
}
```

```

        spaceBetween: 30
    },
    575: {
        slidesPerView: 2,
        spaceBetween: 30
    },
    767: {
        slidesPerView: 2,
        spaceBetween: 30
    },
    991: {
        slidesPerView: 3,
        spaceBetween: 30
    },
    1199: {
        slidesPerView: 4,
        spaceBetween: 30
    }
})
className="swiper-wrapper pb-70 pt-5"
>
{data.map((item, i) => (
    <SwiperSlide key={i}>
        <div className="swiper-slide hover-up">
            <Link legacyBehavior href="/jobs-list">
                <a>
                    <div className="item-logo">
                        <div className="image-left">
                            {item.icon}
                        </div>
                        <div className="text-info-right">
                            <h4>{item.title}</h4>
                            <p className="font-xs">
                                {item.description}
                            </p>
                        </div>
                    </div>
                </a>
            </Link>
        </div>
    </SwiperSlide>
))
)
</Swiper>
</div>
</>
);
};

export default CategorySlider;

```

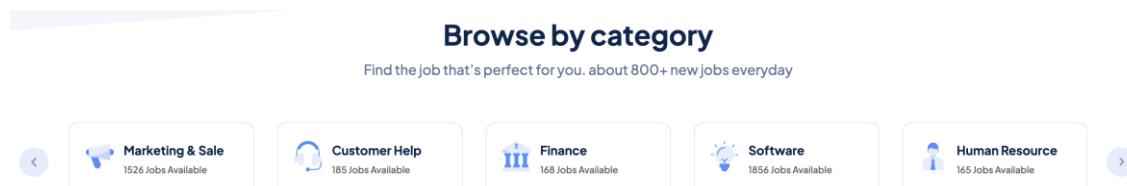


Figura 52 - Slider de categorias inicial

Procurar por categoria

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



Figura 53 - Slider de categorias final

Secção “Como funciona”

Posteriormente, adaptei a secção “Como funciona”. Esta secção é que a que é redirecionada quando se clica no botão do **destaque principal**. A edição foi simples, onde apenas foi realizada a troca de cores e texto.

How It Works

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

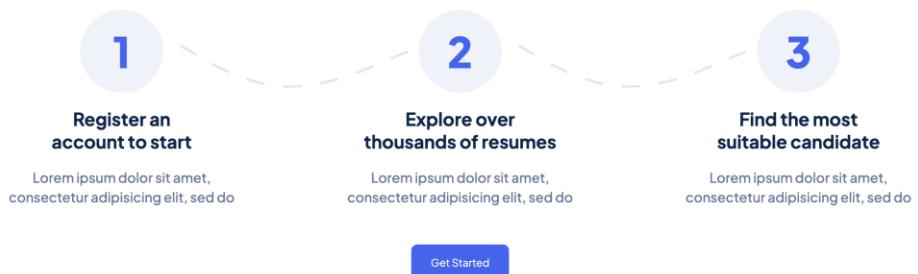


Figura 54 - Como funciona inicial

Como funciona

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



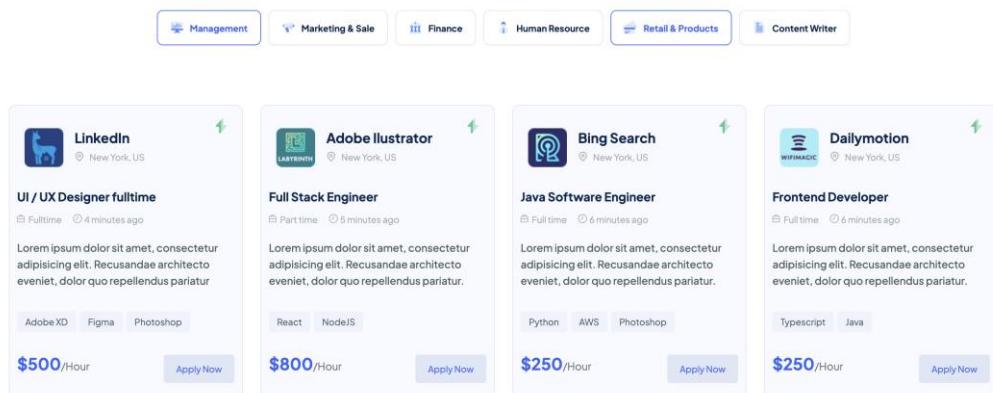
Figura 55 - Como funciona final

Secção “Ofertas mais recentes”

Na secção de ofertas mais recentes, apenas realizar uma edição simples onde comecei, como todas as outras, por alterar as cores para a paleta de cores do projeto, alterei os títulos e adicionei mais categorias e estruturei o layout das mesmas á minha maneira.

Jobs of the day

Search and connect with the right candidates faster.



Platform	Title	Type	Location	Skills	Rate	Action
LinkedIn	UI / UX Designer fulltime	Fulltime	New York, US	Adobe XD, Figma, Photoshop	\$500/Hour	Apply Now
Adobe Illustrator	Full Stack Engineer	Part-time	New York, US	React, NodeJS	\$800/Hour	Apply Now
Bing Search	Java Software Engineer	Full time	New York, US	Python, AWS, Photoshop	\$250/Hour	Apply Now
Dailymotion	Frontend Developer	Fulltime	New York, US	TypeScript, Java	\$250/Hour	Apply Now

Figura 56 - Ofertas mais recentes inicial

Ofertas mais recentes

Encontra as ofertas de trabalho mais recentes.

Desenvolvimento
 Redes
 Cibersegurança
 Dados & IA
 Suporte Técnico

Hardware & IoT
 Cloud
 Marketing
 UI & UX
 Consultoria
 E-commerce

LinkedIn New York, US

UI / UX Designer fulltime Fulltime | 4 minutes ago

Lore ipsum dolor sit amet, consectetur adipiscing elit. Recusandae architecto eveniet, dolor quo repellendus paratur.

Adobe XD | Figma | Photoshop

\$500 /Hour Apply Now

Adobe Illustrator New York, US

Full Stack Engineer Partime | 5 minutes ago

Lore ipsum dolor sit amet, consectetur adipiscing elit. Recusandae architecto eveniet, dolor quo repellendus paratur.

React | NodeJS

\$800 /Hour Apply Now

Bing Search New York, US

Java Software Engineer Fulltime | 6 minutes ago

Lore ipsum dolor sit amet, consectetur adipiscing elit. Recusandae architecto eveniet, dolor quo repellendus paratur.

Python | AWS | Photoshop

\$250 /Hour Apply Now

Dailymotion New York, US

Frontend Developer Fulltime | 6 minutes ago

Lore ipsum dolor sit amet, consectetur adipiscing elit. Recusandae architecto eveniet, dolor quo repellendus paratur.

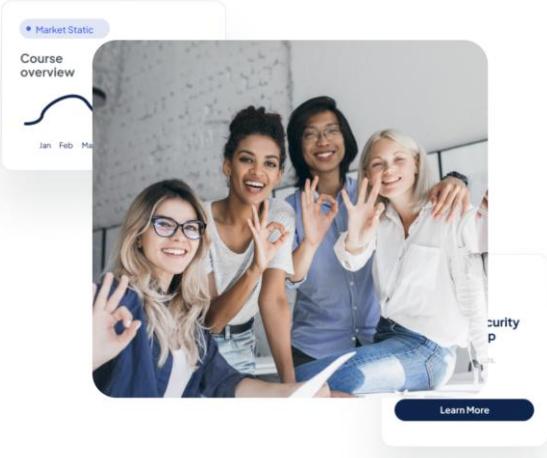
TypeScript | Java

\$250 /Hour Apply Now

Figura 57 - Ofertas mais recentes final

Secção “Procurar trabalho”

De seguida, optei por ir buscar uma secção de outra página do *template*. Esta secção conta com 3 imagens, as quais eu reduzi para apenas uma, um breve texto de promoção à plataforma e um botão que redireciona para a página de oferta de emprego.



Millions Of Jobs.
Find The One That's Right For You

Search all the open positions on the web. Get your own personalized salary estimate. Read reviews on over 600,000 companies worldwide. The right job is out there.

[Search Jobs](#) [Learn More](#)

Figura 58 - Procurar trabalho inicial



**Muita oferta.
Encontra o trabalho
Certo para ti.**

Descobre todas as oportunidades à tua espera. Explora as ofertas disponíveis. Aprende mais sobre o mundo do trabalho, desenvolve-te e conquista o teu futuro. Tudo isto, de forma simples e acessível, na **JetHire**.

 Procurar trabalho

Figura 59 - Procurar trabalho final

Secção “Candidatos recentes”

A secção “Candidatos recentes” é uma secção incompleta visto que, só poderei comecei a adicionar os diversos candidatos quando tiver a base de dados estruturada e pronta. Por isso, as únicas alterações foram as cores e os títulos.

Top Candidates

Jobs is a curated job board of the best jobs for developers, designers and marketers in the tech industry.



Robert Fox
UI/UX Designer
 (65)

[Lorem ipsum dolor sit amet consectetur adipiscing elit. Vero repellendus magni, atque delectus molestias quis?]

Figma Adobe XD PSD
App Digital

📍 Chicago, US ⏰ \$45 / hour



Cody Fisher
Python developer
 (65)

[Lorem ipsum dolor sit amet consectetur adipiscing elit. Vero repellendus magni, atque delectus molestias quis?]

Figma Adobe XD PSD
App Digital

📍 Chicago, US ⏰ \$45 / hour



Jerome Bell
Content Manager
 (65)

[Lorem ipsum dolor sit amet consectetur adipiscing elit. Vero repellendus magni, atque delectus molestias quis?]

Figma Adobe XD PSD
App Digital

📍 Chicago, US ⏰ \$45 / hour



Jane Cooper
Network
 (65)

[Lorem ipsum dolor sit amet consectetur adipiscing elit. Vero repellendus magni, atque delectus molestias quis?]

Figma Adobe XD PSD
App Digital

📍 Chicago, US ⏰ \$45 / hour

Figura 60 - Candidatos recentes inicial

Candidatos recentes

Encontra os candidatos mais recentes da plataforma.


Robert Fox
 UI/UX Designer
 (65)

[Lorem ipsum dolor sit amet consectetur adipiscing elit. Vero repellendus magni, atque delectus molestias quis?]

Figma
Adobe XD
PSD

App
Digital

📍 Chicago, US
⌚ \$45 /hour


Cody Fisher
 Python developer
 (65)

[Lorem ipsum dolor sit amet consectetur adipiscing elit. Vero repellendus magni, atque delectus molestias quis?]

Figma
Adobe XD
PSD

App
Digital

📍 Chicago, US
⌚ \$45 /hour


Jerome Bell
 Content Manager
 (65)

[Lorem ipsum dolor sit amet consectetur adipiscing elit. Vero repellendus magni, atque delectus molestias quis?]

Figma
Adobe XD
PSD

App
Digital

📍 Chicago, US
⌚ \$45 /hour


Jane Cooper
 Network
 (65)

[Lorem ipsum dolor sit amet consectetur adipiscing elit. Vero repellendus magni, atque delectus molestias quis?]

Figma
Adobe XD
PSD

App
Digital

📍 Chicago, US
⌚ \$45 /hour

[Mais](#)

Figura 61 - Candidatos recentes final

Secção “Cria o teu perfil”

Nesta secção, mais uma vez, apenas alterei as cores e o texto para o propósito da plataforma.



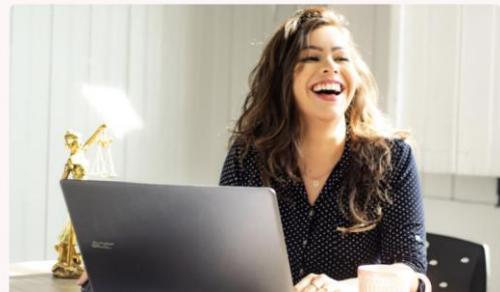
Create Profile

Create Your Personal Account Profile

Work Profile is a personality assessment that measures an individual's work personality through their workplace traits, social and emotional traits; as well as the values and aspirations that drive them forward.

[Create Profile](#)

Figura 62 - Cria o teu perfil inicial



Não percas esta oportunidade

Cria já o teu perfil

Ao partilhas as tuas habilidades, experiências e aspirações, estarás um passo mais próximo de encontrar a oportunidade de trabalho ideal para ti. Não deixes para amanhã o que podes fazer hoje!

[Criar conta](#)

Figura 63 - Cria o teu perfil final

Página de ofertas

A página de ofertas de emprego é onde o utilizador pode ver todas as ofertas e escolher as que mais se adequam a ele. Tal como as restantes páginas, está dividida em diferentes secções, desde o cabeçalho onde se encontra a barra de pesquisa com palavras-chave para as ofertas, uma barra de filtros avançados, onde o utilizador pode filtrar todas as opções a seu gosto e de acordo com as suas preferências.

Comecei por editar o cabeçalho da página que refere quantos trabalhos estão disponíveis e uma barra de pesquisa para a procura de palavras chave.

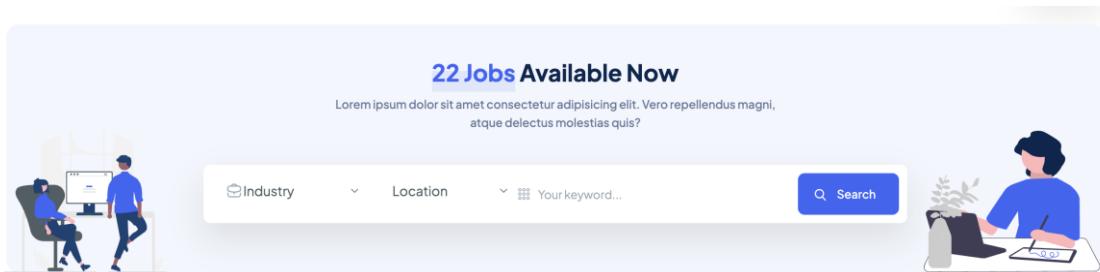


Figura 64 - Cabeçalho da página de ofertas inicial



Figura 65 - Cabeçalho da página de ofertas final

De seguida, editei o menu lateral dos filtros avançados onde alterei todas as propriedades com o intuito do projeto, mas tendo em conta a possibilidade de alterações futuramente. Aqui está o exemplo de todos os filtros que implementei:

Área

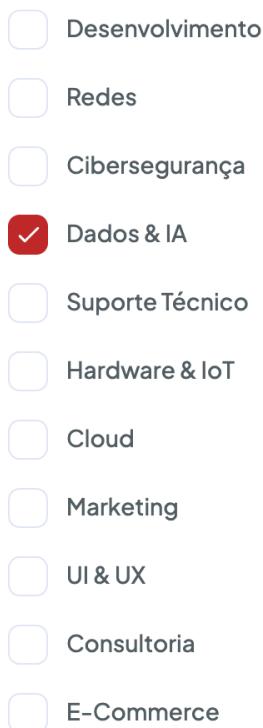


Figura 66 - Filtro avançado: Área

Salário

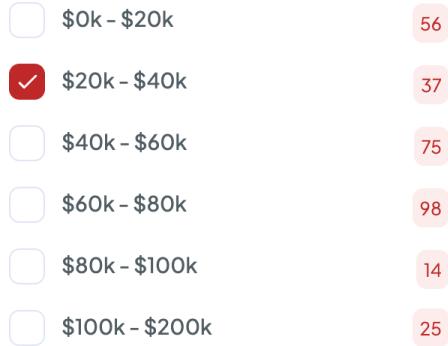


Figura 67 - Filtro avançado: Salário

Modalidade

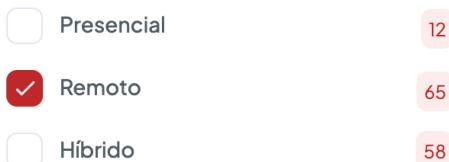


Figura 68 - Filtro avançado: Modalidade

Experiência

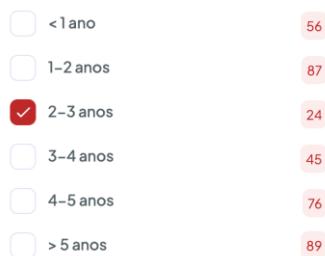


Figura 69 - Filtro avançado:
Experiência

Oferta publicada



Figura 70 - Filtro avançado:
Oferta Publicada

Tipo de trabalho



Figura 71 - Filtro avançado: Tipo
de Trabalho

Por fim, o considerado mais importante, editei a apresentação das ofertas de emprego.

Showing 41–60 of 944 jobs

Show: 12

Sort by: Newest Post


LinkedIn
(1) New York, US

Adobe XD

Figma



UI / UX Designer fulltime

(1) Fulltime

(1) 4 mins ago

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

\$500/Hour

[Apply Now](#)
Figura 72 - Oferta de emprego inicial

A mostrar 41–60 de 944 resultados

Ordenar por: Publicações recentes


LinkedIn
(1) New York, US

Adobe XD

Figma



UI / UX Designer fulltime

(1) Fulltime

(1) 4 mins ago

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

\$500/Hour

[Apply Now](#)
Figura 73 - Oferta de emprego final

Página de empresas

Na página de empresas, mantive o mesmo cabeçalho e o mesmo filtro de pesquisa, apenas editei a apresentação das ofertas de emprego para a apresentação das empresas.

Showing 41–60 of 944 jobs

Show: 12

Sort by: Newest Post


Car Toys
(1) ★★★★★ (66)

(1) New York, US

[12 Jobs Open](#)

Carols Daughter
(1) ★★★★★ (18)

(1) London, UK

[25 Jobs Open](#)

Amazon
(1) ★★★★★ (52)

(1) Tokyo, Japan

[54 Jobs Open](#)
Figura 74 - Apresentação de empresas inicial

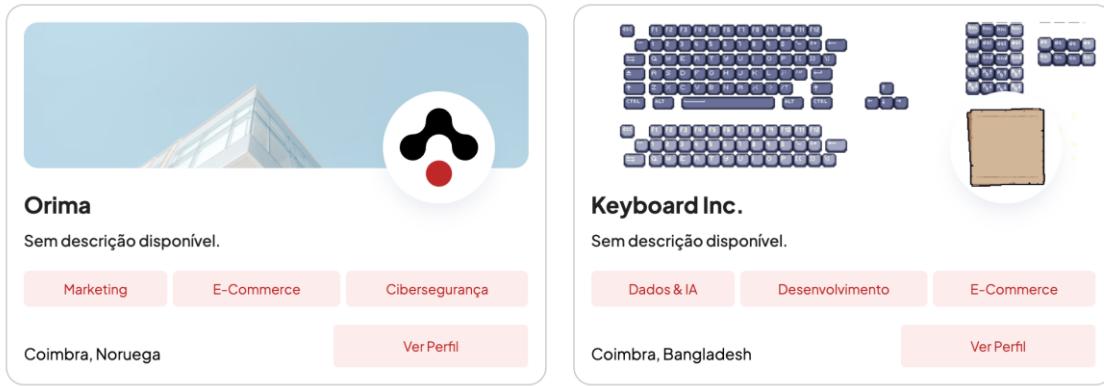
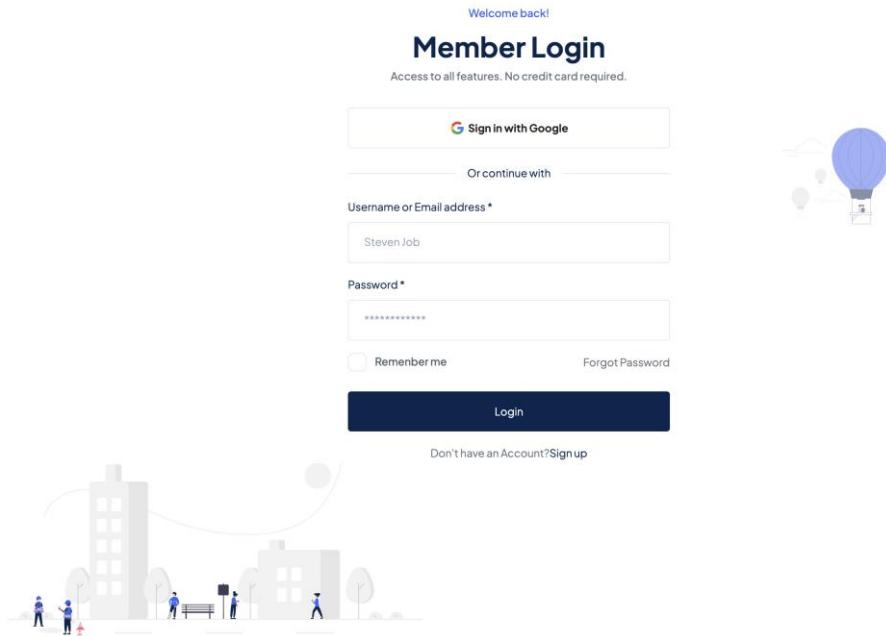


Figura 75 - Apresentação de empresas final

Página de início de sessão

Para a página de início de sessão, limitei-me a alterar as cores e alterar a disposição das outras plataformas (Google, LinkedIn, Github), onde anteriormente apenas existia a Google, agora existem 3 plataformas.



Welcome back!

Member Login

Access to all features. No credit card required.

Sign in with Google

Or continue with

Username or Email address *

Password *

Remember me Forgot Password

Login

Don't have an Account? [Sign up](#)

Figura 76 - Página de início de sessão inicial

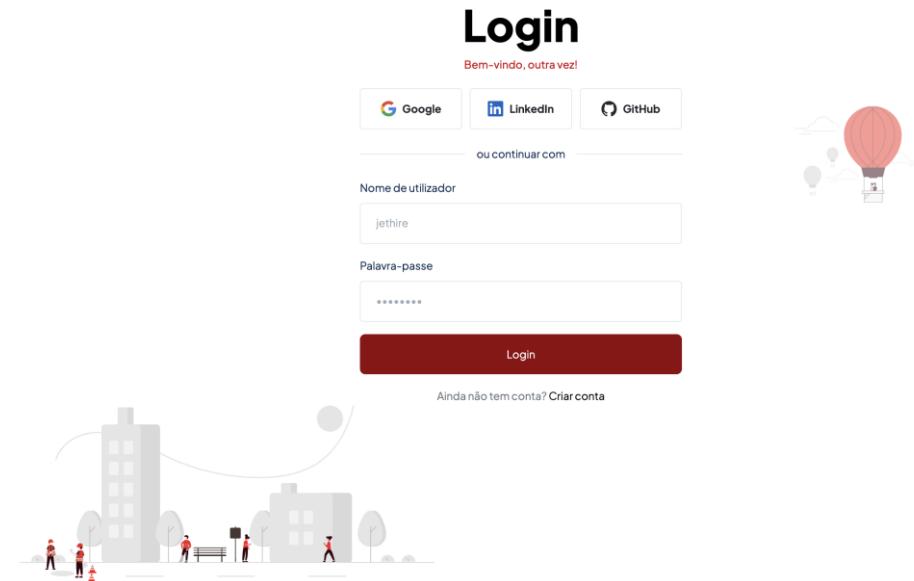


Figura 77 - Página de início de sessão final

Página de registo

Para a página de registo, mantive a mesma estrutura da página de início de sessão, apenas com as informações necessárias para o registo do utilizador.

Figura 78 - Página de registo inicial

[Registe-se](#)

Bem-Vindo!

Acesso a todas as funcionalidades.
Sem necessidade de cartão de crédito.

 Google
 LinkedIn
 GitHub

ou continuar com

Nome Completo *

Jet Hire

Email *

jethire@gmail.com

Nome de utilizador *

jethire

Palavra-passe *

Repita a palavra-passe *

Concordo com os [Termos & Condições](#)

Registrar

Já tem uma conta? [Entrar](#)

Figura 79 - Página de registo final

Página de apresentação de perfil

Na página de apresentação, decidi manter uma estrutura semelhante à do *template* dando apenas o meu toque pessoal.

Comecei por remover certas coisas do primeiro *banner* que considerei desnecessárias. Outro ponto que decidi alterar foi a parte do currículo. Quero que os utilizadores tenham uma experiência mais intuitiva e não precisem de fazer *download* de cada currículo que pretendem ver, então troquei a simbologia para tornar mais claro.

Steven Jobs New York, US
UI/UX Designer. Front end Developer

 Download CV

Figura 80 - Banner inicial

Cofinanciado pela
União Europeia

SELO DE CONFORMIDADE EQAVET
GARANTIA DA QUALIDADE
NA EDUCAÇÃO E FORMAÇÃO PROFISSIONAL

Página | 47

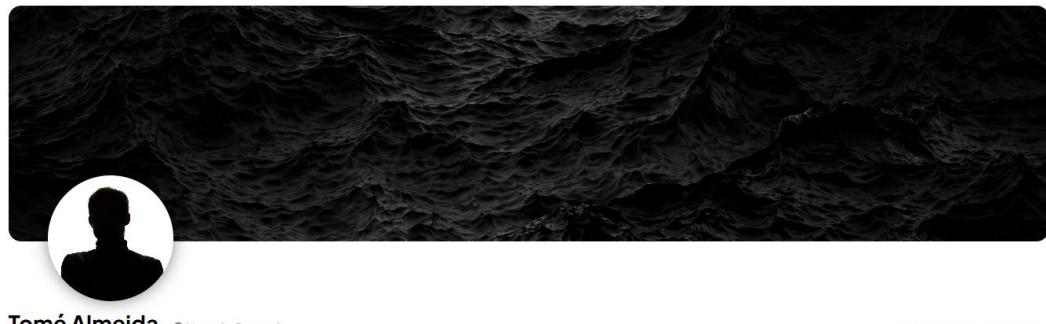


Figura 81 - Banner final

Na secção da descrição não havia muito que alterar, onde apenas alterei ligeiramente a cor do texto obtendo este resultado final:

Sobre mim

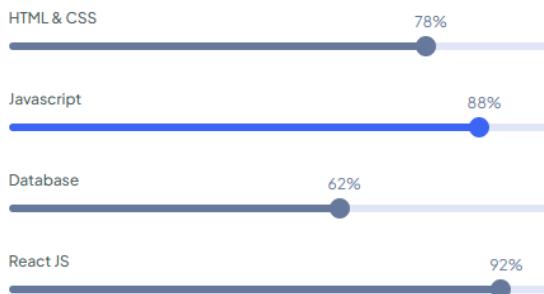
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed convallis mauris quis arcu ultrices, sed vestibulum nibh rhoncus. Curabitur pulvinar auctor nunc, a pretium justo tincidunt id. Morbi eleifend elit felis, venenatis rutrum massa rhoncus non. Pellentesque aliquet tellus consectetur odio finibus hendrerit sit amet vel sem. Duis ipsum quam, sodales ut convallis eget, aliquam ac ligula. Sed condimentum rhoncus mi ac ornare. Aliquam vel egestas eros, non consectetur risus. Donec velit nisl, maximus vel vestibulum id, suscipit in orci. Phasellus efficitur mollis nulla non semper. Nulla vestibulum, ex sed hendrerit dapibus, iacus erat vestibulum urna, eu suscipit lectus magna at est. Integer nec ipsum quis nunc lacinia finibus sit amet in lorem. In rutrum sed magna mollis consequat. Quisque neque nibh, pulvinar vitae ultricies ut, consequat at elit. Suspendisse potenti. Aenean mattis fringilla gravida.

Figura 82 - Descrição final

De seguida, a secção das competências profissionais. Nesta secção só troquei as cores para a cor principal da aplicação, o vermelho.

Professional Skills

Programming



Design

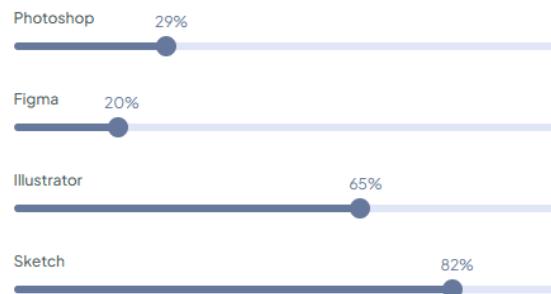


Figura 83 - Competências iniciais

Competências profissionais

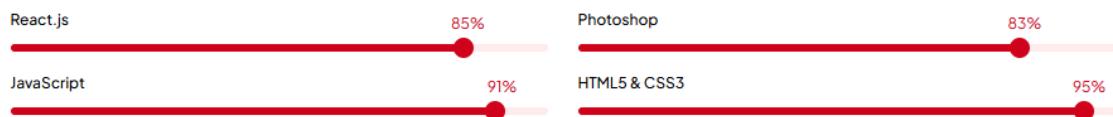


Figura 84 - Competências finais

A secção da educação e da experiência é a secção que sofreu mais alterações. No *template* apenas estava uma lista com vários itens pouco intuitivos, e esta secção é uma das que considero mais relevantes. Transformei-a assim numa lista mas de tópicos com informações do instituto, curso e data para a educação e empresa, função e data para a experiência.

Work Experience

- A portfolio demonstrating well thought through and polished end to end customer journeys
- 5+ years of industry experience in interactive design and / or visual design
- Excellent interpersonal skills
- Aware of trends in mobile, communications, and collaboration
- Ability to create highly polished design prototypes, mockups, and other communication artifacts
- The ability to scope and estimate efforts accurately and prioritize tasks and goals independently
- History of impacting shipping products with your work
- A Bachelor's Degree in Design (or related field) or equivalent professional experience
- Proficiency in a variety of design tools such as Figma, Photoshop, Illustrator, and Sketch

Figura 85 - Experiência inicial

Experiência

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

🏠 ETPC
💻 Full-Stack Developer
⌚ 2019 - 2022

Figura 86 - Experiência final

Education

- Necessitatibus quibusdam facilis
- Velit unde aliquam et voluptas reiciendis non sapiente labore
- Commodo quae ipsum quas est itaque
- Lorem ipsum dolor sit amet, consectetur adipisicing elit
- Deleniti asperiores blanditiis nihil quia officiis dolor

Figura 87 - Educação inicial

Educação

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

Figura 88 - Educação final

No fim da parte principal, decidi criar um elemento que é o **portfólio**. Uma que empresa está a ver um perfil de um candidato, é uma mais-valia ter à disposição o portfólio do mesmo onde pode ver o que este é capaz. Criei assim este elemento de forma simples com apenas uma imagem e uma legenda, e onde o clique redireciona para esse mesmo projeto.



Figura 89 - Portfólio normal

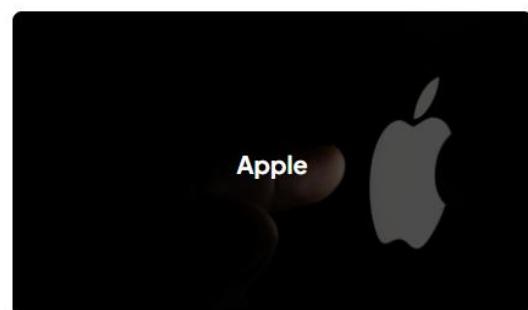


Figura 90 - Portfólio com hover

Por último, a barra lateral. A barra lateral é a mais importante, visto que, é onde qualquer outro utilizador/empresa pode ver a **visão geral** do utilizador, ou seja, toda a informação relevante do mesmo está lá. Neste elemento, mantive a mesma estrutura do *template* onde voltei a apenas dar o meu toque pessoal.

Overview

-  Experience
12 years
-  Expected Salary
\$26k - \$30k
-  Language
English, German
-  Education Level
Master Degree

- 205 North Michigan Avenue, Suite 810 Chicago, 60601, USA
- Phone: (123) 456-7890
- Email: contact@Evara.com

 **Send Message**

Visão geral

-  Experiência
Sem experiência
-  Línguas
Português, Inglês
-  Nível de educação
Curso Técnico-Profissional
-  Site pessoal
-  GitHub
-  LinkedIn

Localização: Murtede, Portugal
Telemóvel: 965360269
Email: rltomealmeida@gmail.com

 **Entrar em contacto**

Figura 91 - Barra lateral inicial
Figura 92 - Barra lateral final

Página de edição de perfil

Optei por criar esta página do zero porque nenhuma das páginas disponíveis no *template* correspondia às minhas expectativas ou necessidades específicas para o projeto. Para facilitar o desenvolvimento e garantir uma interface bem estruturada e responsiva, utilizei a biblioteca **Bootstrap**, aproveitando principalmente o seu sistema de grelhas (*grids*). Este sistema permitiu-me organizar os elementos da página de forma eficiente, assegurando assim, a sua adaptabilidade a diferentes tamanhos de ecrã.

Comecei por criar a secção inicial, onde o utilizador pode guardar os dados mais simples e diretos, sem necessidade de muitos detalhes. Nesta parte, é possível inserir informações como a cidade, o número de telemóvel e até adicionar uma imagem de perfil.

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

País

Telemóvel

Foto de perfil

Profissão atual

Email de contactos

Figura 93 - Primeira secção de edição de perfil

A segunda secção foca-se na personalização do perfil. Inclui um campo para adicionar um *banner* e duas caixas de texto, que é uma das áreas mais importantes do perfil, pois é onde o utilizador pode deixar a sua descrição pessoal.

Foto de fundo

Sobre mim

Conte-nos um pouco sobre si.

0/1800

Agora resuma em poucas palavras

Diga-nos resumidamente o que disse antes.

0/130

Figura 94 - Segunda secção de edição de perfil

Na terceira secção, as informações solicitadas são bastante simples. O utilizador deve indicar o seu nível de educação e listar as línguas que domina. Além disso, é onde este insere as suas competências profissionais, na qual indica a respetiva competência e uma percentagem de 0 a 100% onde mostra a sua experiência na mesma.

Nível de educação	Anos de experiência	Línguas
Sem escolaridade	Sem experiência	ex: Português +

Competências profissionais

ex: React.js

Arraste para o valor que pretender

React.js
79%
x

+ Adic.

Figura 95 - Terceira secção da edição de perfil

Para colocar a parte das competências funcional, implementei **handlers** para gerir a interação do utilizador. O **handleMouseDown** inicia a interação com a barra de progresso, permitindo ajustar o nível de habilidade. O **handleMouseMove** calcula o valor da competência enquanto o utilizador arrasta a barra, e o **handleMouseUp** finaliza o ajuste. Para adicionar uma nova competência, o **handleAddSkill** insere a competência na lista, enquanto o **handleRemoveSkill** permite remover competências existentes, atualizando a lista conforme necessário.

```
const [skills, setSkills] = useState([]);
const [newSkill, setNewSkill] = useState({
  name: '',
  value: 0,
});

const progressRef = useRef(null);

const handleMouseDown = () => {
  document.addEventListener('mousemove', handleMouseMove);
  document.addEventListener('mouseup', handleMouseUp);
};

const handleMouseMove = (e) => {
  const progressBar = progressRef.current;
  if (!progressBar) return;

  const rect = progressBar.getBoundingClientRect();
  let newWidth = ((e.clientX - rect.left) / rect.width) * 100;
  newWidth = Math.min(100, Math.max(0, newWidth));
  setNewSkill({ name: 'React.js', value: newWidth });
};

const handleMouseUp = () => {
  document.removeEventListener('mousemove', handleMouseMove);
  document.removeEventListener('mouseup', handleMouseUp);
};

const handleAddSkill = () => {
  setSkills([...skills, { name: 'React.js', value: 0 }]);
};

const handleRemoveSkill = (skill) => {
  const updatedSkills = skills.filter((s) => s.name !== skill.name);
  setSkills(updatedSkills);
};
```

```
newWidth = Math.max(0, Math.min(newWidth, 100));  
setNewSkill(prev => ({ ...prev, value: Math.round(newWidth) }));  
};  
  
const handleMouseUp = () => {  
  document.removeEventListener('mousemove', handleMouseMove);  
  document.removeEventListener('mouseup', handleMouseUp);  
};  
  
const handleAddSkill = () => {  
  if (newSkill.name.trim() !== "") {  
    const updatedSkills = [...skills, newSkill];  
    setSkills(updatedSkills);  
    setNewSkill({ name: "", value: 0 });  
  }  
};  
  
const handleRemoveSkill = (index) => {  
  const updatedSkills = skills.filter((_, i) => i !== index);  
  setSkills(updatedSkills);  
};
```

A quarta secção revelou-se das mais desafiantes, dado que foi necessário desenvolver um formulário específico para registar tanto a experiência profissional como a experiência académica. Esta parte exigiu maior atenção ao detalhe e uma estrutura cuidadosa para armazenar essas informações de forma eficiente.

Comecei por definir a interface que queria utilizar acabando por ser este o resultado final:

Educação

Curso	Instituto	Ínicio	Fim
Sem nenhum registo.			

Experiência

Curso	Instituto	Ínicio	Fim
Sem nenhum registo.			

Função	Empresa	Ínicio	Fim	Adicionar
ex: Desenvolvedor Front-End	ex: Jet Hire	Selec. ▾	Selec. ▾	+ Adic.

Figura 96 - Quarta secção do registo de perfil

Para começar, é necessário criar um formulário para o utilizador ser capaz de inserir os seus dados. O seguinte código mostra o formulário adaptado para a estrutura anteriormente apresentada:

```
<div className="row">
  {/* Campo para o curso */}
  <div className="col-lg-6">
    <h6 className="mb-10">Curso</h6>
    <div className="input-style mb-20">
      <input
        className="font-sm color-text-paragraph-2"
        name="role" // Nome do campo
        value={newEducation.role} // Valor atual
        placeholder="ex: Engenheiro Informático" // Placeholder
        onChange={handleInputChange} // Função ao alterar o campo
        type="text"
      />
    </div>
  </div>

  {/* Campo para o instituto */}
  <div className="col-lg-3">
    <h6 className="mb-10">Instituto</h6>
    <div className="input-style mb-20">
      <input

```

```

        className="font-sm color-text-paragraph-2"
        name="company" // Nome do campo
        value={newEducation.company} // Valor atual
        placeholder="ex: ISEC" // Placeholder
        onChange={handleInputChange} // Função ao alterar o campo
        type="text"

    />

```

```

</div>

```

```

</div>

/* Seletor para o ano de início */
<div className="col-lg-1">
    <h6 className="mb-10">Ínicio</h6>
    <div className="select-style mb-20">
        <select
            name="startYear" // Nome do campo
            value={newEducation.startYear} // Valor atual
            onChange={handleInputChange} // Função ao alterar
        >
            <option>Selec.</option>
            {years.map((year) => (
                <option key={year} value={year}>
                    {year}
                </option>
            )))
        </select>
    </div>

```

```

</div>

/* Seletor para o ano de fim */
<div className="col-lg-1">
    <h6 className="mb-10">Fim</h6>
    <div className="select-style mb-20">

```

```

<select
    name="endYear" // Nome do campo
    value={newEducation.endYear} // Valor atual
    onChange={handleInputChange} // Função ao alterar
>
    <option>Selec.</option>
    <option value="Atual">Atual</option>
    {years.map((year) => (
        <option key={year} value={year}>
            {year}
        </option>
    )))
</select>
</div>
</div>

/* Botão para adicionar uma nova entrada */
<div className="col-lg-1 mb-20">
    <h6 className="mb-10">Adicionar</h6>
    <div className="mb-20">
        <button className="btn-submit-register-form" onClick={addEducation}>
            <i className="fa-solid fa-plus mr-5"></i> Adic.
        </button>
    </div>
</div>
</div>

```

Agora com o formulário criado é preciso armazenar os dados lá inseridos. Para isso, utilizei o hook **useState**, para armazenar os dados como um *array*. O seguinte bloco de código está comentado com a respetiva explicação:

```

export default function RegistProfile() {
    // Estado para armazenar as experiências adicionadas

```

```

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

// Estado para armazenar os dados da nova experiência que está a ser preenchida
const [newEducation, setNewEducation] = useState({
    role: "",      // Cargo ou função desempenhada
    company: "",   // Nome da instituição ou empresa
    startYear: "", // Ano de início
    endYear: "" ,  // Ano de fim (ou "Atual")
});

// Função para lidar com as alterações nos campos de entrada do formulário
const handleInputChange = (e) => {
    const { name, value } = e.target; // Obtém o nome e o valor do campo
    setNewEducation({ ...newEducation, [name]: value }); // Atualiza o estado da nova
    experiência
};

// Função para adicionar uma nova experiência
const addEducation = (e) => {
    e.preventDefault(); // Impede o recarregamento da página ao submeter o formulário
    const { role, company, startYear, endYear } = newEducation;

    // Verifica se todos os campos estão preenchidos
    if (!role || !company || !startYear || (!endYear && endYear !== "Atual")) {
        alert("Preencha todos os campos.");
        return;
    }

    // Verifica se o ano de início é anterior ao ano de fim (exceto se for "Atual")
    if (startYear !== "Atual" && endYear !== "Atual" && parseInt(startYear) >
    parseInt(endYear)) {
        alert("A data de início deve ser anterior à data de fim.");
        return;
    }
}

```

```

// Adiciona a nova experiência ao array de experiências existentes
setEducations([...educations, newEducation]);

// Limpa os campos do formulário
setNewEducation({ role: "", company: "", startYear: "", endYear: "" });

};

// Função para remover uma experiência com base no índice
const removeEducation = (index) => {
    // Filtra a experiência que corresponde ao índice fornecido
    setEducations(educations.filter((_, i) => i !== index));
};

}

```

Após ter o formulário funcional e um `array` guardar os dados, só falta apresentar os mesmos. Para esse efeito, apenas adicionei o seguinte código onde anteriormente apenas tinha a frase “Sem nenhum registo.”:

```

{educations.length === 0 ? (
    // Caso não existam regtos
    <div className="mb-20">Sem nenhum regsto.</div>
) : (
    // Mapeia as experiências educacionais
    educations.map((edu, index) => (
        <div className="row mb-10" key={index}>
            {/* Mostra o curso */}
            <div className="col-lg-6">
                <div className="input-style">{edu.role}</div>
            </div>

            {/* Mostra o instituto */}
            <div className="col-lg-3">
                <div className="input-style">{edu.company}</div>

```

```

</div>

/* Mostra o ano de início */
<div className="col-lg-1">
    <div>{edu.startYear}</div>
</div>

/* Mostra o ano de fim */
<div className="col-lg-1">
    <div>{edu.endYear}</div>
</div>

/* Botão para remover o registo */
<div className="col-lg-1">
    <button
        className="remove-button-profile"
        onClick={() => removeEducation(index)}
        type="button"
    >
        <strong>Remover</strong>
    </button>
</div>
</div>
))

}

```

Após este processo, é possível adicionar todos os dados pretendidos pelo utilizar e a respetiva eliminação dos mesmos sempre que pretendido, obtendo este resultado:

Educação

Curso

Técnico de Gestão de Programação de Sistemas Informáticos

Curso

ex: Engenheiro Informático

Instituto

ETPC

Instituto

ISEC

Ínicio

2022

Fim

Atual

Ínicio

Ínicio

Fim

Fim

Remover

Adicionar

Selec. ▾

Selec. ▾

+ Adic.

Figura 97 - Exemplo de adição de dados no registo de perfil

Agora a secção do portfólio. Esta secção é muito simples e intuitiva visualmente onde apenas pede o campo da legenda, o *link* e a imagem que em questão. Quando o utilizador preencher os três campos e clicar no botão “Adicionar” irá aparecer o elemento semelhante a como aparece no perfil do mesmo.

Portfolio

Legenda	Link	Imagen	Adicionar
Waves	waves.pt		+ Adic.

Figura 98 - Portfólio na edição de perfil

Portfolio

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

waves.pt



Figura 99 - Exemplo de portfólio na edição de perfil

Por fim, na última secção, o utilizador pode inserir os links para o seu site pessoal e para outras duas plataformas online, como LinkedIn e GitHub. Além disso, é possível adicionar uma imagem ou carregar um ficheiro em formato PDF com o currículo.

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 100 - Quinta secção do registo de perfil

Segurança

Criação de um ambiente seguro

Num projeto Next.js, o ficheiro `.env` é utilizado para guardar informações sensíveis, como `passwords`, passes de API ou URLs de bases de dados, de forma segura e organizada. Este ficheiro cria um ambiente onde essas variáveis ficam disponíveis apenas dentro do projeto, protegendo-as de serem expostas no código ou na internet.

Por padrão, o ficheiro `.env` é incluído no `.gitignore`, o que significa que ele não é enviado para repositórios públicos ou partilhado com outras pessoas, garantindo que os dados confidenciais fiquem seguros. No contexto de um projeto Next.js, este ficheiro permite definir variáveis de ambiente que podem ser acedidas com `process.env`, tal como apresentado na linha seguinte:

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

Assim, o ficheiro `.env` é uma ferramenta essencial para proteger e gerir informações sensíveis de forma prática e eficaz.

MongoDB

No meu projeto, escolhi usar a base de dados MongoDB, que é uma base de dados não-relacional. Tomei esta decisão porque queria aprender algo novo e diferente das bases de dados relacionais, que são as únicas com que trabalhei até agora. O MongoDB chamou a minha atenção por ser uma ferramenta moderna e bastante utilizada em aplicações web. Ele organiza os dados de uma forma mais flexível, o que facilita o trabalho com informações que mudam ou crescem rapidamente. Com esta experiência, espero entender melhor as diferenças entre bases de dados relacionais e não-relacionais, e saber quando usar cada tipo. Além disso, ajuda-me a desenvolver competências para criar aplicações mais dinâmicas e versáteis.

Criação do cluster

Para começar a utilizar MongoDB, é preciso criar um *cluster*.

Um cluster no MongoDB é como um grupo de computadores que trabalham juntos para guardar e gerir os dados. Ele divide os dados em partes menores e distribui essas partes pelos computadores do grupo. Assim, o cluster consegue guardar muitos dados e continuar rápido, mesmo quando várias pessoas o usam ao mesmo tempo. Além disso, se um computador falhar, os outros continuam a funcionar, mantendo os dados seguros e disponíveis.

Para dar início à criação do mesmo, temos que criar conta na plataforma até chegar à página inicial.

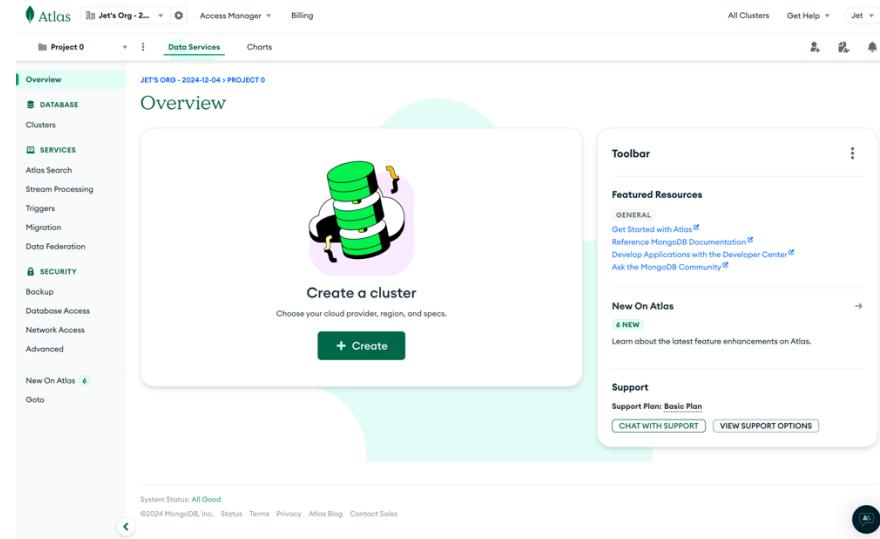


Figura 101 - Página inicial MongoDB

Após a criação de conta, passamos á próxima fase, a criação do *cluster*. Para criar um *cluster* temos que primeiro, criar um projeto, como o exemplo na imagem seguinte:

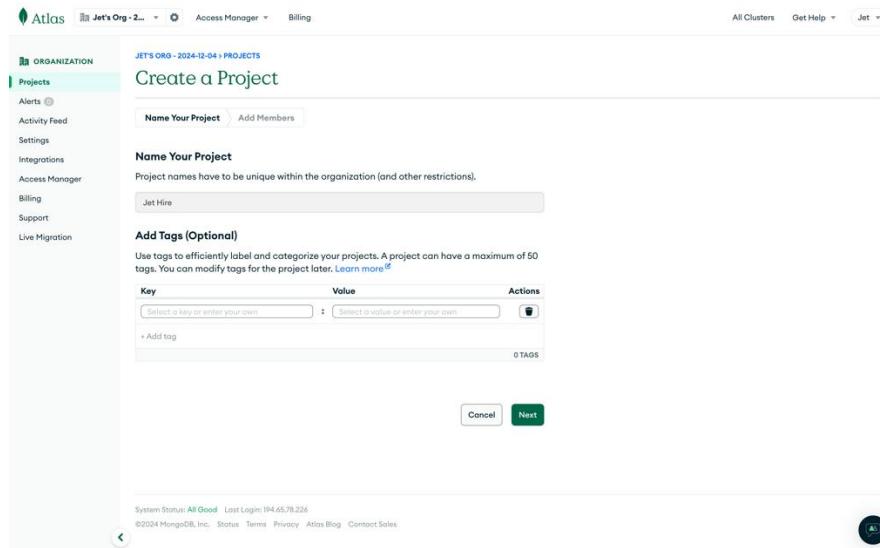


Figura 102 - Criação de projeto no MongoDB

De seguida, sim, podemos criar o *cluster*. Apenas é necessário definir o plano que pretendemos utilizar e a fornecedora do serviço. Neste caso, foi utilizado um plano grátis e a fornecedora escolhida foi a AWS.

Deploy your cluster

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

M10 **\$0.09/hour**
 Dedicated cluster for development environments and low-traffic applications.

STORAGE	RAM	vCPU
10 GB	2 GB	2 vCPUs

Flex **From \$0.011/hour**
 Up to \$30/month
 For application development and testing, with on-demand burst capacity for unpredictable traffic.

STORAGE	RAM, vCPU	OPS/SEC
5 GB	Shared	0 - 500

Free
 For learning and exploring MongoDB in a cloud environment.

STORAGE	RAM, vCPU	OPS/SEC
512 MB	Shared	0 - 100

Free forever! Your free cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

Configurations

Name: Cluster0

Provider: **AWS** (selected)

Region: Paris (eu-west-3) ★ (Low carbon emissions)

Quick setup

Automate security setup

Preload sample dataset

I'll do this later
Go to Advanced Configuration
Create Deployment
\$10

Figura 103 - Criação de um cluster no MongoDB

Agora o *cluster* está criado e pronto a ser utilizado no projeto.

Acesso á base de dados

Para nos conectarmos á base de dados, é necessária uma função responsável para esse mesmo fim. Para esta funcionar de maneira correta, é preciso um URL específico dado pela base de dados como forma de ligação como administrador.

URL do MongoDB

Para obter este URL, é necessário seguir certos passos, começando pela criação de um administrador da base de dados.

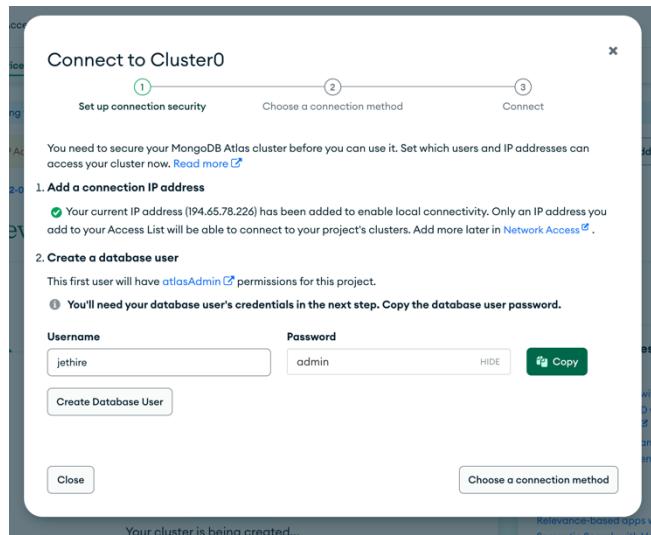


Figura 104 - Criação de administrador no MongoDB

Depois da criação do administrador escolhemos a plataforma em que pretendemos aceder á base de dados.

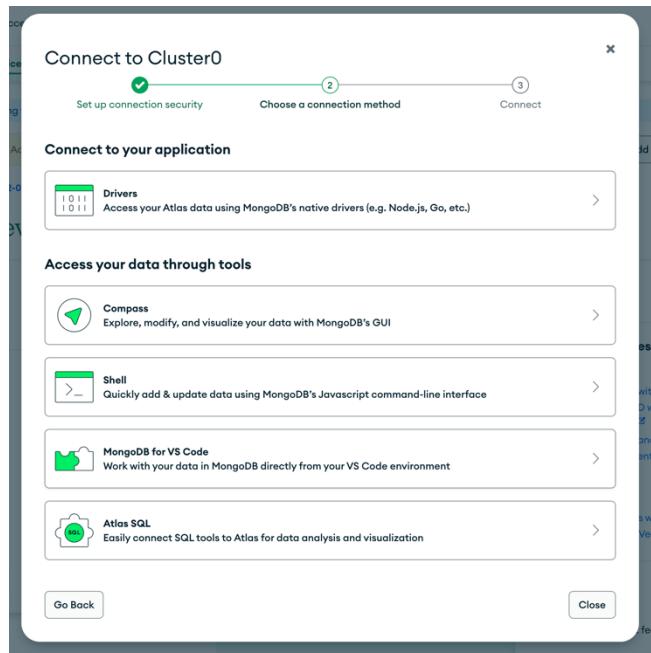


Figura 105 - Escolha de aplicação no MongoDB

Neste exemplo, vou escolher uma aplicação aleatória, para obter o URL.

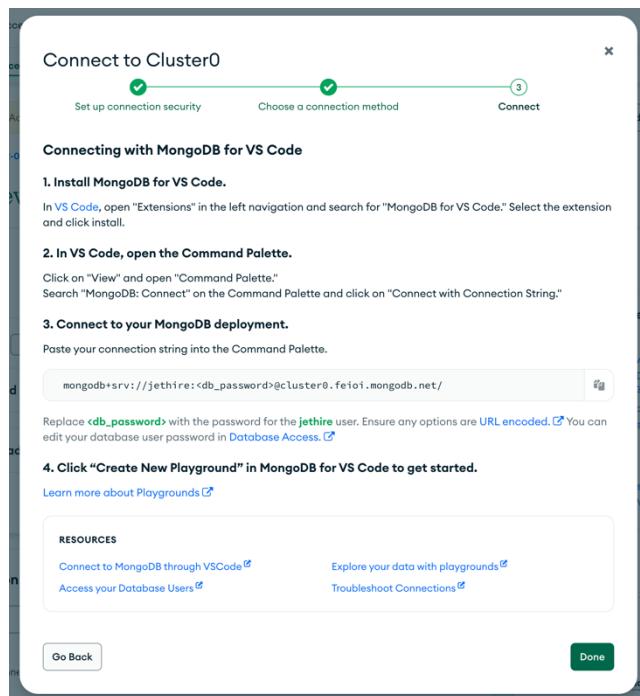


Figura 106 - Ligação ao cluster no MongoDB

Nesta janela já é possível observar o URL para a conexão com a base de dados. Outro fator importante é que é necessário alterar a palavra-passe do URL para a palavra-passe definida para o administrador.

```
mongodb+srv://jethire:<db_password>@cluster0.feioi.mongodb.net/jet_hire_db
```

Visual Studio Code

Para nos conectarmos pelo Visual Studio Code, é preciso utilizar o ficheiro anteriormente criado (**.env**), e inserir o URL dado pela plataforma da seguinte maneira:

```
MONGODB_URL=mongodb+srv://jethire:admin@cluster0.feioi.mongodb.net/jet_hire_db
```

MongoDB Compass

Outra forma de pré-visualizarmos os dados e até mesmo ser possível a edição dos mesmos podemos utilizar a aplicação própria da plataforma (MongoDB Compass), onde apenas é necessário inserir o tal URL na caixa de texto “Nova Conexão”:

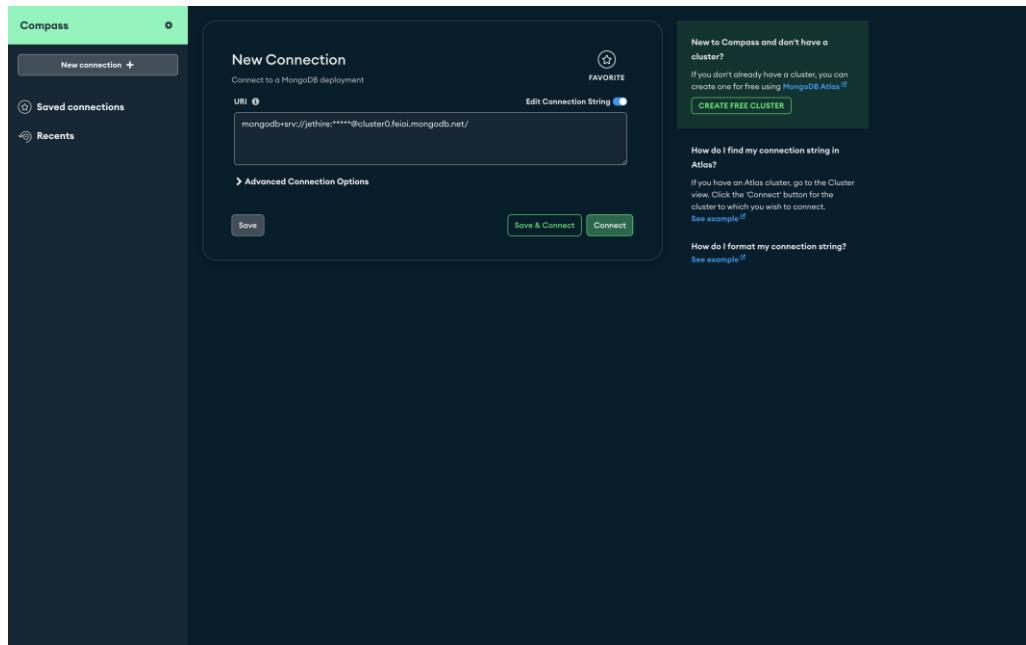


Figura 107 - Página inicial MongoDB Compass

Após realizar a conexão, toda a base de dados está disponível e vem com uma interface (GUI) mais simples e intuitiva, de forma a facilitar a edição de dados.

```

_id: ObjectId('67580ab01cd44b65ca4e1340')
email: "rltomeiedel@gmail.com"
name: "Rui Tomé"
createdat: 2024-12-04T12:41:17.857+00:00
updatedat: 2024-12-04T12:41:17.857+00:00
__v: 0

_id: ObjectId('67582ab01cd44b65ca4e1340')
email: "jethire.pt@gmail.com"
name: "Jet Hire"
createdat: 2024-12-10T11:49:04.629+00:00
updatedat: 2024-12-10T11:49:04.629+00:00
__v: 0

_id: ObjectId('67582bb41cd44b65ca4e1342')
email: "teste@presentacao.com"
name: "Teste Apresentação"
createdat: 2024-12-10T11:53:24.323+00:00
updatedat: 2024-12-10T11:53:24.323+00:00
__v: 0

```

Figura 108 - Interface do MongoDB Compass

Mongoose

O processo anterior mostra como ter acesso à base de dados sendo administrador. Para nos conectarmos realmente, é necessária uma função com a utilização do respetivo URL dado anteriormente. Para isto, optei por utilizar a biblioteca **Mongoose**.

Instalação do Mongoose

Para instalar o Mongoose apenas é necessário utilizar o seguinte comando no terminal:

```
npm i mongoose
```

Se a instalação for bem sucedida, as seguintes pastas irão aparecer dentro da pasta *node_modules*.

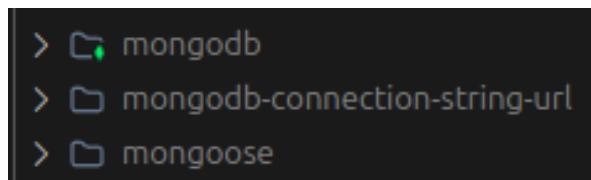


Figura 109 - Dependências Mongoose

Ligaçāo á base de dados com Mongoose

Para a conexāo á base de dados é preciso criar um novo ficheiro **mongodb.js** dentro da pasta **/lib**. Caso esta pasta ainda não exista, realiza-se a criação da mesma. Neste ficheiro apenas é necessário inserir 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);
  }
};
```

Este bloco de código define uma função chamada `connectMongoDB` que estabelece uma ligação com a base de dados.

Ele tenta conectar-se ao URL da base de dados guardado na variável de ambiente MONGODB_URL inserida no ficheiro **.env**. Se a conexão for bem sucedida, exibe “Conectado a MONGODB” no terminal. Caso ocorra algum erro, mostra “Erro a conectar” seguido da descrição do erro.

Schemas

Esta ferramenta é uma das mais populares para interagir com bases de dados MongoDB no ambiente Node.js, oferecendo uma forma mais estruturada e organizada de gerir os dados através da definição de **schemas**.

No Mongoose, os **schemas** funcionam como uma *blueprint* para os documentos dentro de uma coleção MongoDB. Com eles, é possível definir a estrutura dos dados, incluindo os campos que cada documento deverá ter, os seus tipos de dados, valores padrão, validações, e até mesmo métodos ou *middlewares*. Isto garante consistência e reduz erros ao interagir com a base de dados.

Aqui está um exemplo da criação de um **schema** para a coleção **user**.

```
import mongoose, { Schema } from "mongoose";
const userSchema = new Schema(
  {
    type: { type: String, required: true },
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
  },
  { timestamps: true }
);
export const User = mongoose.models.User || mongoose.model("User", userSchema);
```

Este bloco de código começa por importar a biblioteca Mongoose e os respetivos **schemas** da mesma. De seguida, utiliza-se a função “new Schema”. Dentro da mesma define-se os campos que pretendemos e indicamos os respetivos tipos, se são obrigatórios e como no caso do email, se têm que ser únicos.

Existem diversas propriedades além destas, disponíveis na documentação da biblioteca. No fim da definição dos campos, adicionei os *timestamps*. Os *timestamps* são dois campos pré-definidos (*createdAt*, *updatedAt*) que são responsáveis por armazenar a data e hora que os respetivos dados são adicionados e editados.

Por fim, exporta-se a função para se puder importar outros ficheiros e ser possível a utilização da mesma.

Subschemas

Os **subschemas** são uma forma de estruturar e organizar os dados dentro de um modelo principal, criando componentes reutilizáveis que representam entidades independentes, mas que ainda fazem parte de um documento maior.

Para utilizar esta estrutura, cria-se os *subschemas* com a função Schema e depois integra-se estes *subschemas* no esquema principal. Quando se definir uma propriedade como um array de *subschemas*, está-se a dizer ao Mongoose que a propriedade pode conter múltiplos documentos que seguem a estrutura do *subschema* associado.

Optei por utilizar esta funcionalidade, porque decidi adicionar o campo “perfil” dentro da coleção dos utilizadores para desta maneira ter algo mais organizado e fácil de me orientar. Quando me apercebi, que iriam ter campos iguais para dois parâmetros diferentes optei por explorar esta forma de organização.

No exemplo apresentado, temos dois *subschemas*: o *workExperienceSchema*, que define os campos relacionados à experiência profissional do utilizador, como o nome da instituição (*institute*), o cargo (*role*) e as datas de início e fim (*start* e *end*); e o *educationSchema*, que representa os dados académicos do utilizador, incluindo a instituição frequentada (*institute*), o curso (*course*) e as respetivas datas de início e fim.

```
const workExperienceSchema = new Schema({  
    institute: { type: String, required: true },  
    role: { type: String, required: true },  
    start: { type: String, required: true },  
    end: { type: String, required: false },  
});
```

```
const educationSchema = new Schema({  
    institute: { type: String, required: true },
```

```
course: { type: String, required: true },
start: { type: String, required: true },
end: { type: String, required: false },
});

const userSchema = new Schema(
{
  profile: {
    workExperience: [workExperienceSchema],
    education: [educationSchema],
  },
},
);


```

Criação de um modelo Mongoose

Um **modelo Mongoose** é a ferramenta que permite à aplicação **usar** um esquema previamente definido para **interagir com a base de dados**. Ou seja, enquanto o **schema** define a estrutura dos dados (como os campos e tipos), o **modelo** transforma esse schema num objeto que pode ser usado para **criar, ler, atualizar ou eliminar (CRUD)** documentos numa coleção MongoDB.

É através do modelo que conseguimos, por exemplo, adicionar um novo utilizador à base de dados ou procurar todos os registos existentes. O modelo representa uma **coleção concreta** da base de dados e segue sempre a estrutura definida no schema correspondente.

Normalmente, cada modelo é guardado num ficheiro próprio, o que ajuda a manter o projeto organizado. De seguida está o modelo que utilizei para os utilizadores.

/models/user.js

```
const userSchema = new Schema(
{
  type: { type: String, required: true },
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
```

```
password: { type: String },  
profile: {  
    city: String,  
    country: String,  
    aboutMe: String,  
    aboutMeShort: String,  
    yearsExperience: String,  
    role: String,  
    languages: [String],  
    educationLevel: String,  
    phone: String,  
    contactEmail: String,  
    site: String,  
    github: String,  
    linkedin: String,  
    workExperience: [workExperienceSchema],  
    education: [educationSchema],  
    skills: [skillsSchema],  
    portfolio: [portfolioSchema],  
    pfp: String,  
    pfp_id: String,  
    banner: String,  
    banner_id: String,  
    cv: String,  
    cv_id: String  
},  
},  
{ timestamps: true }  
);  
  
const User = mongoose.models.User || mongoose.model("User", userSchema);  
export default User;
```

Envio de dados para a base de dados

Para que uma aplicação consiga enviar dados para uma base de dados **MongoDB**, utilizando **Mongoose**, é necessário seguir três passos fundamentais e interligados.

- **Ligaçāo á base de dados;**
- **Criação de um modelo;**
- **Front-end:** responsável por enviar os dados para a API;
- **API:** responsável por enviar os dados para a base de dados.

Os dois primeiros passos estão explicados anteriormente. De seguida está a explicação dos restantes passos:

Front-end

No **front-end**, os **dados** inseridos pelo utilizador (geralmente através de um **formulário**) são recolhidos com **useState** e enviados para uma **rota de API** interna da aplicação **Next.js**. A submissão é feita com a função **fetch**, utilizando o método **POST**, com cabeçalho **Content-Type: application/json** e o corpo convertido para **JSON**. Por exemplo, ao submeter um nome e um e-mail, os dados são enviados para a rota **/api/users/add**. A resposta da API pode incluir mensagens de **sucesso** ou **erro**.

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

API

A **API** atua como **ponte** entre o front-end e a base de dados. Dentro da pasta **pages/api**, a rota correspondente importa a função de ligação **dbConnect** e o modelo **Mongoose** adequado (ex.: **User**). Após garantir a **ligação à base de dados**, os dados recebidos em **req.body** são usados para criar uma nova instânciā do modelo, que é guardada com **save()**.

```
import dbConnect from '@/lib/mongodb';  
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 });  
    }  
}
```

Desenvolvimento funcional da aplicação

Gestão de formulários: useState, handleChange e handleSubmit

Este tópico é abordado nesta fase do relatório porque estas funções são **utilizadas frequentemente ao longo do projeto**, especialmente na criação e edição de perfis, publicações, e outras interações com o utilizador. Compreender como funcionam permite perceber melhor o fluxo de dados e a estrutura lógica da aplicação.

Em React, é habitual recorrer-se ao **hook useState** em conjunto com as funções **handleChange** e **handleSubmit** para construir e gerir formulários de forma eficiente. Estes três elementos trabalham em conjunto para garantir uma **gestão controlada dos dados inseridos** pelo utilizador.

O **useState** é utilizado para armazenar o estado do formulário, normalmente através de um objeto chamado formData, que contém todos os campos como propriedades. Isto permite manter os dados organizados num só lugar, facilitando o acesso, a validação e a submissão. Por exemplo:

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

Com este estado, o formulário começa com os campos todos vazios, e à medida que o utilizador escreve, os dados vão sendo atualizados automaticamente.

A função **handleChange** serve para **atualizar o estado do formulário** sempre que um campo é alterado. Utiliza **e.target.name** e **e.target.value** para saber qual campo foi modificado e aplicar essa alteração ao **formData**. Esta abordagem permite criar **inputs controlados**, onde o valor apresentado no input depende do estado da aplicação. Um exemplo típico:

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

Esta função é extremamente útil para formulários dinâmicos, tornando fácil a manutenção e a adição de novos campos sem necessidade de criar funções específicas para cada um.

A função **handleSubmit** é usada quando o formulário é submetido. Serve para **impedir o comportamento padrão do navegador** (que recarregaria a página) e para processar os dados recolhidos. Normalmente, nesta função, os dados do **formData** são validados e enviados para uma API ou base de dados. Por exemplo:

```
const handleSubmit = (e) => {
  e.preventDefault();
  console.log(formData); // Envio de dados ou outra lógica
};
```

Esta estrutura dá controlo total sobre o que acontece após a submissão dos dados, permitindo executar ações específicas, como mostrar mensagens de sucesso ou erro, limpar o formulário ou redirecionar o utilizador.

Um exemplo de formulário simples que utiliza os três elementos acima descritos seria:

```
<form onSubmit={handleSubmit}>
  <input
    type="email"
    name="email"
    value={formData.email}
    onChange={handleChange}
    placeholder="Email"
  />
```

```
<textarea  
    name="mensagem"  
    value={formData.mensagem}  
    onChange={handleChange}  
    placeholder="Escreve a tua mensagem"  
/>  
<button type="submit">Enviar</button>  
</form>
```

Registo de utilizadores

A primeira interação que tive com a base de dados foi no **registo de utilizadores**. Depois de personalizar o visual da página, criei um **formulário** com os campos essenciais para o registo: **nome**, **email** e **palavra-passe** e **repetição da palavra-passe**. Para garantir a coerência dos dados, incluí uma **validação simples** para verificar se as palavras-passe coincidem antes de os enviar.

Quando o formulário é **submetido**, os dados são enviados para uma **API interna** da aplicação, que trata do registo. Nessa API, verifiquei se já existia um utilizador com o mesmo **email** e, se não existisse, a **palavra-passe** era encriptada e os dados guardados na **base de dados**.

Se tudo correr bem, o utilizador recebe uma **confirmação de sucesso** e é redirecionado para a página de **login**. Caso contrário, são apresentadas **mensagens de erro**, como por exemplo se o **email** já estiver a ser usado ou se faltar algum campo.

Este processo serviu como base para entender como ligar o **front-end à base de dados** de forma **segura e funcional**.

DÚVIDA: NESTAS SITUAÇÕES COLOCO CÓDIGO? TENHO FICHEIROS DE 300 LINHAS.

As palavras-passe não coincidem.

Nome Completo *

Email *

Palavra-passe *

Repita a palavra-passe *

Figura 110 - Exemplo de erro no registo

Início de sessão

O método que optei por escolher para o início de sessões no meu projeto foi a biblioteca **Next-Auth.js**. Esta escolha veio pela facilidade de integração desta biblioteca num projeto Next.js e simplicidade da sintaxe da mesma.

Para começar, é preciso instalar a biblioteca no projeto. Para isso utiliza-se o seguinte comando no terminal:

```
npm install next-auth
```

Depois da instalação das dependências, o próximo passo é adicionar uma *API route*, ou seja, um ficheiro responsável por criar uma rota dinâmica para o API, e que ainda contém as configurações globais do Next-Auth. Para isto é preciso criar um novo ficheiro **[...nextauth].js** dentro das seguintes pastas: **pages/api/auth**.

Após a criação ficheiro, o mesmo deve conter a seguinte estrutura:

```
import NextAuth from "next-auth"
export const authOptions = {
  providers: [
    //Adicionar aqui os diferentes providers
  ],
}
export default NextAuth(authOptions);
```

React Hook – useSession()

O hook useSession() no Next.js é a maneira mais fácil de verificar se alguém está com uma sessão ativa. Este hook pode ser utilizado em qualquer lugar do projeto.

De seguida está um exemplo da utilização deste mesmo hook.

```
import { useSession } from "next-auth/react"
export default function Component() {
  const { data: session } = useSession()
  if (session) {
    return (
      //Caso exista alguma sessão iniciada, este bloco ativa.
    )
  }
}
```

Sessão partilhada

Para ser possível utilizar o hook anterior, é preciso configurar o SessionProvider.

O SessionProvider funciona como uma “caixa” que partilha as informações da sessão com toda a aplicação. Assim, qualquer componente pode aceder aos dados da sessão e saber se o utilizador está autenticado ou não através da variável *status*.

```
import { useSession } from 'next-auth/react';
const {status, data: session} = useSession();
if (status === 'authenticated') {
  //Sessão iniciada
} else {
  //Sessão por iniciar
}
```

Além disso, o SessionProvider mantém a sessão atualizada e sincronizada entre diferentes separadores ou janelas do navegador. Ou seja, as mudanças na sessão são refletidas em toda a aplicação automaticamente.

Para utilizar o SessionProvider é preciso englobar todos os elementos do projeto através da página *_app.js*:

```
import { SessionProvider } from "next-auth/react"
export default function App({ Component, pageProps: { session, ...pageProps } }) {
```

```
return (  
  <SessionProvider session={session}>  
    <Component {...pageProps} />  
  </SessionProvider>  
)  
}
```

signIn & signOut

Para permitir que os utilizadores façam login e logout na aplicação, podemos utilizar as funções signIn e signOut fornecidas pelo NextAuth. A função signIn é utilizada para autenticar o utilizador, enquanto que a função signOut serve para terminar a sessão do utilizador.

```
import { signIn } from "next-auth/react";  
  
function LoginButton() {  
  return (  
    <button onClick={() => signIn("google")}>Login com Google</button>  
  );  
}
```

Neste bloco de código é apresentada a função *signIn*, onde é obrigatório inserir o *Provider* que pretendemos iniciar a sessão. Neste exemplo, o *Provider* definido foi o Google.

```
import { signOut } from "next-auth/react";  
  
function LogoutButton() {  
  return (  
    <button onClick={() => signOut()}>Logout</button>  
  );  
}
```

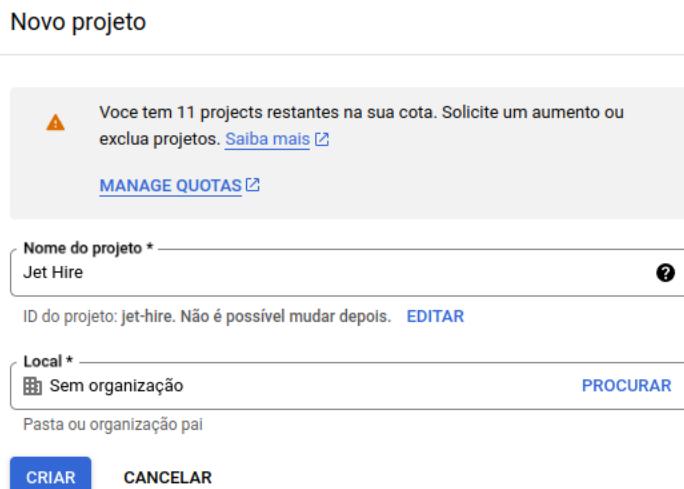
Ao contrário da função `signIn`, a função `signOut` não precisa de qualquer *Provider*. Quando esta é chamada, ele termina as sessões de qualquer *Provider*.

Google Cloud

Para utilizar o Google como *provider* de autenticação, é necessário configurar o Google Cloud Console. O processo envolve criar um projeto, configurar as credenciais e adicionar a chave do cliente ao arquivo de configuração do NextAuth.

Criação do projeto no Google Cloud Console

Ao entrar na Google Cloud Console, a própria plataforma faz-nos criar um projeto através do seguinte formulário:



The screenshot shows the 'Novo projeto' (New project) page in the Google Cloud Console. It includes fields for 'Nome do projeto' (Project name) set to 'Jet Hire', 'Local' (Location) set to 'Sem organização' (No organization), and buttons for 'CRIAR' (Create) and 'CANCELAR' (Cancel). A warning message at the top states: 'Voce tem 11 projects restantes na sua cota. Solicite um aumento ou exclua projetos.' with a link to 'Saiba mais'.

Figura 111 - Criação de projeto na GCC

Após a criação do projeto, somos redirecionados para a página principal do projeto.

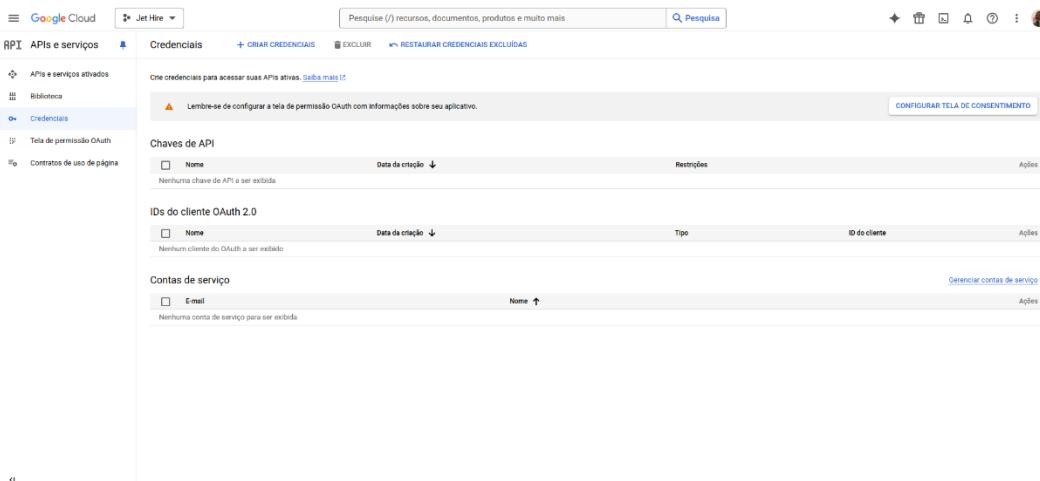


Figura 112 - Dashboard da GCC

Criação do token OAuth

Um **token OAuth** é uma “chave digital” temporária que permite que uma aplicação aceda de forma segura a recursos ou informações de outro serviço sem precisar da tua palavra-passe. Para a criação deste token é preciso criar as credenciais do mesmo:

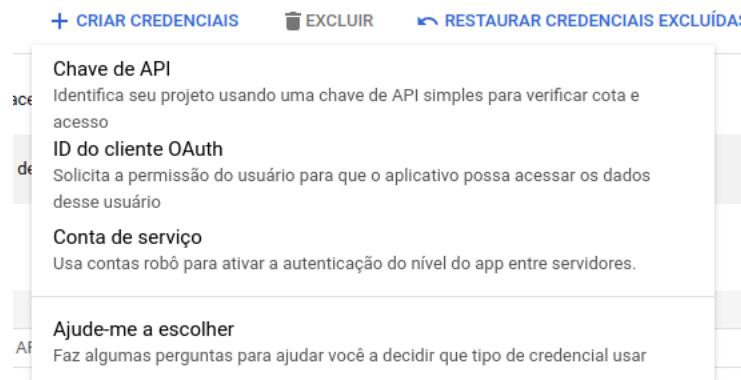


Figura 113 - Criação do token OAuth: Parte 1

De seguida, tive que definir o meu tipo de utilizador. Visto que este token foi utilizado num projeto pessoal, tive que escolher a opção de utilizador externo.

User Type
 Interno 

Disponível apenas para usuários na sua organização. Não será necessário enviar seu aplicativo para verificação. [Saiba mais sobre o tipo de usuário](#)

 Externo 

Disponível para qualquer usuário de teste que tenha uma Conta do Google. O aplicativo ficará no modo de testes e só será disponibilizado aos usuários que você adicionou à lista de usuários de teste. Quando o aplicativo estiver pronto para ser enviado à produção, talvez seja necessário verificá-lo. [Saiba mais sobre o tipo de usuário](#)

CRIAR
[Dê sua opinião](#) sobre nossa experiência com o OAuth

Figura 114 - Criação do token OAuth: Parte 2

Depois da escolha do tipo de utilizador, foi necessário inserir os dados da plataforma, neste caso, do projeto.

Editar registro do app
Informações do app

Essas informações são exibidas na tela de consentimento. Elas informam aos usuários finais quem é você e como entrar em contato.

Nome do app *

O nome do aplicativo que precisa da permissão

E-mail para suporte do usuário *

Para que os usuários entrem em contato com você a respeito do consentimento. [Saiba mais](#)

Logotipo do app

Este é seu logotipo. Ele ajuda as pessoas a reconhecerem o app e aparece em destaque na tela de permissão OAuth.

Depois de fazer upload de um logotipo, será necessário enviar o app para verificação, a menos que esteja configurado para uso interno ou tenha o status de publicação "Teste". [Saiba mais](#)



Visualização do logotipo do app

Arquivo de logotipo a fazer upload

PROCURAR

Faça upload de uma imagem de até 1 MB na tela de permissão para ajudar os usuários a reconhecerem seu app. Os formatos de imagem permitidos são JPG, PNG e BMP. Para garantir o melhor resultado, os logotipos precisam ser quadrados e ter a resolução de 120 px por 120 px.

Figura 115 - Criação do token OAuth: Parte 3

Após isto, é preciso definir o tipo de aplicação que é, e definir um nome para a identificar dentro da Google Cloud Console.

[Criar ID do cliente do OAuth](#)

Um ID do cliente é usado para identificar um único app nos servidores OAuth do Google. Se o app for executado em várias plataformas, cada uma precisará de um ID do cliente. Para acessar mais informações, consulte [Como configurar o OAuth 2.0](#). [Saiba mais](#) sobre tipos de clientes OAuth.

Tipo de aplicativo *	Aplicativo da Web <div style="float: right; margin-top: -20px;">▼</div>
Nome *	Google Auth Jet Hire
O nome do cliente OAuth 2.0. Esse nome é usado apenas para identificar o cliente no console e não será mostrado aos usuários finais.	

Figura 116 - Criação do token OAuth: Parte 4

Por fim, visto que estou a desenvolver este projeto em ambiente local o meu *host* é o *localhost* e é preciso definir isso para a Google conseguir enviar solicitações/pedidos á plataforma. É ainda preciso adicionar URL's de redirecionamento, neste caso, é redirecionado para o código que está dentro do ficheiro [...nextauth].js que mais tarde redireciona para o index.js do endpoint dos utilizadores.

Origens JavaScript autorizadas

Para usar com solicitações de um navegador

URIs 1 *	http://localhost:3000
+ ADICIONAR URI	

URIs de redirecionamento autorizados

Para usar com solicitações de um servidor da Web

URIs 1 *	http://localhost:3000/api/auth/callback/google
+ ADICIONAR URI	

Observação: pode levar de cinco minutos a algumas horas para que as configurações entrem em vigor

[CRIAR](#) [CANCELAR](#)

Figura 117 - Criação do token OAuth: Parte 5

Depois de realizar todo este processo, o *token* está finalmente criado e assim temos acessos ás variáveis necessárias para utilizar no projeto.

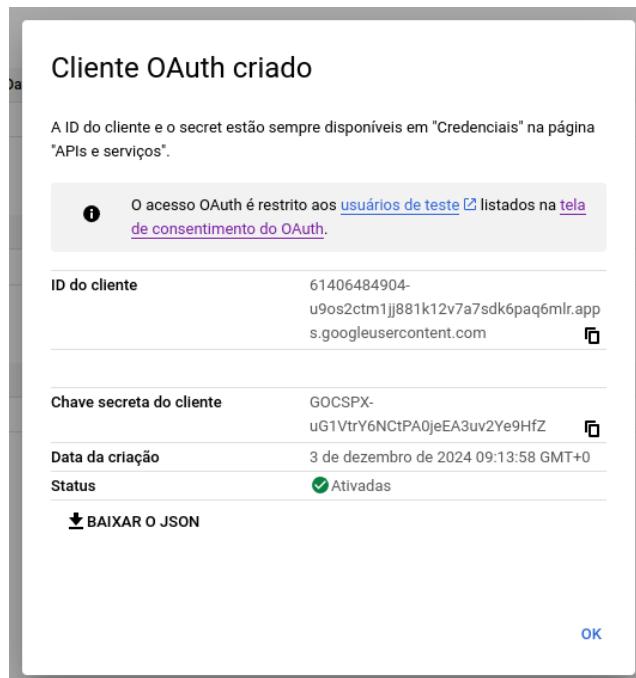


Figura 118 - Criação do token OAuth: Parte 6

Variáveis em ambiente seguro

Depois de ter estas variáveis, é preciso adicioná-las ao projeto. Para isso, é preciso voltar ao ficheiro `.env` e adicionar as mesmas desta forma:

```
GOOGLE_CLIENT_ID=61406484904-u9os2ctm1jj881k12v7a7sdk6paq6mlr.apps.googleusercontent.com
GOOGLE_CLIENT_SECRET=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
```

Adição das credenciais ao Next-Auth

Agora com as variáveis já inseridas no projeto, é necessário adicioná-las no próprio Next-Auth. Para este fim, apenas é necessário ir ao ficheiro `[...nextauth].js` e ao definir a Google Provider adicionar as mesmas desta maneira:

```
import NextAuth from "next-auth";
import GoogleProvider from "next-auth/providers/google";
const authOptions = {
  providers: [
```

```
GoogleProvider({  
    clientId: process.env.GOOGLE_CLIENT_ID,  
    clientSecret: process.env.GOOGLE_CLIENT_SECRET,  
},  
],  
}
```

Credenciais

O método de autenticação por **credenciais** é o **provider mais simples de implementar**, uma vez que requer apenas a introdução de dados básicos como o **email** e a **palavra-passe**.

Este processo decorre em **duas etapas principais**:

- Os dados inseridos pelo utilizador são enviados para o ficheiro de configuração do **NextAuth**;
- O sistema verifica na **base de dados** se existe um utilizador com o email fornecido (sendo este um campo **único**). Se a **palavra-passe** corresponder à armazenada para esse utilizador, a autenticação é validada com sucesso.

Este processo é tratado no **back-end do NextAuth**, através do seguinte bloco de código.

```
const authOptions = {  
  providers: [  
    CredentialsProvider({  
      name: "Credentials",  
      credentials: {  
        email: { label: "Email", type: "email" },  
        password: { label: "Password", type: "password" }  
      },  
      async authorize(credentials) {  
        await connectMongoDB();  
  
        const user = await User.findOne({ email: credentials.email });  
      }  
    })  
  ]  
};
```

```
if (!user) {  
    throw new Error("Utilizador não encontrado");  
}  
  
const isValidPassword = await compare(credentials.password, user.password);  
  
if (!isValidPassword) {  
    throw new Error("Password incorreta");  
}  
  
return { user };  
}  
}  
],  
};  
  
export default NextAuth(authOptions);
```

Após a validação das credenciais, a próxima etapa é a criação de um **token JWT** e da respetiva **sessão**.

Criação de um token JWT

JSON Web Token é um padrão utilizado para autenticação de utilizadores de forma segura, sem depender de sessões tradicionais armazenadas no servidor. O token JWT é composto por três partes principais: o **cabeçalho (header)**, o **corpo (payload)** e a **assinatura (signature)**. Estas três partes permitem que o token seja transmitido de forma compacta e segura entre o servidor e o cliente. Após o login do utilizador, o servidor gera o JWT e envia-o ao cliente, que o envia de volta em cada pedido subsequente, permitindo que o servidor autentique o utilizador sem necessitar de consultar a base de dados a cada requisição.

No contexto do **NextAuth**, a configuração `session.strategy: "jwt"` é utilizada para especificar que a autenticação será gerida por meio de tokens JWT, em vez de sessões baseadas na base de dados. Esta abordagem simplifica a gestão das sessões e melhora a escalabilidade da aplicação.

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

Após o utilizador efetuar login, o NextAuth gera automaticamente um token JWT que contém as informações do utilizador e o armazena num cookie seguro.

A criação e gestão do JWT no NextAuth é configurada através de dois **callbacks**: jwt e session. O **callback jwt** é executado sempre que o token JWT é criado ou atualizado. Quando o utilizador faz login, os dados do utilizador (como id, name, email e image) são armazenados dentro do token, conforme o seguinte bloco de código:

```
callbacks: {  
    jwt({ token, user }) {  
        if (user) {  
            token.id = user.id;  
            token.name = user.name;  
            token.email = user.email;  
            token.image = user.image;  
        }  
        return token;  
    }  
}
```

Este código assegura que as informações essenciais do utilizador sejam adicionadas ao token JWT, permitindo que sejam acessadas posteriormente.

O **callback session** é utilizado para transformar o token JWT em uma sessão que pode ser utilizada pelo cliente. Este callback é executado sempre que a sessão do utilizador é acedida, por exemplo, quando se utiliza o hook `useSession()` no lado do cliente. O conteúdo do JWT é então transferido para o objeto `session.user`, conforme exemplificado a seguir:

```
async session({ session, token }) {  
    session.user.id = token.id;  
    session.user.name = token.name;  
    session.user.email = token.email;
```

```
session.user.image = token.image;  
return session;  
}
```

Este fluxo permite que os dados do utilizador sejam acedidos de forma eficiente e segura no lado do cliente, sem necessidade de consultar repetidamente a base de dados ou realizar autenticações adicionais.

A utilização de JWT no NextAuth oferece uma solução **eficiente, segura e escalável** para autenticação de utilizadores, uma vez que as informações do utilizador são mantidas dentro do token, que é assinado e, portanto, protegido contra alterações. Além disso, a utilização de tokens JWT facilita a gestão de sessões em aplicações de grande escala.

A partir deste ponto, é possível utilizar o hook **useSession()** para aceder às quatro informações armazenadas durante o início da sessão.

Barra de navegação funcional

Para tornar a **barra de navegação funcional**, foi implementada a lógica de exibição dinâmica baseada no estado da autenticação do utilizador.

Utilizando o **hook useSession()** do NextAuth, a aplicação consegue verificar se o utilizador está autenticado e exibir as opções de navegação apropriadas. O código seguinte exemplifica como a barra de navegação se adapta ao estado de autenticação:

```
import { useSession } from 'next-auth/react';  
const { status, data: session } = useSession();  
  
{status === 'authenticated' ? (  
    <ul className="main-menu">  
        <li className="has-children">  
            <a>  
                <span className="mr-10 navbar-name">{session?.user?.name}</span>  
                {session?.user?.image && (  
                    <img  
                        src={session.user.image} />  
                )}  
            </a>  
        </li>  
    </ul>  
) : (  
    <ul>  
        <li>Login</li>  
        <li>Logout</li>  
    </ul>  
)}
```

```

height={30}

width={30}

style={{ borderRadius: '50%', marginRight: "5px", objectFit: "cover" }}

/>


)}

</a>

<ul className="sub-menu">

<li>

<Link legacyBehavior href={`/profile/${session.user.id}`}>

<button>

<i className="fa-solid fa-user mr-5"></i>Ver perfil

</button>

</Link>

</li>

<li>

<button>

<i className="fa-solid fa-gear mr-5"></i>Definições

</button>

</li>

<li>

<button onClick={() => handleSignOut()}>

<i className="fa-solid fa-person-running mr-5"></i>Sair

</button>

</li>

</ul>

</li>

</ul>

) : (



<>

<Link legacyBehavior href="page-register">

<a className="">Criar conta</a>

</Link>

```

```

<Link legacyBehavior href="page-signin">
    <a className="btn btn-default btn-shadow ml-40">Login</a>
</Link>
</>
)}

```

No código acima, a função **useSession()** verifica se o estado da sessão é '**authenticated**'. Se o utilizador estiver autenticado, a barra de navegação exibe o nome do utilizador, a sua imagem de perfil (se disponível) e opções adicionais, como o **link para o perfil**, **definições** e **sair**. Caso o utilizador não esteja autenticado, são apresentados os **links para criar uma conta e fazer login**.

A utilização do **session?.user?.name** e **session?.user?.image** permite que as informações do utilizador sejam dinamicamente mostradas na barra de navegação, enquanto o botão de **logout** chama a função **handleSignOut()** para encerrar a sessão do utilizador.

Esta implementação torna a navegação mais personalizada e funcional, proporcionando uma **experiência de utilizador fluida e dinâmica**, onde as opções da barra de navegação mudam automaticamente com base no estado de autenticação.



Figura 119 - Barra de navegação sem sessão



Figura 120 - Barra de navegação com sessão

Perfil dinâmico

Na minha aplicação, um dos aspetos mais importantes e desafiantes foi a construção de um **perfil dinâmico**. Em vez de criar uma página estática, onde todos os perfis têm exatamente a mesma estrutura e conteúdo, optei por implementar uma página que se

adapta automaticamente a cada utilizador com base nos dados reais que estão na base de dados.

Estrutura da página

A estrutura principal do perfil divide-se em **duas áreas principais**:

- **Mainbar (conteúdo principal)** – onde são apresentadas as informações mais relevantes do utilizador, como:
 - Secção "Sobre mim"
 - Competências profissionais
 - Experiência profissional
 - Educação
 - Projetos de portfólio
- **Sidebar (barra lateral)** – funciona como um complemento, que apresenta:
 - Uma visão geral do perfil (experiência, línguas, educação, etc.)
 - Ligações para redes sociais (GitHub, LinkedIn, site pessoal)
 - Informações de contacto
 - Um alerta de preenchimento (caso o perfil ainda esteja incompleto)

Esta separação permite que o utilizador tenha uma **visão mais organizada e funcional** da sua página, e também melhora a experiência de leitura para quem estiver a visitar o seu perfil.

Recolha de dados com `getServerSideProps`

Para que o perfil seja realmente dinâmico, foi essencial garantir que os dados do utilizador estivessem disponíveis **logo no momento em que a página fosse carregada**. Para isso, usei uma funcionalidade do Next.js chamada `getServerSideProps`.

Esta função corre **do lado do servidor** sempre que alguém acede a um perfil, e é responsável por ir buscar à base de dados todas as informações do utilizador com base no ID presente no URL. O código é o seguinte:

```
export async function getServerSideProps(context) {  
  const { id } = context.params;  
  
  try {  
    await connectMongoDB();  
  } catch (err) {  
    console.error(err);  
  }  
}
```

```
const user = await User.findById(id).lean();

if (!user) {
    return { notFound: true };
}

const verificationResult = checkProfileCompletion(user.profile);
const verificationMainResult = checkMainProfile(user.profile);
const verificationSidebarResult = checkSidebarProfile(user.profile);

return {
    props: {
        user: JSON.parse(JSON.stringify(user)),
        profileComplete: verificationResult.complete,
        profilePercentage: verificationResult.percentage,
        mainPercentage: verificationMainResult.percentage,
        sidebarPercentage: verificationSidebarResult.percentage,
    },
};

} catch (error) {
    console.error(error);
    return {
        props: {
            user: null,
            profileComplete: false,
            profilePercentage: 0,
            mainPercentage: 0,
            sidebarPercentage: 0,
        },
    };
}
}
```

Este método é importante não só porque garante que a página carrega com os dados certos, mas também porque melhora o **desempenho e o SEO da aplicação**, uma vez que os conteúdos são renderizados no servidor antes de chegarem ao *browser*.

Renderização condicional e layout responsivo

Como nem todos os utilizadores têm o perfil completo, utilizei várias **condições** para mostrar ou esconder secções consoante os dados disponíveis.

Por exemplo, o conteúdo principal (mainbar) só é mostrado se houver dados para apresentar. Caso contrário, aparece uma mensagem a avisar que o utilizador ainda não configurou o seu perfil:

```
{mainPercentage != 0 ? (
  <div>
    /* Conteúdo do perfil como "Sobre mim", "Competências", etc. */
  </div>
) : (
  <div className="profile-noreresults">
    <i className="fa-solid fa-heart-crack"></i>
    <h5>Parece que este utilizador ainda não configurou o seu perfil...</h5>
    {session?.user?.id === user._id && (
      <div className="text-center mt-20">
        <a href={`/edit/${user._id}`} className="btn btn-default">Editar perfil</a>
      </div>
    )}
  </div>
)}
```

A lógica da sidebar segue o mesmo princípio. Se não houver dados suficientes para preencher essa área, a mainbar **expande-se automaticamente** para ocupar toda a largura da página, graças a esta condição nas classes CSS baseadas em BootStrap:

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

Isto garante que o layout se ajusta dinamicamente consoante o conteúdo existente, mantendo sempre um visual limpo e apelativo.

Percentagem de Preenchimento

Para melhorar a experiência do utilizador, implementei três funções que analisam diferentes partes do perfil e calculam a respetiva **percentagem de preenchimento**:

- checkProfileCompletion – verifica se o perfil está globalmente completo
- checkMainProfile – verifica se há conteúdo suficiente na mainbar
- checkSidebarProfile – analisa os dados para a sidebar

Exemplo simplificado de como estas funções funcionam:

```
const requiredFields = ["city", "country", "aboutMe", "skills", ...];

let filledCount = 0;
requiredFields.forEach(field => {
  if (Array.isArray(profile[field])) {
    if (profile[field].length > 0) filledCount++;
  } else if (profile[field]) {
    filledCount++;
  }
});

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

Com base nestas percentagens, são apresentadas mensagens ao utilizador:

```
{!profileComplete ? (
  <>
  <h5 className="f-14">Parece que ainda não completou o seu perfil.</h5>
  <p>Vamos mudar isso!</p>
  <h5 className="f-14 mt-5">{profilePercentage}% completo</h5>
</>
) : (
  <>
```

```
<h5 className="f-14">O seu perfil está completo!</h5>
<p>Se quiser fazer alguma alteração, ainda pode.</p>
</>
)}
```

Nas seguintes imagens pode ver as diferentes versões:



Figura 121 - Verificação de perfil completa



Figura 122 - Verificação de perfil por concluir

Segurança e personalização

Por fim, garanti que apenas o próprio utilizador pode editar o seu perfil, usando esta verificação de segurança:

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

Assim, protejo os perfis de acessos não autorizados e mantenho a integridade da aplicação.

Tomé Almeida
À procura de emprego

Parece que este utilizador ainda não configurou o seu perfil...

Copyright © 2025. Desenvolvido por Tomé Almeida | Política de privacidade | Termos & Condições | Política de cookies

Figura 123 - Exemplo de perfil por preencher


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

Tomé Almeida Murtede, Portugal
Full-Stack Developer

 [Curriculum Vitae](#)

Sobre mim

... ipsum dolor sit amet, consectetur adipiscing elit. Sed convallis mauris quis arcu ultrices, sed vestibulum nibh rhoncus. Curabitur pulvinar auctor nunc, a pretium justo tincidunt id. Morbi eleifend elit felis, venenatis rutrum massa rhoncus non. Pellentesque aliquet tellus consectetur odio finibus hendrerit sit amet vel sem. Duis ipsum quam, sodales ut convallis eget, aliquam ac ligula. Sed condimentum rhoncus mi ac ornare. Aliquam vel egestas eros, non consectetur risus. Donec velit nisl, maximus vel vestibulum id, suscipit in orci. Phasellus efficitur mollis nulla non semper. Nulla vestibulum, ex sed hendrerit dapibus, lacus erat vestibulum urna, eu suscipit lectus magna at est. Integer nec ipsum quis nunc facinia finibus sit amet in lorem. In rutrum sed magna mollis consequat. Quisque neque nibh, pulvinar vitae ultricies ut, consequat at elit. Suspendisse potenti. Aenean mattis fringilla gravida.

Phasellus fermentum dolor nibh, in auctor nibh molestie ac. Nam porta neque at nisi egestas feugiat. Sed vel erat est. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Sed sagittis porttitor facilisis. Etiam eu nulla enim. Fusce consequat nisi nisi, molestie elementum nunc auctor id. Ut feugiat metus vitae lacus mollis, egestas egestas sapien imperdiet. Nunc quis pharetra lorem, non cursus dui. Mauris sit amet diam dapibus, pharetra lacus eu, egestas lacus. Aenean vel ante et risus sodales accumsan. Curabitur suscipit felis nec imperdiet sagittis. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Fusce nec semper lectus. Suspendisse tincidunt semper pharetra.

Competências profissionais



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


O seu perfil está completo!

Se quiser fazer alguma alteração, ainda pode.

[Editar perfil](#)

Visão geral

- Experiência
Sem experiência
- Línguas
Português, Inglês
- Nível de educação
Curso Técnico-Profissional
- Site pessoal
- GitHub
- LinkedIn

Localização: Murtede, Portugal

Telemóvel: 965360269

Email: ritomealmeida@gmail.com

 [Entrar em contacto](#)
Figura 124 - Exemplo de perfil completo

Edição de perfil

A secção de **edição de perfil** da aplicação foi desenvolvida com o objetivo de oferecer aos utilizadores uma forma **simples e eficiente** de atualizar os seus dados pessoais e profissionais. No entanto, apesar da interface ser intuitiva, a lógica por trás desta funcionalidade é bastante avançada, envolvendo a **gestão dinâmica do estado da aplicação, integração com serviços externos** como o **Cloudinary**, e uma **arquitetura modular** baseada em componentes reutilizáveis.

Atualização dos dados em tempo real

A base da edição está construída sobre um **formulário dinâmico em React**, controlado pelo estado formData, onde todos os dados do utilizador são armazenados temporariamente até serem submetidos.

Ao carregar a página, os dados do utilizador são recebidos via getServerSideProps, e o estado local é preenchido com os mesmos:

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

Cada input do formulário está ligado ao formData, o que permite que qualquer alteração seja **imediatamente refletida**. Quando o utilizador submete o formulário, os dados são enviados para a **API de atualização de perfil**:

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

Modularização do formulário

Para manter o código organizado e facilitar futuras expansões, cada parte do perfil foi dividida em **componentes independentes**, como Education, Experience, Skills, Languages, entre outros. Por exemplo, o componente Education permite ao utilizador adicionar múltiplos registos de formação académica. Os dados são adicionados dinamicamente e inseridos no formData através de:

```
setFormData((prevFormData) => ({
```

```

...prevFormData,
profile: {
  ...prevFormData.profile,
  education: [...(prevFormData.profile?.education || []), updatedEducation],
},
});

```

Esta abordagem **modular** melhora a organização do código e permite **reaproveitar estruturas semelhantes**, como acontece com o componente Experience.

Preparação do Cloudinary

Para utilizar o **Cloudinary**, é necessário seguir alguns passos, começando pela criação de uma conta na plataforma.

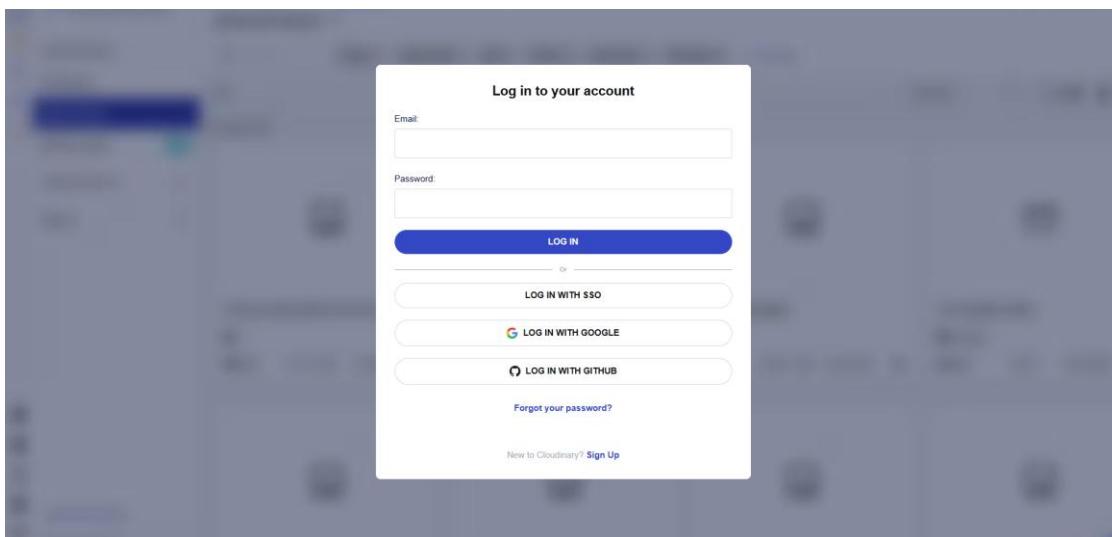
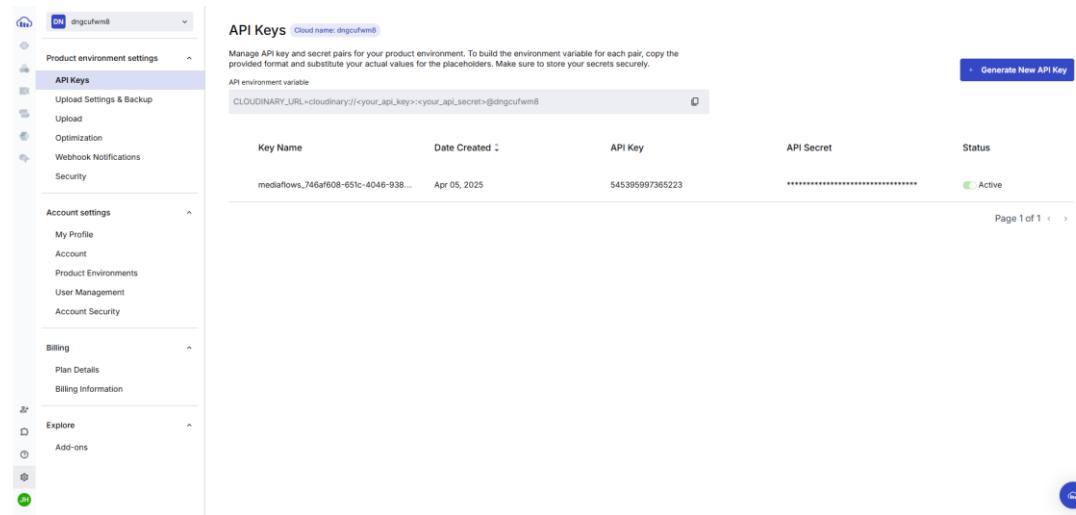


Figura 125 - Criação de conta no Cloudinary

Após o registo, o utilizador é direcionado para uma página onde pode aceder às **API Keys**, bastando clicar em "**Go To API Keys**". Essas credenciais devem ser copiadas e inseridas no ficheiro .env do projeto. Um exemplo do formato da variável é seguinte:



The screenshot shows the Jet Hire interface for managing API keys. The left sidebar includes sections for Product environment settings (API Keys, Upload Settings & Backup, Upload, Optimization, Webhook Notifications, Security), Account settings (My Profile, Account, Product Environments, User Management, Account Security), Billing (Plan Details, Billing Information), Explore, and Add-ons. The main content area is titled "API Keys" and shows a table with one row. The table columns are Key Name, Date Created, API Key, API Secret, and Status. The status is marked as "Active". The API key itself is displayed as a long string: CLOUDINARY_URL=cloudinary://545395997365223:y77xzgR5ETuuv0Ema4TiOEkyH7Q@dngcufwm8. A "Generate New API Key" button is located at the top right of the table.

Figura 126 - Página de "API Keys"

Essas credenciais devem ser copiadas e inseridas no ficheiro `.env` do projeto. Um exemplo do formato da variável é o seguinte:

```
CLOUDINARY_URL=cloudinary://545395997365223:y77xzgR5ETuuv0Ema4TiOEkyH7Q@dngcufwm8
```

Com as credenciais configuradas, o próximo passo é importar a biblioteca do **Cloudinary** e definir a **configuração** base através do seguinte bloco de código:

```
import { v2 as cloudinary } from "cloudinary";

cloudinary.config({
  secure: true,
});

export const config = {
  api: {
    bodyParser: {
      sizeLimit: "2mb",
    },
  },
};
```

Este código garante que a ligação ao serviço é feita de forma **segura** e também permite definir o tamanho máximo dos ficheiros enviados — neste exemplo, o limite está fixado em 2MB, podendo ser **ajustado** conforme as necessidades da aplicação.

Upload Presets do Cloudinary

No **Cloudinary**, os **upload presets** são como “regras pré-definidas” que se criam para facilitar o envio de ficheiros (imagens, vídeos, etc.). Em vez de se definir tudo manualmente sempre que se faz um upload, o preset já tem tudo configurado.

Por exemplo, é possível criar um preset que diga:

- “Quero que todas as imagens vão para a pasta perfis.”
- “Quero que fiquem privadas, só acessíveis com token.”
- “Quero que redimensionem automaticamente para 500x500.”
- “Quero que tenham a tag foto-perfil.”

Depois, sempre que se usar esse preset, o Cloudinary já sabe exatamente o que fazer — **poupa-se tempo e evita-se erros**.

O único *preset* presente neste projeto é para o portfólio para facilitar assim o processo de envio de várias imagens para cada utilizador.

Upload de ficheiros com Cloudinary

Uma das partes **mais complexas** desta funcionalidade foi a **integração com o Cloudinary** para o **upload e substituição de ficheiros**. Três elementos do perfil — **foto de perfil, imagem de banner e currículo** — são enviados para o Cloudinary diretamente a partir da **API /api/profile**.

Antes de um novo upload, o sistema verifica se já existe um ficheiro antigo e, se existir, este é removido:

```
if (oldId) {  
    await cloudinary.uploader.destroy(oldId);  
}
```

De seguida, o novo ficheiro é carregado para o Cloudinary, e os dados atualizados:

```
const uploadRes = await cloudinary.uploader.upload(`data:image/jpeg;base64,${pfp}`, {
    folder: "pfp",
});

formData.profile = {
    ...formData.profile,
    pfp: uploadRes.secure_url,
    pfp_id: uploadRes.public_id,
};
```

Esta lógica assegura que **não ficam ficheiros antigos a ocupar espaço**, mantendo o armazenamento limpo e profissional.

Portfólio com gestão de imagens

O componente Portfolio é uma das partes **mais interativas** e visualmente atrativas da edição de perfil. O utilizador pode adicionar projetos ao seu portfólio, sendo necessário preencher três campos: **legenda, link e imagem**.

Quando a imagem é selecionada, esta é enviada diretamente do cliente para o **Cloudinary**, utilizando o endpoint público:

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

const response = await fetch('https://api.cloudinary.com/v1_1.../image/upload', {
    method: 'POST',
    body: formData,
});
```

A imagem é então associada ao projeto e guardada no estado formData. Caso o utilizador remova um projeto, a imagem correspondente é também **apagada do Cloudinary** através de uma chamada à API /api/delete-image:

```
await fetch('/api/delete-image', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
```

```
    body: JSON.stringify({ public_id: itemToRemove.image.id }),  
});
```

Este sistema garante uma **gestão automática de recursos na cloud**, sem desperdícios nem ficheiros desnecessários.

Segurança e permissões

Para garantir que apenas o **próprio utilizador** pode editar o seu perfil, foi implementada uma verificação de sessão no getServerSideProps. Se o ID da sessão não coincidir com o ID presente na URL, o acesso é bloqueado e o utilizador é redirecionado:

```
if (!session || !session.user || session.user.id !== id) {  
  return {  
    redirect: {  
      destination: '/',  
      permanent: false,  
    },  
  };  
}
```

Esta medida reforça a **segurança e privacidade dos dados** do utilizador.

Página de candidatos

Uma das funcionalidades centrais da plataforma foi a criação de uma **página de consulta de candidatos**. Esta secção permite às entidades empregadoras **explorar o conjunto de utilizadores disponíveis**, com ferramentas para **pesquisa por nome** e **ordenação por data de registo**.

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

O componente CandidateGrid é responsável por toda a lógica e renderização desta página.

A estrutura divide-se em três partes principais:

- **Cabeçalho e estatísticas** — apresenta o número total de candidatos registados.
- **Campo de pesquisa e filtros de ordenação** — permite filtrar resultados de forma dinâmica.
- **Listagem condicional dos candidatos** — adaptável conforme o estado de carregamento, presença de resultados ou ausência total.

Logo no início, são definidos vários estados importantes:

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

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.

Carregamento dos dados com feedback visual

No useEffect, é feita uma **chamada à API /api/candidates** assim que a página é carregada. Durante este processo, o estado isLoading é ativado para mostrar um indicador visual ao utilizador:

```
useEffect(() => {  
  const fetchUsers = async () => {  
    setIsLoading(true);  
    const response = await fetch('/api/candidates');  
    const data = await response.json();  
    setUsers(data);  
    setIsLoading(false);  
  };  
  fetchUsers();  
}, []);
```

Durante o carregamento, o utilizador vê o seguinte:

```
{isLoading && (
```

```
<div className="text-center py-5">
  <div className="spinner-border text-danger" role="status" />
  <p className="mt-3">A carregar candidatos...</p>
</div>
)}
```

Este pequeno pormenor **melhora muito a experiência de navegação**, pois evita que o utilizador fique sem saber o que está a acontecer.



A carregar candidatos...

Figura 127 - Exemplo de "A carregar candidatos"

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 <strong>{filteredUsers.length}</strong> de  
<strong>{users.length}</strong> candidatos  
</span>
```

A mostrar 5 de 5 candidatos

Figura 128 - Exemplo de contador de apresentação de utilizadores

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>
```

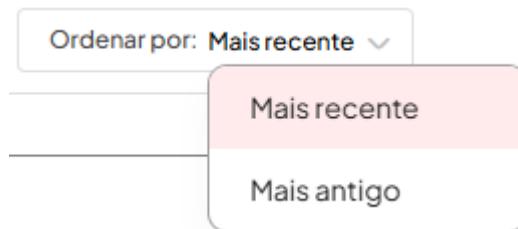
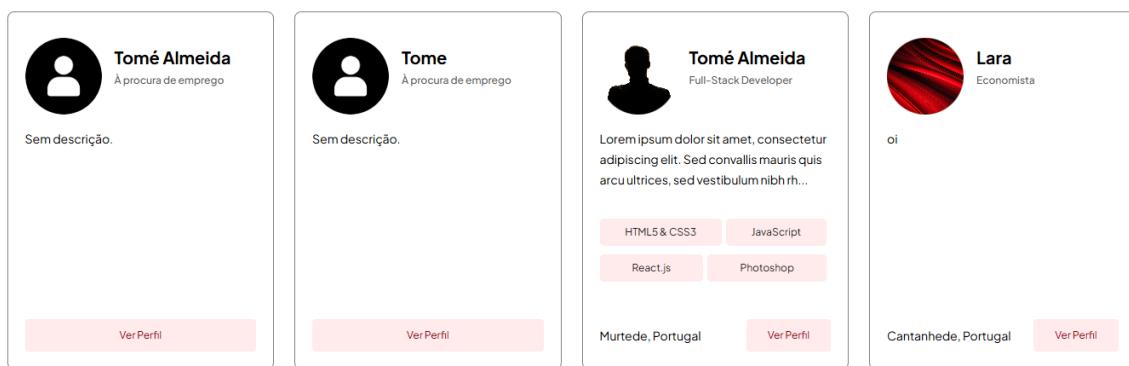


Figura 129 - Exemplo de "dropdown" na aplicação

A screenshot of a web application showing a list of candidates. At the top left, it says 'A mostrar 5 de 5 candidatos'. On the right, there's a dropdown menu labeled 'Ordenar por: Mais recente'. Below this, there are four candidate profiles in cards:

- 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
Murteide, Portugal [Ver Perfil](#)
- Tome** (À procura de emprego)
Sem descrição.
[Ver Perfil](#)

Figura 130 - Candidatos ordenados por "Mais Recente"





Tomé Almeida
À procura de emprego

Sem descrição.

[Ver Perfil](#)



Tome
À procura de emprego

Sem descrição.

[Ver Perfil](#)



Tomé Almeida
Full-Stack Developer

Lore 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](#)



Lara
Economista

oi

Cantanhede, Portugal [Ver Perfil](#)

Figura 131 - Candidatos ordenados por "Mais Antigo"

Renderização condicional com mensagens de fallback

A apresentação final dos candidatos depende de três estados possíveis:

- **Carregamento em curso:** (isLoading)
- **Nenhum resultado encontrado:** (filteredUsers.length === 0)
- **Lista de candidatos:** disponível

Cada um destes casos é tratado de forma clara:

Se estiver a carregar:



5 candidatos disponíveis

Descobre talentos prontos para integrar a sua equipa.

Palavras-chave...

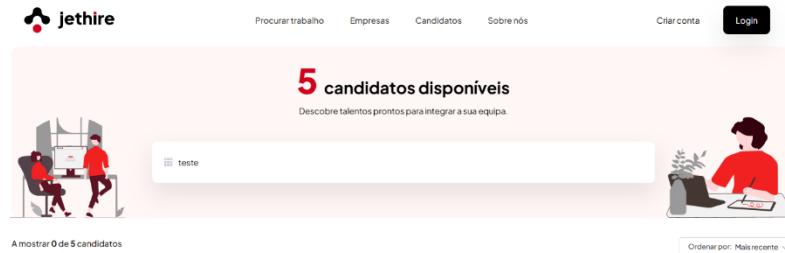
A mostrar 5 de 8 candidatos

Ordenar por: Mais recente

A carregar candidatos...

Figura 132 - Candidatos a carregar

Se não houver resultados:



5 candidatos disponíveis
Descobre talentos prontos para integrar a sua equipa.

A mostrar 0 de 5 candidatos

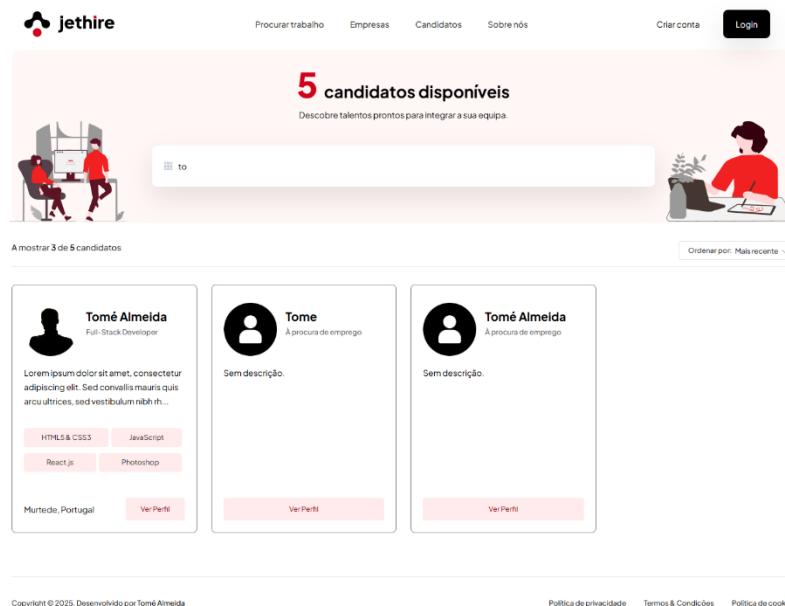
Ordenar por: Mais recente

A
Não existem resultados.

Copyright © 2025. Desenvolvido por Tomé Almeida | Política de privacidade | Termos & Condições | Política de cookies

Figura 133 - Sem resultados

Se houver resultados válidos:



5 candidatos disponíveis
Descobre talentos prontos para integrar a sua equipa.

A mostrar 3 de 5 candidatos

Ordenar por: Mais recente

 Tomé Almeida Full-Stack Developer <small>Locação: Portugal</small> <small>HTML5 & CSS3, JavaScript, React.js, Photoshop</small> <small>Murdeia, Portugal</small> Ver Perfil	 Tome À procura de emprego <small>Sem descrição.</small> Ver Perfil	 Tomé Almeida À procura de emprego <small>Sem descrição.</small> Ver Perfil
--	---	---

Figura 134 - Resultado da pesquisa de candidatos

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

Página de empresas

Registo de empresas

Para permitir o registo de novas empresas na plataforma, foi desenvolvido um **formulário interativo** localizado no final da página, conforme ilustrado na imagem seguinte:



Figura 135 - Placeholder do formulário do registo de empresas

Quando o utilizador **clica sobre o formulário**, este expande-se, permitindo o preenchimento dos dados necessários para o registo da empresa.

A interface web do formulário de registo de empresas. No topo, uma barra negra com o logo "Jethire" e links para "Procurar trabalho", "Empresas", "Candidatos", "Jet AI", "Sobre nós", "Criar conta" e "Logout". O formulário principal tem um fundo branco e é dividido em seções: "Dados da empresa" (Nome, NIF, Ano de fundação), "Localização" (País, Cidade, Morada completa), "Contactos" (Telefone, Email, Website) e "Responsável pelo registo" (Nome, Email, Telefone). Cada seção contém campos de texto e botões para selecionar opções. Um botão "Enviar" está no fundo do formulário. No rodapé, há links para "Copyright ©2025. Desenvolvido por Tomé Almeida", "Política de privacidade", "Termos & Condições" e "Política de cookies".

Figura 136 - Formulário de registo de empresas

Após o preenchimento completo e ao submeter o formulário, o **pedido de registo é enviado para a base de dados**, seguindo o processo previamente descrito.

Estrutura geral da página de candidatos

A página apresenta, em primeiro lugar, um **cabeçalho com funcionalidade de pesquisa**, que permite ao utilizador procurar empresas com base no nome, descrição e outros critérios relevantes.



Figura 137 - Cabeçalho da página de empresas

Logo abaixo, encontra-se uma **barra lateral de filtros avançados**, concebida para ajudar o utilizador a refinar os resultados e encontrar com maior precisão as empresas que melhor correspondem ao seu perfil e interesses.



Experiência

<input type="checkbox"/> < 1 ano	56
<input type="checkbox"/> 1-2 anos	87
<input checked="" type="checkbox"/> 2-3 anos	24
<input type="checkbox"/> 3-4 anos	45
<input type="checkbox"/> 4-5 anos	76
<input type="checkbox"/> > 5 anos	89

Modalidade

<input type="checkbox"/> Presencial	12
<input checked="" type="checkbox"/> Remoto	65
<input checked="" type="checkbox"/> Híbrido	58

Figura 138 - Secção 1 da barra lateral da página de empresas

Figura 139 - Secção 2 da barra lateral da página de empresas

Por fim, surge a **secção principal da página**, onde são exibidas as diferentes empresas, apresentadas sob a forma de **cartões informativos**. Cada cartão contém os dados essenciais de cada empresa, proporcionando uma visão clara e organizada das opções disponíveis.



Figura 140 - Apresentação das empresas

Criação da dashboard

Para garantir maior **segurança e organização**, o projeto foi dividido em duas partes distintas: uma dedicada às **empresas** e outra reservada aos **administradores** da aplicação. Esta separação permite que cada grupo tenha acesso apenas às funcionalidades relevantes, garantindo uma gestão mais eficaz e segura.

A dashboard foi pensada para oferecer:

- **Ferramentas de moderação** para administradores (gestão de utilizadores, empresas e ofertas).
- **Funcionalidades empresariais** para os membros da empresa (edição de perfil, criação de ofertas, gestão da equipa, etc).

Middleware

Logo no início do desenvolvimento, foi criado um **middleware** que restringe o acesso à aplicação apenas a utilizadores com sessão iniciada. Este sistema garante que **nenhuma página sensível é acedida sem autenticação**, exceto algumas rotas públicas como:

- /signin (início de sessão)
- /signup (registo)

- /invite (convites)

Além disso, foi dada permissão para o carregamento de recursos como **favicon**, **imagens** e outros ficheiros estáticos essenciais.

```
import { NextResponse, NextRequest } from 'next/server';
import { getToken } from 'next-auth/jwt';

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) {
    const signInUrl = new URL('/signin', req.url);
    return NextResponse.redirect(signInUrl);
  }

  return NextResponse.next();
}

export const config = {
```

```
matcher: ['/((?!api|_next|favicon.ico|images|public|lib).*)'],  
};
```

Criação de modelos de empresa e de administrador

Modelo de Administrador

Os administradores são representados por um modelo simples, que contém apenas os **dados essenciais** para a gestão da aplicação.

Este modelo inclui:

- Nome de utilizador
- Nome real
- Email
- Palavra-passe encriptada
- Contactos e redes sociais
- Estado de ativação

É um modelo leve, visto que os administradores não interagem com o público da mesma forma que as empresas.

```
import mongoose from "mongoose";  
  
const administratorSchema = new mongoose.Schema({  
  
    user: { type: String, required: true },  
  
    name: { type: String, required: true },  
  
    email: { type: String, required: true, lowercase: true },  
  
    password: { type: String, required: true },  
  
    role: {  
  
        type: String,  
  
        default: 'jethire-admin',  
  
        required: true,  
  
    },  
  
    logo: {  
  
        secure_url: String,
```

```

    public_id: String,
},
city: { type: String },
country: { type: String },
phone: { type: String },
socials: [
  {
    platform: String,
    url: String
  },
  {
    type: Boolean,
    default: true,
  },
  { _id: true }
];

export default mongoose.models.Administrator ||
mongoose.model("Administrator", administratorSchema);

```

Modelo de Empresa

As empresas têm um modelo mais complexo, que representa não só a sua identidade como também a **estrutura interna da equipa** e os **pedidos pendentes**.

O modelo inclui:

- **Informações básicas** (nome, localização, descrição, área de atuação)
- **Imagens institucionais** (logótipo e banner)
- **Tags e características do trabalho**
- **Equipa da empresa** (cada membro com os seus dados e permissões)
- **Pedidos pendentes** com estado e token de ativação
- **Secções de descrição** com títulos e texto, que permitem apresentar a empresa de forma detalhada no perfil

Esta estrutura permite representar empresas com diferentes tamanhos, funções e níveis de hierarquia.

```
import mongoose, { Schema } from "mongoose";
```

```
const descriptionSchema = new Schema(  
{  
    title: { type: String, required: true },  
    text: { type: String, required: true },  
,  
    { _id: false }  
);  
  
const userSchema = new mongoose.Schema(  
{  
    user: { type: String, required: true },  

```

```
    default: true,  
  },  
},  
{ _id: true }  
);  
  
const pendingSchema = new mongoose.Schema(  
{  
  name: { type: String, required: true },  
  email: { type: String, required: true, lowercase: true },  
  role: {  
    type: String,  
    enum: ['admin', 'manager', 'recruiter'],  
    default: 'recruiter',  
    required: true,  
  },  
  status: {  
    type: String,  
    enum: ['active', 'pending'],  
    default: 'pending',  
    required: true,  
  },  
  token: { type: String, required: true },  
  expiresAt: { type: Date, required: true },  
},  
{ timestamps: true }  
);  
  
const companySchema = new Schema(  
{  
  type: { type: String, default: "company" },  
  
  name: { type: String, required: true },
```

```
slogan: String,  
  
city: String,  
country: String,  
address: String,  
  
shortDesc: String,  
description: [descriptionSchema],  
  
field: String,  
workType: String,  
foundationYear: Number,  
  
tags: { type: [String], default: [] },  
remote: { type: String, default: "" },  
  
banner: {  
    secure_url: String,  
    public_id: String,  
},  
  
logo: {  
    secure_url: String,  
    public_id: String,  
},  
  
contact: {  
    email: String,  
    phone: String,  
    website: String,  
},  
  
isActive: {
```

```
    type: Boolean,  
    default: true,  
,  
  
    team: [userSchema],  
    pending: [pendingSchema],  
  
,  
{ timestamps: true }  
);  
  
const Company =  
mongoose.models.Company || mongoose.model("Company", companySchema);  
  
export default Company;
```

Depois de ter todos os modelos definidos podemos passar para a página de login.

Início de sessão

Interface de Login

A página de login foi adaptada a partir de um **template** já existente, ao qual foram aplicadas alterações de estilo, cores e logótipo da aplicação. O objetivo foi garantir uma **identidade visual coesa** desde o início da interação do utilizador com a plataforma.

[« Back to dashboard](#)
Sign In

Enter your email and password to sign in!

 Sign in with Google
 Sign in with X

Or

Email *

Password *



Keep me logged in
[Forgot password?](#)

Sign in

[Don't have an account? Sign Up](#)
 TailAdmin

Free and Open-Source Tailwind CSS Admin Dashboard Template


Figura 141 - Página de início de sessão da dashboard inicial
Seja bem-vindo!

Coloque as credenciais para iniciar sessão.

Utilizador *

Palavra-passe *



Entrar

[Tem um token de acesso? Entre por aqui](#)
 jethire

Procura emprego, contrata talentos.


Figura 142 - Página de início de sessão da dashboard final

Configuração da Autenticação (Next-Auth)

Foi implementado o **Next-Auth** com um sistema de autenticação por **credentials (utilizador e palavra-passe)**.

```
import { NextAuthOptions } from "next-auth";
import CredentialsProvider from "next-auth/providers/credentials";
import bcrypt from "bcrypt";
import dbConnect from "@/lib/dbConnect";
```

```
import Administrator from "@/models/admin";
import Company from "@/models/company";

interface TeamMember {
  user: string;
  password: string;
  role: string;
  company: string;
}

export const authOptions: NextAuthOptions = {
  providers: [
    CredentialsProvider({
      name: "Credentials",
      credentials: {
        user: { label: "Utilizador", type: "text" },
        pass: { label: "Palavra-passe", type: "password" },
      },
      async authorize(credentials) {
        await dbConnect();
        const { user, pass } = credentials!;

        const admin = await Administrator.findOne({ user });
        if (admin) {
          const isPasswordValid = await bcrypt.compare(pass, admin.password);
          if (!isPasswordValid) throw new Error("Credenciais inválidas.");
          return {
            id: admin._id.toString()
          };
        }
      }
    })
  ]
}
```

```
const teamMember = company.team.find((member: TeamMember) =>
member.user === user);

if (teamMember) {
    const isPasswordValid = await bcrypt.compare(pass, teamMember.password);
    if(!isPasswordValid) throw new Error("Credenciais inválidas.");
    return {
        id: teamMember._id.toString(),
    };
}

}

throw new Error("Credenciais inválidas.");
},
}),
],
pages: {
    signIn: "/signin",
},
session: {
    strategy: "jwt",
},
secret: process.env.NEXTAUTH_SECRET,
callbacks: {
    async jwt({ token, user }) {
        if (user) {
            token.id = user.id;
        }
        return token;
    },
    async session({ session, token }) {
        if (token) {
            session.user.id = token.id;
        }
        return session;
    }
}
```

```
 },  
 },  
 };
```

Este sistema verifica os dados inseridos em duas coleções distintas:

1. **Coleção de administradores**
2. **Coleção de membros da empresa**

Se o utilizador for encontrado numa destas coleções e a palavra-passe for válida, é criada uma **sessão segura**, onde é guardado o **ID do utilizador**. A sessão utiliza **JWTs (JSON Web Tokens)** para garantir a segurança e escalabilidade da autenticação.

Este processo assegura que apenas **utilizadores legítimos** conseguem aceder à aplicação, e que cada sessão carrega os dados mínimos necessários para identificar o utilizador.

Registo e Validação de Empresas

Após um pedido de registo de empresa ser submetido, este pode ser **validado ou rejeitado** por um administrador.

Para isso, foi criada uma **página de gestão de pedidos**, onde os administradores conseguem:

- Visualizar os dados de cada empresa
- Aceder rapidamente ao contacto
- Validar ou eliminar o pedido com apenas um clique

Glacier Systems

i **Dados da Empresa**

NIF: 91238891724837 Ano de fundação: 2021 Estado: Aprovado

globe **Localização**

País: África do Sul Cidade: Murtede Morada: Rua dos Olivais

envelope **Contactos**

Email geral: ritomealmeida@gmail.com Telefone: 965360269 Website: glacier.systems

user **Responsável**

Nome: Tomé Almeida Email: ritomealmeida@gmail.com Telefone: 965360269

✓ Validar ✗ Remover

Figura 143 - Exemplo de cartão de registo de empresa

Integração com Resend para Envio de Emails

O que é o Resend?

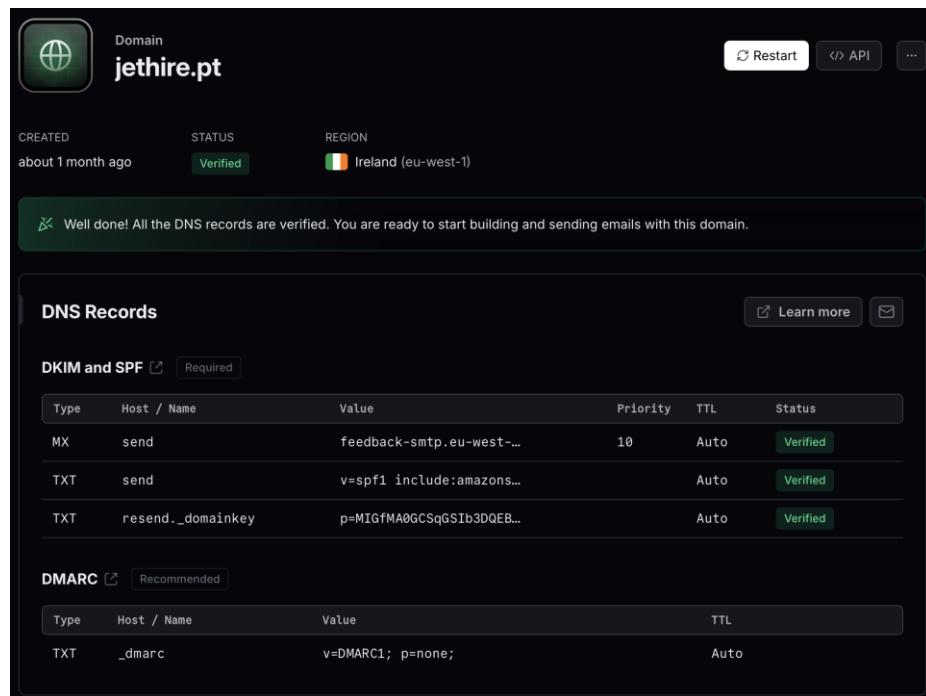
O **Resend** é uma plataforma especializada no envio de **emails transacionais**, especialmente desenhada para programadores. Foi utilizada neste projeto para garantir um sistema de envio de emails **automático e fiável**, essencial na fase de ativação de empresas.

Preparação

Para utilizar o Resend foi necessário:

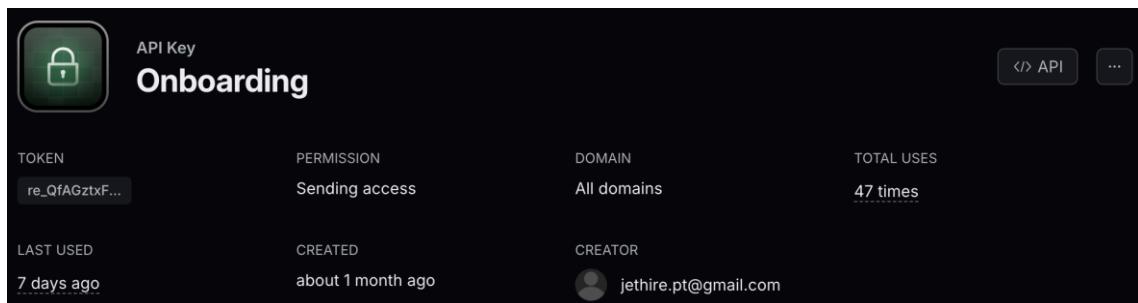
- Criar uma conta
- Associar o domínio da aplicação à conta Resend
- Inserir a **API Key** no ficheiro .env da aplicação

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



The screenshot shows the Resend platform interface for domain verification. At the top, it displays the domain **jethire.pt** with status **Verified**. Below this, it shows creation details: **CREATED about 1 month ago**, **STATUS Verified**, and **REGION Ireland (eu-west-1)**. A green message box states: "Well done! All the DNS records are verified. You are ready to start building and sending emails with this domain." The interface includes sections for **DNS Records** and **DKIM and SPF** (Required), listing MX, TXT, and DMARC records with their respective values and statuses. It also includes a **Learn more** button and an envelope icon.

Figura 144 - Domínio verificado no Resend



The screenshot shows the Resend API key onboarding interface. It features a lock icon and the title **API Key Onboarding**. It displays the following information:

TOKEN	PERMISSION	DOMAIN	TOTAL USES
re_QfAGztxF...	Sending access	All domains	47 times

Below this, it shows:

LAST USED	CREATED	CREATOR
7 days ago	about 1 month ago	jethire.pt@gmail.com

Figura 145 - API key do Resend

Personalização de Emails

Com a ajuda da biblioteca `@react-email/components`, foi criado um **template de email visualmente profissional**, com:

- **Logótipo da aplicação**
- **Mensagem personalizada**
- **Token de ativação**

Link direto para a página de registo

Este email é enviado automaticamente assim que o administrador valida o pedido de registo da empresa.

Exemplo de Email

O email apresenta um **token único** e uma mensagem que orienta o utilizador a usá-lo no registo da sua empresa. O conteúdo é apresentado com estilo profissional e personalizado.



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 146 - Exemplo de email de token de acesso

```
import {  
  Body,  
  Container,  
  Head,  
  Heading,  
  Html,  
  Img,  
  Link,  
  Preview,  
  Section,  
  Text,  
}
```

```
} from '@react-email/components';

interface TokenEmailProps {
  token?: string;
}

const baseUrl = "https://dashboard.jethire.pt";

export const TokenEmail = ({ token }: TokenEmailProps) => (
  <Html>
    <Head />
    <Body style={main}>
      <Preview>O seu token de acesso!</Preview>
      <Container style={container}>
        <Section style={logoContainer}>
          <Img
            src={`${baseUrl}/images/logo/logo.png`}
            width="120"
            height="36"
            alt="Jet Hire"
          />
        </Section>
        <Heading style={h1}>Aqui está o seu token!</Heading>
        <Text style={heroText}>
          A empresa que registou foi validada por um administrador. Utilize o seguinte
          token para a registar na nossa dashboard.
        </Text>
        <Section style={codeBox}>
          <Text style={confirmationCodeText}>{token}</Text>
        </Section>
        <Section>
          <Text style={text}>
```

```

Pode utilizar o token neste link:{' '}

<Link style={inlineLink} href="https://dashboard.jethire.pt/signup/">
    https://dashboard.jethire.pt/signup/
</Link>
</Text>
</Section>

<Text style={text}>
    Se não tentou registar nenhuma empresa, pode ignorar este email ou contactar-nos para perceber o que aconteceu.
</Text>

<Section>
    <Link
        style={footerLink}
        href="https://jethire.pt/"
        target="_blank"
        rel="noopener noreferrer"
    >
        Jet Hire
    </Link>
    <Text style={footerText}>
        ©2025 Jet Hire.
        Todos os direitos reservados.
    </Text>
</Section>
</Container>
</Body>
</Html>
);

TokenEmail.PreviewProps = {
    token: 'DJZ-TLX',
} as TokenEmailProps;

```

```
export default TokenEmail;

const footerText = {
  fontSize: '12px',
  color: '#b7b7b7',
  lineHeight: '15px',
  textAlign: 'left' as const,
  marginBottom: '50px',
};

const footerLink = {
  color: '#b7b7b7',
};

const main = {
  backgroundColor: '#ffffff',
  margin: '0 auto',
  fontFamily:
    "-apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen', 'Ubuntu',
    'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue', sans-serif",
};

const container = {
  margin: '0 auto',
  padding: '0px 20px',
};

const logoContainer = {
  marginTop: '32px',
};

const h1 = {
  color: '#1d1c1d',
```

```
fontSize: '36px',
fontWeight: '700',
margin: '30px 0',
padding: '0',
lineHeight: '42px',
};

const heroText = {
  fontSize: '20px',
  lineHeight: '28px',
  marginBottom: '30px',
};

const codeBox = {
  background: 'rgb(245, 244, 245)',
  borderRadius: '4px',
  marginBottom: '30px',
  padding: '40px 10px',
};

const confirmationCodeText = {
  fontSize: '30px',
  textAlign: 'center' as const,
  verticalAlign: 'middle',
};

const text = {
  color: '#000',
  fontSize: '14px',
  lineHeight: '24px',
};

const inlineLink = {
```

```
fontSize: '14px',
color: '#0a84ff',
textDecoration: 'underline',
};
```

Envio de Email com Token

Na API responsável pela ativação, é gerado um **token aleatório**, que é encriptado com bcrypt e guardado na base de dados.

Depois, o email é enviado para o responsável da empresa, utilizando o componente `TokenEmail`, com o token visível no corpo da mensagem.

Este processo automatiza a comunicação entre a aplicação e o utilizador, tornando o processo de validação **rápido, seguro e fiável**.

```
import { Resend } from "resend";
import TokenEmail from "@/emails/TokenEmail";

const resend = new Resend(process.env.RESEND_API_KEY);

export async function PATCH(req: NextRequest) {
  try {

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

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

{ }

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.



Registo de empresa
Introduza o seu token de acesso e defina a sua palavra-passe.

Token de acesso*
Introduza aqui o token

Palavra-passe*
Introduza a palavra-passe

Confirmar palavra-passe*
Confirme a palavra-passe

Ao verificar este caixa significa que concordo com os [Termos e Condições](#), e o [nosso Políaco de Privacidade](#).

Registrar empresa

Já registou a sua empresa? [Fazer login](#)

jethire
Procura emprego, contrata talentos.

Figura 147 - Interface de registo de empresa

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 148 - Exemplo de email de registo de empresa

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)	
	Nome		Email	Função	Estado
	Emanuel Oliveira		emanuel.oliveira@glacier.pt	Recrutador	
	Tomé Almeida		ritomealmeida@gmail.com	Recrutador	
	Teste		emanuel.oliveira@glacier.pt	Recrutador	

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.

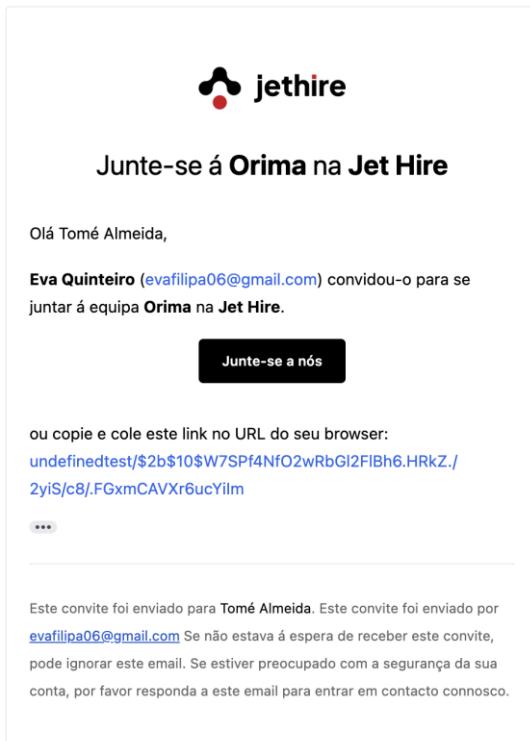
Envio do Convite por Email

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.



Criação de conta de convidado

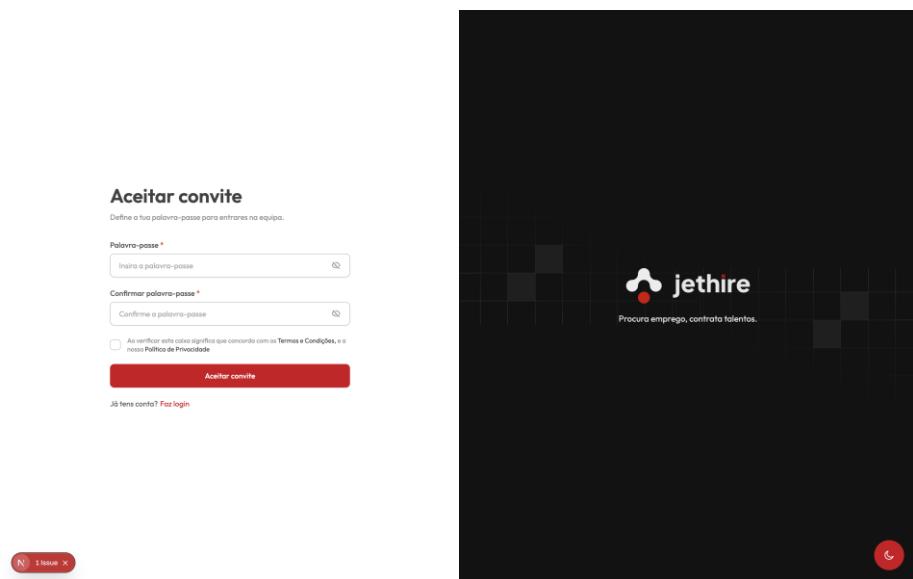
Depois de o convite ser enviado por email, o utilizador convidado recebe um link com um **token de acesso exclusivo**. Este link leva-o diretamente para uma página de registo simplificada, específica para **aceitação de convites**.

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.



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.



Bem-vindo à Jet Hire, Orima!

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

Utilizador: orima-tomealmeida

Palavra-passe: 1234

Aceda à sua conta através do seguinte link:

<https://dashboard.jethire.pt/signin>

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.

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.



Eva Quinteiro
Administrador

Email: evafilipa06@gmail.com
Telefone: 965360269
Cidade: Coimbra
País: Portugal

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:

Secção	Conteúdo
Logótipo	Upload da imagem representativa da empresa
Foto de fundo	Imagen de capa utilizada no perfil público
Informações básicas	Nome, slogan, 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:

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



Figura 149 - UserMetaCard

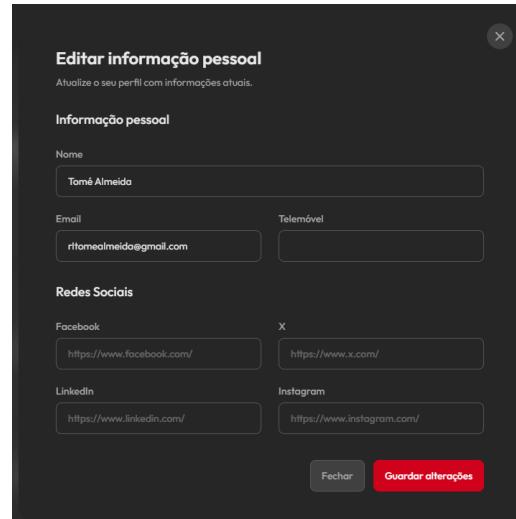


Figura 150 - UserInfoCard



Figura 151 - 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:

1. A imagem é convertida para Buffer
2. Se já existir uma imagem antiga, é removida da **Cloudinary**
3. A nova imagem é enviada para a **Cloudinary**
4. 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:

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)
Fundada em	Ano de fundação
Cidade / País	Localização geográfica
Endereço	Morada física da empresa
Email	Email institucional
Telefone	Número de contacto
Website	Link externo da empresa



Figura 152 - Cartão de apresentação de empresas

Carregamento de dados

O componente CompanyList faz o carregamento assíncrono dos dados logo ao montar a página:

```
useEffect(() => {
  const fetchCompanies = async () => {
    const res = await fetch("/api/company/list");
    const data = await res.json();
    setCompanies(data);
  };
  fetchCompanies();
}, []);
```

Durante o carregamento é apresentada uma **mensagem de feedback com ícone animado**:

```
<Loader className="animate-spin" /> A carregar empresas ...
```

 A carregar empresas ...

Figura 153 - Feedback de carregamento de empresas

E caso a pesquisa não devolva resultados, o utilizador é informado:

<p>Nenhuma empresa encontrada.</p>

Conclusão

O desenvolvimento da Jet Hire foi uma experiência extremamente enriquecedora, tanto a nível técnico como pessoal.

Ao longo deste projeto, aprofundei os meus conhecimentos em tecnologias modernas como Next.js, MongoDB, Mongoose e autenticação com NextAuth, além de consolidar boas práticas de desenvolvimento.

Este projeto permitiu-me aplicar, de forma prática, os conteúdos aprendidos durante o curso, desafiando-me a encontrar soluções para problemas reais. Além disso, tive a oportunidade de explorar ferramentas como o Cloudinary e a criação de formulários dinâmicos, que contribuíram para a construção de uma plataforma moderna, intuitiva e funcional.

Acredito que esta aplicação poderá ser útil para jovens e empresas da área das Tecnologias de Informação e, se for desenvolvida e mantida futuramente, poderá mesmo ser disponibilizada como solução real no mercado.

Com este trabalho, concluo a minha Prova de Aptidão Profissional com orgulho no percurso que trilhei e motivação para continuar a evoluir na área da programação.

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/>

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>