

—
ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

P.PORTO

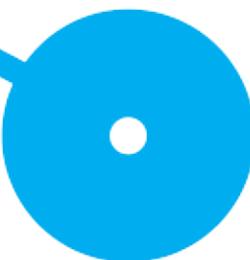
L —

LICENCIATURA
ENGENHARIA INFORMÁTICA

Plataforma de gestão de treinos para atletas de alta competição

Daniel Sousa

Março de 2021



[PÁGINA PROPOSITADAMENTE DEIXADA EM BRANCO]

—
ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

P.PORTO

L

—
LICENCIATURA
ENGENHARIA INFORMÁTICA

Plataforma de gestão de treinos para atletas de alta competição

Daniel Sousa

Professor Ricardo Jorge Santos

Este trabalho não inclui as críticas e sugestões feitas pelo Júri

[PÁGINA PROPOSITADAMENTE DEIXADA EM BRANCO]

Conteúdo

Lista de Figuras	viii
Lista de Tabelas	ix
Lista de Excertos de Código	xi
Abreviaturas	xii
Glossário	xiv
Resumo	1
Agradecimentos	2
Apresentação do Autor	3
Apresentação da Entidade de Acolhimento	4
Convenções e Nomenclatura	5
1 Contextualização e Motivação	7
1.1 Contextualização	7
1.2 Objetivos	7
1.3 Resultados	8
1.4 Organização do Documento	8
2 Ferramentas & Tecnologias	10

2.1	Enquadramento Tecnológico	10
2.1.1	TypeScript	10
2.1.2	React	12
2.1.3	Sass	12
2.1.4	NodeJS	13
2.1.5	GraphQL	14
2.1.6	PostgreSQL	15
2.2	Ambiente de Desenvolvimento	16
2.2.1	IDE	16
2.2.1.1	Visual Studio Code	16
2.2.1.2	WebStorm	16
2.2.2	Prettier – formatação de código	17
2.2.3	Gestor de Pacotes	17
2.3	Controlo de Versões	18
2.4	Metodologia	19
2.4.1	GitLab	19
2.4.2	Jira	19
2.4.3	ClickUp	20
3	Visão geral do projeto	22
3.1	Perspetiva do Produto	22
3.2	Perspetiva do Utilizador	22
3.3	Pressupostos de Restrições	23
3.4	Dependências	24
4	Design & Implementação da Solução	25
4.1	Arquitetura Conceptual	25
4.2	Diagramas de Sequência	26

4.2.1	Gestão de Sessão	26
4.2.2	Início de Sessão	27
4.2.3	Atleta – criação de conta	29
4.2.4	Lista de Atletas	32
4.3	Estrutura de Pastas	35
5	Resultados	37
5.1	<i>Backoffice</i>	37
5.2	<i>Frontoffice</i>	39
6	Conclusão	43
6.1	Formações Adicionais	43
6.2	Trabalhos Futuros	44
6.2.1	Biblioteca de Componentes	45
6.2.2	Testes	45
6.2.2.1	<i>Unit Testing</i>	45
6.2.2.2	<i>End-to-end Testing</i>	47
Referências Bibliográficas		48
Anexos		53
TypeScript		53
Instalação		53
Configuração		54
React		56
Criação do Projeto		56
Opções Adicionais		58
Estrutura de Pastas		58
Ficheiros Iniciais		60

Execução do Projeto	61
Componentes Genéricos Desenvolvidos	62
<i>TextInput</i>	62
<i>Button</i>	64
<i>Avatar</i>	67
<i>SearchInput</i>	68
Sass	71
Instalação	71
Mixins	71
Herança	72
Variáveis	74
GraphQL	75
Instalação	75
Apollo Client	75
Criação de um <i>client</i>	76
Execução de <i>queries</i>	76
Comparação entre GraphQL e REST	77
Yarn	79
Instalação	79
Linux/Windows/macOS	79
macOS	79
Linux	79
NodeJs	81
Instalação	81
macOS	81
Linux	81
Windows	81

Arquitetura	82
PostgreSQL	83
Instalação	83
Container Docker	83
Windows	83
macOS	83
Linux	84
Testes	85
Visual Studio Code	89
Configurações	89
Extensões	91
Prettier	93
Configuração	93
Husky e Git hooks	93

Lista de Figuras

1	Jimmy Boys – Icon	4
2	TypeScript – logo	10
3	React – logo	12
4	Sass – logo	12
5	Ficheiro <code>.sass</code> compilado para um ficheiro <code>.css</code>	13
6	NodeJS – logo	13
7	GraphQL – logo	14
8	Demonstração do GraphiQL	14
9	PostgreSQL – logo	15
10	Visual Studio Code – logo	16
11	WebStorm – logo	16
12	Prettier – logo	17
13	Workflow seguido no GitLab durante o desenvolvimento	18
14	Board utilizada no GitLab	19
15	Board utilizada no Jira	20
16	Vista em board no ClickUp	21
17	Exemplo de utilização por parte dos atletas e treinadores	23
18	Arquitetura Conecptual do projeto	25
19	Diagrama de Sequência de Gestão de Sessão	26
20	Diagrama de Sequência para o inicio de sessão	27

21	Mockup do ecrã de inicio de sessão – backoffice	28
22	Mockup do ecrã de inicio de sessão com redes sociais – frontoffice	28
23	Mockup do ecrã de inicio de sessão – frontoffice	28
24	Diagrama de Sequência para a criação de conta de um atleta	29
25	Mockup do ecrã de criação de conta – frontoffice	30
26	Mockup do ecrã de criação de conta com dados inválidos – frontoffice	30
27	Mockup de informações de registo – frontoffice	31
28	Inputs de carregamento de imagens	31
29	Componente Tooltip	31
30	Imagens carregadas para o perfil do atleta – frontoffice	31
31	Diagram de Sequência para a consulta da lista de atletas	32
32	Lista em mosaico dos atletas – backoffice	32
33	Lista dos atletas – backoffice	33
34	React – estrutura de pastas utilizada	36
35	Backoffice: pasta components	38
36	Backoffice: pasta controllers	38
37	Backoffice: vista em mosaico dos atletas	39
38	Backoffice: vista em lista dos atletas	39
39	Backoffice: página de perfil do atleta	39
40	Frontoffice: pasta components	40
41	Frontoffice: pasta controllers	40
42	Frontoffice: criação de conta do atleta	41
43	Frontoffice: finalização da criação de conta	41
44	Frontoffice: passo inicial do <i>wizard</i>	41
45	Frontoffice: informações pessoais e de pagamento do <i>wizard</i>	41
46	Frontoffice: resumo do <i>wizard</i>	42
47	Frontoffice: <i>popup</i> apresentado no final do <i>wizard</i>	42

48	Jest – cobertura de código	46
49	Cypress – resultado do teste apresentado	47
50	Página inicial do React após a execução do projeto	58
51	React – possível estrutura de pastas	59
52	React – estrutura de pastas e ficheiros gerados pelo <code>create-react-app</code>	59
53	Projeto React executado com sucesso	62
54	Componente Avatar na página de perfil do atleta	67
55	Componente Avatar na vista em mosaico na lista de atletas	67
56	Componente Avatar na lista de atletas	67
57	Aparência do componente SearchInput	68
58	Comparação entre GraphQL e REST	77
59	Ecrã inicial da instalação do NodeJS no Windows	81
60	Arquitetura do NodeJS em comparação com a arquitetura tradicional	82
61	Execução dos testes com sucesso	86
62	Execução dos testes com erro no componente Status Badge	86
63	Extensão ES7 React/Redux/GraphQL/React-Native snippets	92
64	Extensão Auto Import	92
65	Extensão Auto Close Tag	92
66	Extensão Auto Rename Tag	92
67	Extensão ESLint	92
68	Extensão Sass	92

Lista de Tabelas

1	Principais diferenças entre TypeScript e JavaScript	11
2	Comparação entre Yarn e NPM	17

Lista de Excertos de Código

1	Demonstração de excerto de código	6
2	Exerto de código com validação TypeScript	11
3	GraphQL – Exemplo de <i>query</i>	15
4	GraphQL – Exemplo de resposta à <i>query</i> realizada	15
5	Exemplo de pedido com o package Axios e autenticação por HTTP Headers	27
6	Função para renderizar a lista de atletas	34
7	Função para renderizar o mosaico de atletas	34
8	Código da página de listagem dos atletas existentes	35
9	Jest – exemplo de um teste para um componente React	46
10	Cypress – exemplo de teste	47
11	TypeScript – Ficheiro <code>tsconfig.json</code>	55
12	Scripts para a execução do projeto em React	56
13	Ficheiro <code>index.html</code> de um projeto React	60
14	Ficheiro <code>index.jsx</code> de um projeto React	60
15	Ficheiro <code>index.css</code> de um projeto React	61
16	Ficheiro <code>app.jsx</code> de um projeto React	61
17	Propriedades recebidas no componente TextInput	63
18	Propriedade rounded aplicada ao input	63
19	Estilo adicionado na class da propriedade rounded	64
20	Utilização da propriedade multiline no componente TextInput	64
21	Estilo aplicado no uso da propriedade modifier com o valor <code>light</code>	64
22	Propriedades recebidas no componente Button	65
23	Uso de Link ou button no componente Button	65
24	Aplicação de classes adicionais para as propriedades rounded e modifier	65
25	Estilo aplicado para as propriedades modifier e rounded	67
26	Código desenvolvido para o componente Avatar	68
27	Código do componente SearchInput	69
28	Uso do componente SearchInput em conjunto com o package Fuse.js	70
29	Definição e uso de <i>mixins</i> no Sass	71
30	Demonstração de herança no Sass	72
31	Código CSS resultante da compilação do excerto de código anterior	72
32	Demonstração de <i>placeholders</i> em Sass	73
33	Código CSS resultante da compilação do excerto de código com <i>placeholder</i>	74

34	Utilização de variáveis em Sass	74
35	Código CSS resultante do excerto de código com variáveis em Sass	74
36	Declaração e uso de variáveis em CSS	74
37	Importação do GraphQL em JavaScript	75
38	Importação do GraphQL em TypeScript	75
39	Criação de um <i>client</i> recorrendo ao Apollo Client no React	76
40	Execução de uma query recorrendo ao Apollo Client no React	76
41	GraphQL – Exemplo de query com seleção de campos	78
42	Testes realizados ao componente Button	85
43	Testes realizados ao componente tatus Badge	86
44	Código com o data-testid definido no componente Status Badge	87
45	Teste com preenchimento de inputs	87
46	Configurações utilizadas no Visual Studio Code	91
47	Configurações utilizadas no Prettier	93
48	Exemplo do conteúdo do ficheiro .prettierignore	93
49	Configurações utilizadas no Husky , Lint Staged e Prettier	94

Abreviaturas

LEI Licenciatura em Engenharia Informática

CTeSP Curso Técnico Superior Profissional

ESTG Escola Superior de Tecnologia e Gestão

ES ECMAScript

JSON JavaScript Object Notation

YAML YAML Ain't Markup Language

SQL Structured Query Language

HTML Hyper Text Markup Language

CSS Cascading Style Sheets

Sass Syntactically Awesome Style Sheets

JS JavaScript

TS TypeScript

NoSQL No SQL –Not Only SQL

NVM Node Version Manager

NPM Node Package Manager

JWT JSON Web Token

UI User Interface

UX User Experience

HTTP HyperText Transfer Protocol

CDN Content Delivery Network

CMS Content Management System

CRUD Create, Read, Update, Delete

DRY Don't Repeat Yourself

DOM Document Object Model

MVC Model-View-Controller

REST Representational State Transfer

API Application Programming Interface

URL Uniform Resource Locator

DB Database

CI Continuous Integration

CD Continuous Delivery

IDE Integrated Development Environment

SVG Scalable Vector Graphics

CORS Cross-Origin Resource Sharing

SRP Single Responsibility Principle

RGPD Regulamento Geral sobre a Proteção de Dados

WIP Work In Progress

Glossário

Roles Forma de distinguir os diversos tipos de utilizadores de uma aplicação, contendo como tal diferentes tipos de permissões e ações possíveis de realizar

Responsive / Responsivo Conjunto de técnicas aplicadas a um *layout* de forma a este se adaptar a qualquer tamanho de ecrã, independentemente do dispositivo

Layout Forma como são organizadas ou distribuídas as diferentes partes de algo: *layout de armazém*, *layout do teclado*, por [Lexico](#)

Mockups Próotipo de um projeto ou dispositivo, tendo como principal objetivo representar as principais funcionalidades do projeto/dispositivo. Utilizado frequentemente em projetos de desenvolvimento web para obter feedback do cliente

Front-end Parte vocacionada ao utilizador final, focada na *interface* visualizada, bem como a interação com o sistema. Essencialmente são usadas as linguagens/tecnologias **HTML, CSS e JavaScript**

Back-end Parte vocacionada na implementação, lógica e regras de negócio, não contendo *interface*. Nesta componente podem ser utilizadas linguagens como:

- C#;
- PHP;
- Java;
- Python;
- ...

Lazy Loading Consiste na técnica de adiar o carregamento de determinado componente ou class até este ser necessário.

Sprite Consiste numa imagem que contêm múltiplas imagens, bastante utilizado para armazenar todos os ícones de uma aplicação num único ficheiro.

Packages Módulos ou pacotes do **NodeJS** disponibilizadas publicamente e que podem ser instalados e posteriormente utilizados no projeto.

PWA Ou Progressive Web App, são aplicações híbridas com a possibilidade de serem utilizadas num browser, mas também contam com a possibilidade de serem instaladas num smartphone, sendo removida toda a interface do browser, ou seja, barra de navegação, barra de favoritos, etc..

Template Modelo, normalmente constituído por múltiplos ficheiros prontos a ser utilizados ou a necessitarem de alterações mínimas para funcionar

Workflow Fluxo de trabalho ou automação de procedimentos de trabalho.

Snippet Uma snippet é um pedaço pequeno código reutilizável, sendo comum nos IDE's a criação de snippets para inserir pedaços de código utilizados com frequência.

Autocomplete Capacidade de auxiliar durante o processo de programação, recorrendo a sugestões de excertos de código frequentemente utilizados ou snippets existentes para determinada linguagem.

Resumo

No âmbito da unidade curricular **Projeto Final** e, ao longo dos últimos três meses, foi desenvolvida uma aplicação web destinada a treinadores e atletas de alta competição. Este projeto já se encontra em funcionamento, porém, surgiu a necessidade da criação de novas funcionalidades, bem como, um *layout* mais apelativo e moderno.

A principal necessidade do projeto é proporcionar tanto aos treinadores, como atletas, uma melhor experiência de utilização e interação com o projeto, facilitando assim a consulta e gestão de treinos, bem como o acompanhamento. Assim e, através de gráficos e *dashboards*, atletas e treinadores conseguem facilmente visualizar a evolução de um atleta, bem como as suas demais informações.

O projeto é constituído por uma arquitetura cliente-servidor, *back-end* e *front-end*, sendo que do lado do *back-end* é possível encontrar tecnologias e ferramentas como **KoaJS**, **PostgreSQL**, **TypeORM**, **Apollo Server**, **GraphQL**, entre outras. Já do lado do *front-end* é possível encontrar tecnologias e ferramentas como **React** e **Apollo Client**.

Em ambas as camadas desta arquitetura é possível encontrar o uso de **TypeScript**, existindo determinadas vantagens no seu uso como será possível analisar ao longo deste documento.

A camada do cliente, ou seja, o *front-end*, é separada em duas vertentes, a vertente de *backoffice* - sendo esta destinada aos treinadores e administrador(es) - e a vertente de *frontoffice* destinada aos atletas. Ao longo do documento será possível encontrar em mais detalhe as funcionalidades presentes em cada vertente e respetivo *layout*.

No final e, apesar do projeto não se encontrar concluído, existem páginas de cada vertente funcional prontas a comunicar com o *back-end* do projeto e a realizar todas as ações necessárias. Sendo assim, atingidos os objetivos inicialmente propostos para o projeto.

Por fim, importa referir que, apesar do projeto contar com a camada do lado do servidor - o *back-end* - o foco do projeto é a camada do lado do cliente, o *front-end*.

Palavras-chave: React, Desenvolvimento Web, Front-end, BeAPT

Agradecimentos

“Life is a mirror: if you frown at it, it frowns back; if you smile, it returns the greeting.”

William Makepeace Thackeray

Em primeiro lugar, agradeço à **Escola Superior de Tecnologia e Gestão** e todos os seus docentes, pelo conhecimento transmitido durante a **Licenciatura em Engenharia Informática**. Agradeço em especial ao professor Ricardo Jorge Santos pela prontidão e tempo despendido na orientação e acompanhamento do projeto, contribuindo para produzir um trabalho com melhor qualidade.

Agradeço a oportunidade e a experiência proporcionada pela empresa **Jimmy Boys**, em especial aos engenheiros **Edgar Esteves** e **Pedro Souto** por todo o conhecimento transmitido e pela disponibilidade demonstrada.

À minha família que mais do que apoio, teve uma papel preponderante em todo o percurso académico essencial para a minha aprendizagem e desenvolvimento com vista a traçar um futuro melhor, o objetivo final de quem me rodeia.

Como não podia deixar de ser, agradeço a todos os amigos e colegas que de uma forma ou de outra contribuíram ao longo deste percurso.

Muito obrigado!

Apresentação do Autor

Daniel Sousa, nasceu a 12 de dezembro de 1995, em Massarelos, no distrito do Porto. A quando a redação deste documento encontra-se matriculado no terceiro ano da **Licenciatura em Engenharia Informática da Escola Superior de Tecnologia e Gestão**, em Felgueiras, estando a realizar estágio académico na empresa **Jimmy Boys**.

Desde cedo interessado pelo mundo da tecnologia, realizou um curso profissional em **Técnico de Gestão de Equipamentos Informáticos**(nível IV do **Quadro Nacional de Qualificações**), realizando o seu primeiro contacto com a criação de websites no projeto de aptidão profissional, recorrendo para tal ao [CMS Joomla](#).

Mais tarde frequentou ainda um **CTeSP** em **Informática de Gestão** (nível V do **Quadro Nacional de Qualificações**), onde através do estágio académico realizado, encontrou a paixão pelo desenvolvimento em ambiente web. Durante a realização deste estágio surgiu a necessidade de aquisição de novas competências em tecnologias como **PHP, HTML, CSS e MySQL**.

Assim sendo, ao participar no projeto em questão, além da aquisição de novas competências na área de sua preferência, conseguiu ainda melhorar conhecimentos previamente adquiridos.

Apresentação da Entidade de Acolhimento

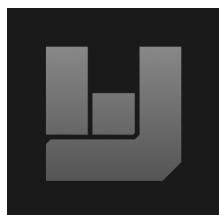


Figura 1: Jimmy Boys – Icon

A **Jimmy Boys** é uma empresa que opera no ramo do desenvolvimento de software desde 2012. A **Jimmy Boys** desenvolve tanto os próprios softwares, bem como em outsourcing para outras empresas.

A **Jimmy Boys** opera tanto em front-end, back-end e mobile, realizando projetos nas mais diversas tecnologias, como **React**, **GraphQL**, **Rust**, **Flutter**, entre outras. Além destas tecnologias, realiza ainda projetos de UI e UX.

“Along the way, we have been working with different technologies and different business needs. This helped us grow and prepared us for more demanding projects.”

Jimmy Boys

Outsourcing consiste na contratação de recursos a outra empresa. Por exemplo, quando uma empresa não possui um departamento de marketing, recorre a uma empresa desta área para realizar esse serviço em nome desta empresa.

Ao trabalhar em outsourcing, a **Jimmy Boys** além de disponibilizar os seus colaboradores para a realização do projeto em questão, promove ainda workshops dentro da empresa, com o objetivo de integrar a equipa da empresa no projeto, explorando temas como boas práticas no desenvolvimento ou, como criar um projeto em determinada tecnologia.

Abaixo seguem as principais ligações da empresa.

- [LinkedIn](#);
- [Website](#)

Convenções e Nomenclatura

Ao longo deste relatório, optou-se por seguir um conjunto de convenções de forma a facilitar a interpretação do texto, exemplos e excertos de código apresentados.

Desta forma textos em **itálico** terão como objetivo representar estrangeirismos, já textos em **negrito** terão como objetivo realçar termos com maior relevância ou mesmo nomes de empresas, marcas, etc.. Em casos de textos sublinhados, por norma, referem-se a ligações no documento, por exemplo a ligação para uma determinada definição no glossário, para ligações a websites externos é utilizada a cor **azul**.

Contudo e, sempre que seja pertinente realçar uma determinada nota, será utilizado o formato que é apresentado de seguida.

Nota

Informação da nota

Porém e, recorrendo ao esquema anterior, sempre que seja necessário apresentar informações sobre um erro que poderá ocorrer ou que ocorreu, será utilizado o formato apresentado abaixo.

⚠ Erro Apresentado

Mensagem ou informações sobre o erro.

Sempre que seja pertinente adicionar determinada citação, será utilizado o formato apresentado abaixo.

“Citação”

Autor ou Referência da citação

No caso de excertos de código e, de forma a manter a syntax o mais correta possível, será utilizado o formato apresentado abaixo, sendo possível visualizar os números das linhas, bem como, caso seja pertinente, destacar alguma destas linhas.

¹ // Exemplo de Excerto de Código

```
2 console.log("Hello World");
```

Exerto de Código 1: Demonstração de excerto de código

No que toca a nomenclatura e, tal como será possível analisar ao longo deste documento, são seguidas as seguintes regras:

- **Componentes React:** nomes em **Pascal Case**, ou seja, a primeira letra do identificador e a primeira letra de cada palavra são escritas em maiúsculas;
- **Interfaces:** seguem novamente o *naming convention* **Pascal Case** e começam pela letra **I**, que representa interface;

Capítulo 1

Contextualização e Motivação

“Creativity is just connecting things.”

Steve Jobs

1.1 Contextualização

O projeto **BeApt**, ou **Be Armada Portuguesa do Trail**, tem como objetivo de preparar atletas para desafios de alta competição, tal como ultramaratonas, ultra-trails, triatlos, entre outros. Desta forma, é necessário o desenvolvimento de uma aplicação web, permitindo a sua utilização em qualquer lugar e dispositivo, que permita tanto a atletas como treinadores gerir, visualizar e analisar resultados de uma forma simples e eficaz.

Para tal, foi idealizada uma aplicação web composta por duas vertentes, a vertente de *backoffice*, destinada a treinadores e administradores da plataforma e, a vertente de *frontoffice*, destinada aos atletas, onde estes conseguirão ver os treinos que lhe foram atribuídos, bem como registar resultados desses mesmos treinos.

Um dos pontos com grande importância no projeto é a criação de gráficos e dashboards de forma a permitir que tanto atletas como treinadores consigam facilmente analisar os valores registados.

1.2 Objetivos

Os principais objetivos do projeto consistem na criação de uma aplicação web, que seja facilmente utilizada, onde atletas e treinadores possam interagir. Para tal, é necessário que a aplicação permita:

- Criação de treinos e blocos de treino modelo, para posteriormente serem atribuídos;

- A atribuição de treinos a atletas, adaptando determinados valores aos seus parâmetros biométricos;
- A consulta de treinos atribuídos e realizados pelo atleta;
- A visualização da evolução do atleta nas mais diversas fases;
- A consulta e edição dos dados do atleta;

Para facilitar o acesso por ambos os utilizadores, atletas e treinadores, o projeto é separado em duas vertentes, o **backoffice** – destinado ao administrador e treinadores e, o **frontoffice** – destinado aos atletas.

1.3 Resultados

A primeira interação com o projeto começou com a criação de componentes **React**, porém utilizando **JavaScript**, o que pouco tempo depois, levou à migração de todo o código já produzido para **TypeScript**. O **TypeScript** é atualmente quase que um requisito obrigatório, tendo também a sua parte de garantia de qualidade do código, visto que os erros são mais facilmente detectados devido à tipagem necessária.

Posteriormente, após a migração de ambas as vertentes (*backoffice* e *frontoffice*) foram realizados em primeiro lugar os componentes com mais utilização no projeto¹, componentes estes que numa interação futura seriam colocados numa biblioteca de componentes uma vez que são utilizados em ambas as vertentes. Após estes componentes criados, o processo de desenvolvimento passou por criar páginas, nesta fase com dados estáticos, que por sua vez, originavam a criação de outros componentes.

Uma das grandes aprendizagens na realização deste projeto foi o uso de sprites **SVG** para armazenar todos os ícones necessários para o projeto, sendo criado um componente **React** que iria aceder a cada ícone colocado nesta sprite através do **id** a este associado.

No capítulo referente aos resultados é possível analisar com mais detalhe todo o desenvolvimento realizado, apresentando figuras relativas ao mesmo, bem como os componentes **React** desenvolvidos para as necessidades que cada página apresentava.

1.4 Organização do Documento

Este documento encontra-se organizado em vários capítulos, de forma a facilitar a leitura do mesmo. Desta forma é possível encontrar os seguintes capítulos:

¹Ver anexos (páginas 62 a 70).

- **Capítulo 1 – Contextualização e Motivação:** no primeiro capítulo é possível encontrar uma breve introdução ao projeto, sendo apresentados os principais objetivos e como se encontra organizado este documento, apresentando os principais tópicos de cada capítulo;
- **Capítulo 2 – Fundamentação Teórica:** neste capítulo são abordadas as tecnologias e ferramentas utilizadas no decorrer do projeto, bem como a metodologia utilizada e informações relativas ao controlo de versões;
- **Capítulo 3 – Visão geral do projeto:** na visão geral do projeto é possível conhecer melhor o projeto, conhecer os objetivos, dependências e quem são os utilizadores do projeto;
- **Capítulo 4 – Design & Implementação da solução:** no que toca ao design e implementação da solução é possível encontrar essencialmente diagramas referentes ao mesmo, começando pelo diagrama da arquitetura conceptual, passando aos diagramas de sequência onde é apresentado de forma geral o fluxo da aplicação em determinados ecrãs do mesmo. No tópico referente aos diagramas de sequência são também apresentados os componentes **React** com mais importância, apresentando as partes importantes da sua implementação. Por fim, é apresentada a estrutura de pastas utilizada em ambas as vertentes do projeto (o *backoffice* e *frontoffice*);
- **Capítulo 5 – Resultados:** como o nome deste capítulo indica, serão apresentados os resultados obtidos referentes ao desenvolvimento do projeto, bem como algumas reflexões sobre o mesmo;
- **Capítulo 6 – Conclusão:** neste capítulo é possível encontrar uma conclusão global referente ao projeto, algumas das formações realizadas de forma a conseguir melhorar a prestação durante a realização do projeto, bem como os trabalhos a serem implementados futuramente.

Importante referir que em anexos é possível encontrar informações adicionais relativas às tecnologias utilizadas, bem como alguns guias referentes às mesmas.

Capítulo 2

Ferramentas & Tecnologias

O desenvolvimento de soluções web continua em expansão e, cada vez mais, surgem novas frameworks e bibliotecas para auxiliar no processo de desenvolvimento, quer para front-end como back-end.

Assim sendo, nesta secção é possível encontrar informações relacionadas com as tecnologias utilizadas, bem como as demais ferramentas utilizadas durante o processo de desenvolvimento. Além destes pontos é também possível encontrar informações relacionadas com a metodologia utilizada.

2.1 Enquadramento Tecnológico

Neste projeto foram utilizadas tecnologias tanto do lado do cliente, front-end, como do lado do servidor, back-end, apesar que o foco deste relatório é o lado do cliente (front-end) é necessário referir que este irá comunicar com o lado do servidor (back-end), onde estão armazenadas todas as informações da aplicação.

2.1.1 TypeScript



Figura 2:
TypeScript
– logo

O **TypeScript** é uma das tecnologias que é possível encontrar neste projeto tanto em front-end como back-end.

O **TypeScript**, segundo a própria **Microsoft** (detentora do **TypeScript**), é nada mais nada menos do que **JavaScript**, porém com a adição de tipos.

"TypeScript extends JavaScript by adding types."

Retirado do [website oficial](#)

Em 2020, o **TypeScript** ficou em segundo lugar das linguagens preferidas e, em quarto lugar das linguagens mais procuradas, dados do [StackOverflow](#).

Devido a esta tipagem que é adicionada, o código torna-se mais facilmente interpretado, facilitando também o processo de debug, bem como as validações realizadas no processo de *build*. O excerto de código abaixo, retirado do [website oficial](#), tem como objetivo demonstrar a validação que é realizada pelo **TypeScript**.

```
1 const user = {  
2   firstName: "Angela",  
3   lastName: "Davis",  
4   role: "Professor"  
5 }  
6  
7 console.log(user.name)
```

Excerto de Código 2: Excerto de código com validação **TypeScript**

No caso, a linha 7 (assinalada com a cor vermelha), irá causar a mensagem de erro abaixo que indica que a propriedade **name** não existe no objeto **user**.

Erro Apresentado

Property '**name**' does not exist on type '{ **firstName**: string; **lastName**: string; **role**: string; }'.

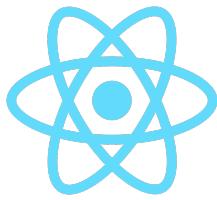
A tabela apresentada demonstra as principais diferenças entre o **TypeScript** e o **JavaScript**.

TypeScript	JavaScript
Linguagem orientada a objetos	Linguagem de Scripting
Possui tipagem estática	Não possui tipagem
Suporte a módulos	Sem suporte a módulos
Possui suporte a definição de parâmetros opcionais em funções	Não suporta a definição de parâmetros opcionais em funções

Tabela 1: Principais diferenças entre **TypeScript** e **JavaScript**

Em [anexo \(página 53 a 55\)](#) é possível encontrar como realizar a instalação do **TypeScript** e ainda, um exemplo de uma configuração realizada através do ficheiro **tsconfig.json**.

2.1.2 React



Existe quem considere que o **React** é uma *framework* de **JavaScrip**t, porém e, ao mesmo tempo, há quem a considere como uma biblioteca de **JavaScript** baseada em componentes, sendo este o termo correto.

Os principais objetivos desta biblioteca são essencialmente:

Figura 3:

React

– logo

- Fácil Aprendizagem;
- Rapidez;
- Escalável.

Importante referir que em 2020, segundo o [StackOverflow](#), o **React** ficou em segundo lugar nas frameworks preferidas dos programadores e em primeiro lugar nas mais procuradas.

Em anexo (páginas 56 a 70) é possível encontrar todas as instruções relativas à criação de um projeto, bem como estrutura de pastas e execução de um projeto **React**.

2.1.3 Sass



Figura 4:

Sass – logo

O **Sass**, ou *Syntactically Awesome Style Sheets* é um preprocessor de **CSS** possuindo duas variantes:

- **.sass** – não necessita de ; nem {}, apenas que o código esteja corretamente indentado;
- **.scss** – esta variante necessita de ; e {}, bem como a correta indentação do código.

Durante a realização deste projeto será utilizada a variante sem ; e {}, sendo apresentados os principais detalhes da mesma.

A figura apresentada de seguida representa a transformação que é realizada após a compilação de um código **Sass** (**.sass**) em código **CSS** (**.css**), onde é possível analisar a declaração de uma variável (`$darkColor`), bem como o seu uso. Sendo ainda possível analisar o suporte a *nesting*, ou seja, **CSS** dentro de **CSS** (de uma forma simplificada), algo que não é suportado em **CSS**.

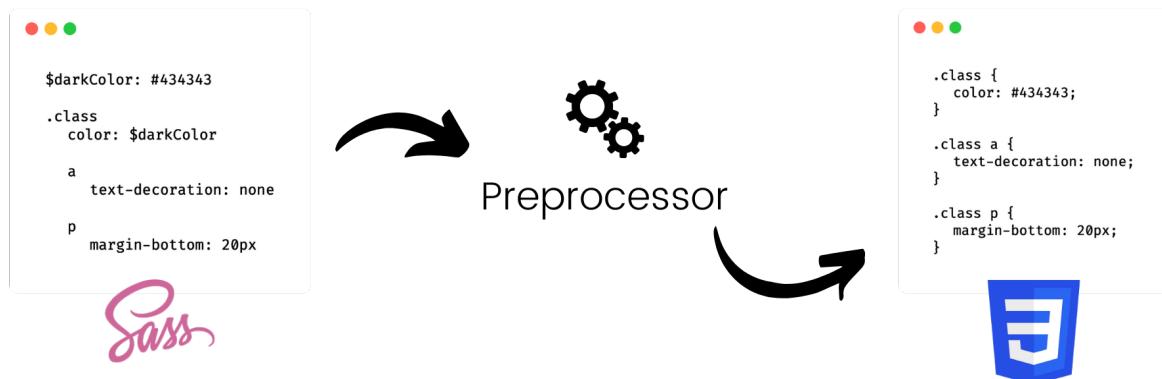


Figura 5: Ficheiro `.sass` compilado para um ficheiro `.css`

Em anexo (página 71 a 74) é possível encontrar algumas vantagens do uso de **Sass**, recorrendo para tal a exemplos práticos.

2.1.4 NodeJS



Figura 6:
NodeJS
– logo

NodeJS é um ambiente de execução **JavaScript**, *open source*, que permite desenvolver aplicações do lado do servidor (back-end). Desta forma é possível criar aplicações utilizando **JavaScript** que não necessitam de um *browser* para a sua execução.

A performance do **NodeJS** deve-se essencialmente ao uso do interpretador **V8 da Google**, interpretador este que é o core do **Google Chrome**.

Em anexo (página 82) é possível encontrar a imagem que representa a arquitetura do **NodeJS** em comparação com a arquitetura tradicional².

Desta forma é possível analisar que no caso da arquitetura tradicional é criada uma nova *thread* para cada pedido, já no **NodeJS**, apenas existe uma *thread* que possui I/O não bloqueante, permitindo várias requisições simultâneas, ficando retidas no *event-loop*.

Em 2020, segundo dados do **StackOverflow**, o **NodeJS** ficou em sétimo lugar na lista de outras frameworks, bibliotecas ou ferramentas preferidas dos programadores e, em primeiro lugar na lista de outras frameworks, bibliotecas ou ferramentas mais procuradas.

Além disso, é ainda possível encontrar em anexo (página 81) como realizar a instalação do **NodeJS** nos diversos sistemas operativos.

²Imagem retirada de [7]

2.1.5 GraphQL

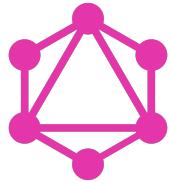


Figura 7:
GraphQL
— logo

GraphQL é uma query language open source criada pelo **Facebook** tendo como principais objetivos tornar as API's mais rápidas, flexíveis e intuitivas. Além disso o **GraphQL** traz consigo um IDE, chamado **GraphiQL**, que permite testar queries e analisar o seu resultado no próprio browser.

A imagem apresentada abaixo demonstra a utilização do **GraphiQL**, onde do lado esquerdo são apresentadas as queries e do lado direito o resultado das mesmas.

```

1 fragment PeopleInfo on PlanetResidentsConnection {
2   residents {
3     name
4     birthYear
5     hairColor
6   }
7 }
8
9 fragment PlanetInfo on Planet {
10   id
11   name
12   climates
13   gravity
14   population
15   residentConnection {
16     ...PeopleInfo
17   }
18 }
19
20 query Planets($firstN: Int) {
21   allPlanets(first: $firstN) {
22     totalCount
23     planets {
24       ...PlanetInfo
25     }
26   }
27 }
28
  
```

QUERY VARIABLES

```

1 {
2   "firstN": 2
3 }
  
```

RESULTADO

```

{
  "data": {
    "allPlanets": {
      "totalCount": 61,
      "planets": [
        {
          "id": "cGxhbmV0czox",
          "name": "Tatooine",
          "climates": [
            "arid"
          ],
          "gravity": "1 standard",
          "population": 200000,
          "residentConnection": {
            "residents": [
              {
                "name": "Luke Skywalker",
                "birthYear": "19BBY",
                "hairColor": "blond"
              },
              {
                "name": "C-3PO",
                "birthYear": "112BBY",
                "hairColor": "n/a"
              },
              {
                "name": "Darth Vader",
                "birthYear": "41.9BBY",
                "hairColor": "none"
              },
              {
                "name": "Owen Lars",
                "birthYear": "52BBY",
                "hairColor": "brown, grey"
              },
              {
                "name": "Beru Whitesun Lars",
                "birthYear": "47BBY",
                "hairColor": "brown"
              },
              {
                "name": "R5-D4",
                "birthYear": "11BBY"
              }
            ]
          }
        }
      ]
    }
  }
  
```

Figura 8: Demonstração do **GraphiQL**

Uma das principais vantagens do **GraphQL** em comparação a um arquitetura REST é a capacidade de apenas requisitarem os dados que precisam num único pedido. Além disso, o **GraphQL** pode ser utilizado tanto em back-end como em front-end, recorrendo para tal ao **Apollo Server** para back-end e ao **Apollo Client** para front-end.

O excerto de código abaixo apresenta um exemplo de query retirada da documentação oficial, onde é possível analisar, de uma forma muito abstrata, que é pedido o nome do herói, bem como o nome dos seus amigos (**friends**).

```

1 {
2   hero {
3     name
  
```

```

4     # Queries can have comments!
5
6     friends {
7         name
8     }
9 }
```

Excerto de Código 3: GraphQL – Exemplo de query

Por sua vez, o excerto de código que se segue apresenta o resultado desta query, sendo este apresentado no formato de um objeto **JSON**, contendo a propriedade **data**, possuindo todos os resultados obtidos.

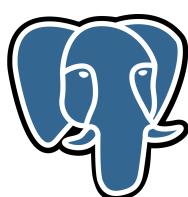
```

1  {
2      "data": {
3          "hero": {
4              "name": "R2-D2",
5              "friends": [
6                  {"name": "Luke Skywalker"},
7                  {"name": "Han Solo"},
8                  {"name": "Leia Organa"}
9              ]
10         }
11     }
12 }
```

Excerto de Código 4: GraphQL – Exemplo de resposta à query realizada

Em anexo (página 75 a 78) é possível encontrar como realizar a instalação do **GraphQL**. Além da instalação é possível encontrar exemplos da utilização do **Apollo Client**, visto o foco ser o front-end do projeto.

2.1.6 PostgreSQL



*Figura 9:
Post-
greSQL
– logo*

PostgreSQL é uma base de dados open source com boa reputação devido à sua flexibilidade e confiabilidade. O **PostgreSQL**, ao contrário de outros sistemas de gestão de base de dados relacionais, suporta tipos de dados relacionais como não relacionais.

Em 2020, segundo dados do [StackOverflow](#), o **PostgreSQL** ficou em segundo lugar das base de dados preferidas e mais procuradas dos programadores.

2.2 Ambiente de Desenvolvimento

No que toca ao ambiente de desenvolvimento este pode ser constituído por diversos *software* auxiliares, variando de programador para programador. Nos tópicos que se seguem são apresentadas algumas destas ferramentas, referindo que algumas será uma questão de gosto pessoal, podendo ou não ajudar a melhorar o workflow do programador.

Apesar de não ser apresentado nos tópicos que se seguem, é necessário um browser à escolha do programador/utilizador, uma vez que o projeto será utilizado para visualizar o projeto durante o desenvolvimento. Apesar de não existir nenhuma obrigatoriedade, não é recomendado o uso do **Internet Explorer**, uma vez que este se encontra *outdated*.

2.2.1 IDE

O IDE é a ferramenta com mais destaque no processo de desenvolvimento, visto ser através deste que será escrito todo o código.

No caso do IDE não existe nenhuma obrigatoriedade sobre qual usar, o programador deve escolher qual o IDE com que se identifica mais, conseguindo assim optimizar o seu workflow.

2.2.1.1 Visual Studio Code



Figura 10:
Visual
Studio
Code – logo

O **Visual Studio Code** é por norma o IDE de preferência de muitos programadores e, isso deve-se essencialmente à sua versatilidade e às diversas extensões disponíveis para o mesmo.

Em anexo (página 89 a 92) é possível encontrar a configuração utilizada no **Visual Studio Code** durante a realização deste projeto. Além destas configurações, é ainda possível encontrar as seguintes referências sobre a configuração e uso deste IDE para desenvolvimento **JavaScript** e **React**: [35, 47, 55, 57]

2.2.1.2 WebStorm



Figura 11:
WebStorm
– logo

O **WebStorm** é outro IDE bastante conhecido e “poderoso”, não sendo necessário instalar *plugins/extensões* devido a este ser bastante completo.

Este IDE faz parte das muitas ferramentas disponibilizadas pela **JetBrains**, tendo como principal vantagem a capacidade de autocomplete sem a necessidade de *plugins/extensões* adicionais.

2.2.2 Prettier – formatação de código



Figura 12:
Prettier
– logo

O **Prettier** é um package destinado à formatação do código auxiliando os desenvolvedores durante todo o processo de desenvolvimento, permitindo criar um ficheiro de configuração com todas as regras que serão aplicadas a quando a formatação do código.

O **Prettier** suporta linguagens/frameworks como **JavaScript**, **JSX**, **Markdown**, **HTML**, **CSS**, **Less**, entre outros.

Juntamente com o **Prettier** pode ainda ser utilizando os seguintes packages:

- **Husky**: permitindo a definição de *hooks* a realizar na execução de comandos do **Git**;
- **Lint Staged**: para executar *hooks* apenas em ficheiros modificados com determinadas extensões (por exemplo `.tsx`);

Em anexo (páginas 93 e 94) é possível encontrar mais detalhes sobre o uso de **Prettier** juntamente com o **Husky**, apresentado ainda como realizar algumas configurações no ficheiro `.prettierrc`.

2.2.3 Gestor de Pacotes

Como gestor de pacotes, ou *package manager*, podem ser utilizadas duas soluções, sendo elas o **NPM** e o **Yarn**. Ambos possuem o mesmo objetivo, a gestão de pacotes num projeto, sendo que o **NPM** vem incluso na instalação no **NodeJS**, já por sua vez o **Yarn** necessita de ser instalado posteriormente.

O **Yarn** conta com algumas melhorias em relação ao **NPM**, na tabela que se segue é possível analisar uma pequena comparação entre ambos³.

	Sem Cache	Com Cache	Reinstalar
NPM 6.13.4	67 segundos	61 segundos	28 segundos
Yarn 1.21.1	57 segundos	29 segundos	1.2 segundos

Tabela 2: Comparação entre **Yarn** e **NPM**

Além das diferenças apresentadas acima, o **Yarn** conta com outras melhorias em comparação ao **NPM**, como por exemplo:

- Interface mais *clean*;

³Retirado de [5]

- Facilidade de uso – determinados comandos tornam-se mais intuitivos com o **Yarn**;
- Possibilidade de reinstalar packages sem conexão à Internet.

Em anexo (páginas 79 e 80) é possível encontrar como proceder à instalação do **Yarn**.

2.3 Controlo de Versões

Para controlo de versões e alterações foi utilizado o **GIT** em conjunto com o **GitLab**, sendo seguido o workflow apresentado na imagem que se segue.

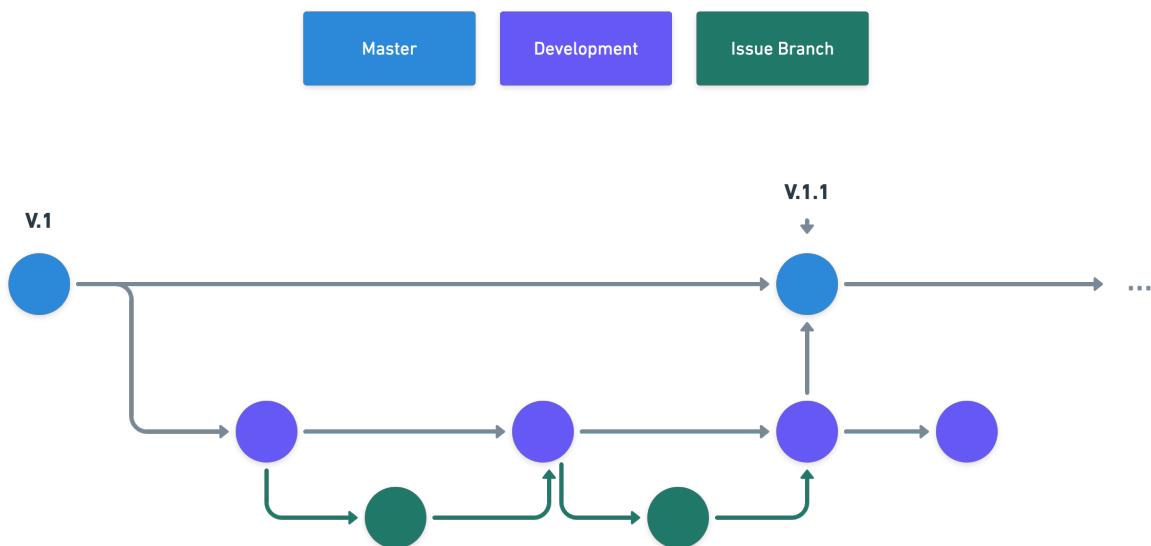


Figura 13: Workflow seguido no **GitLab** durante o desenvolvimento

Como é possível analisar, existe uma branch principal, normalmente com o nome **Master** ou **Main** que contém essencialmente versões do projeto. Durante o desenvolvimento existe uma outra branch destinada apenas ao desenvolvimento, que por sua vez são criadas branches através desta para a resolução de issues.

Ao criar uma branch para a resolução de uma issue, é criado também o merge request para a mesma, ficando este em estado de WIP ou Draft de forma a indicar que ainda existe trabalho em progresso.

Assim que a issue é concluída, é removido do merge request o estado de WIP ou Draft sendo assim analisado e posteriormente realizado o merge para a branch em questão.

2.4 Metodologia

A metodologia adotada para este projeto foi baseada em **SCRUM**, contando com reuniões diárias, bem como *sprints* quinzenais ou semanais, como ainda com *user stories* e *issues*.

Para auxiliar nesta metodologia, bem como toda a gestão das *issues* e tarefas em questão, foi necessário recorrer a software externo, conseguindo assim uma maior optimização na distribuição e organização das *issues*. Os softwares utilizados ao longo do projeto encontram-se listados de seguida.

2.4.1 GitLab

A primeira ferramenta utilizada para controlar as tarefas existentes para o projeto foi o **GitLab**, criando para tal *issues*, sendo apresentadas numa *board*. Posteriormente, era através destas *issues* também criado um *merge request*, ficando assim associados.

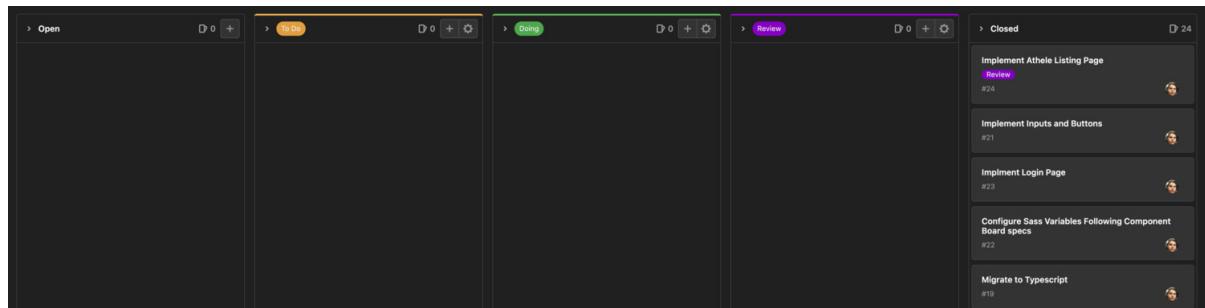


Figura 14: Board utilizada no **GitLab**

A imagem anterior apresenta a *board* utilizada, bem como as colunas existentes para as vários etapas de desenvolvimento de uma *issue*.

2.4.2 Jira

O **Jira** é um software bastante comum em empresas que usam metodologias *Agile*, sendo bastante completo, mas ao mesmo tempo complexo.

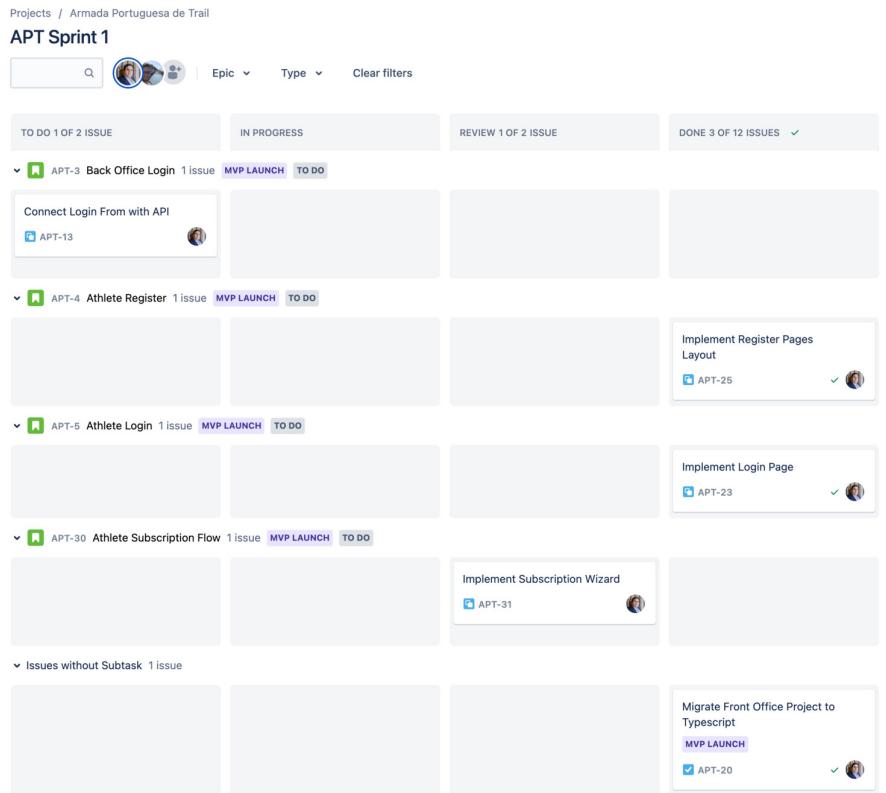


Figura 15: Board utilizada no **Jira**

Como é possível analisar, a *board* é novamente constituída por várias colunas que representam o processo de desenvolvimento de uma *issue/tarefa*.

Além da vista em *board*, o **Jira** permite ainda visualizar as *issues* no modo *backlog*, aparecendo no formato de lista.

2.4.3 ClickUp

Por fim, o último *software* utilizado para a gestão das tarefas do projeto foi o **ClickUp**. Este *software* conta com uma interface mais moderna, bem como diversas integrações possíveis, tornando-o assim bastante completo.

O **ClickUp** conta com diversos tipos de visualizações, desde da vista em *board*, lista, calendário ou até mesmo em vista de gráfico de *Gant*. A imagem que se segue representa a vista em *board*.

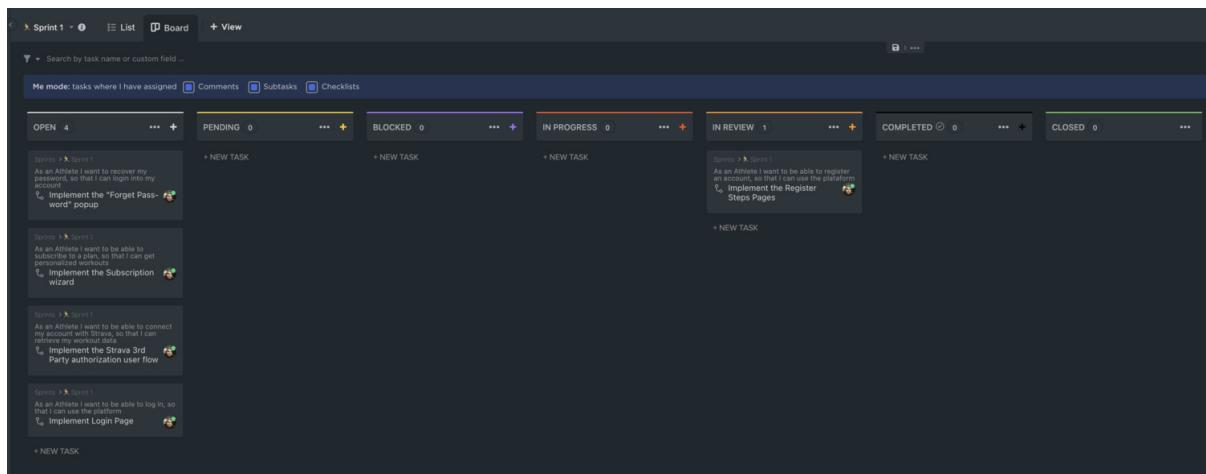


Figura 16: Vista em board no *ClickUp*

Capítulo 3

Visão geral do projeto

Neste capítulo será apresentada uma visão geral do projeto, sendo apresentados os principais objetivos do projeto para cada uma das vertentes (*backoffice* e *frontoffice*), analisando de forma resumida o workflow executado em cada vertente do projeto (*backoffice* e *frontoffice*). As principais dependências externas serão também apresentadas neste capítulo, bem como a sua aplicabilidade de forma resumida.

3.1 Perspetiva do Produto

O projeto **BeApt**, ou **Be Armada Portuguesa do Trail**, tem como objetivo preparar atletas para desafios de alta competição, tal como ultramaratonas, ultra-trails, triatlos, entre outros. Desta forma, o projeto tem que ser capaz de:

- **Backoffice:** Possibilitar a gestão de atletas e treinos;
- **Frontoffice:** Possibilitar uma fácil interpretação dos treinos, bem como o registo dos resultados obtidos.

Para tal, foi idealizada uma aplicação web composta por duas vertentes, a vertente de *backoffice*, destinada a treinadores e administradores da plataforma e, a vertente de *frontoffice*, destinada aos atletas, onde estes conseguirão ver os treinos que lhe foram atribuídos, bem como registar resultados desses mesmos treinos.

3.2 Perspetiva do Utilizador

Como é possível perceber pelo tópico anterior, existem essencialmente 2 tipos de utilizadores, os atletas e treinadores, onde se encaixa também o administrador ou administradores. Desta forma, a figura que se segue é um pequeno exemplo do fluxo destas duas vertentes. De referir que os atletas irão sempre executar ações na vertente de *frontoffice* do projeto e, por sua vez, o administrador e treinadores irão executar na vertente de *backoffice*.

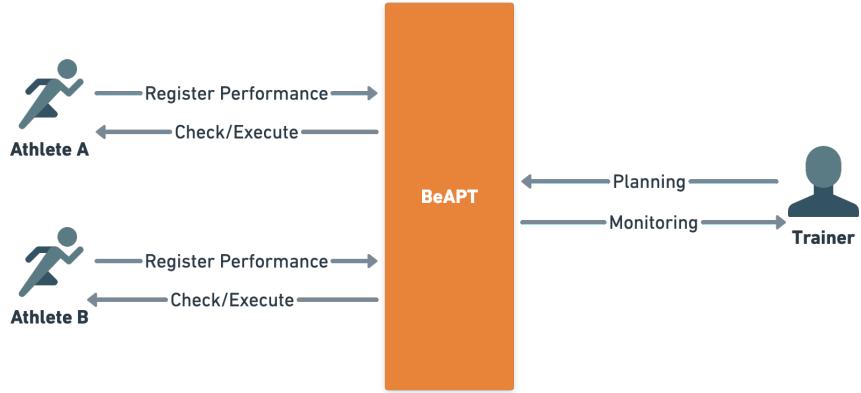


Figura 17: Exemplo de utilização por parte dos atletas e treinadores

A lista que se segue apresenta, com mais detalhe, as funcionalidades que estão ao dispor para os atletas na vertente de frontoffice:

- Um local para a edição dos seus dados pessoais e biométricos, estando protegido por autenticação;
- Um local onde é possível a consulta e carregamento de treinos realizados, bem como os treinos que lhe foram atribuídos;
- Possibilidade de visualizar a sua evolução graficamente.

Por sua vez, já o treinador e administrador, contam com as seguintes funcionalidades na vertente de backoffice:

- Um local destinado à criação de treinos modelo, como um *template*, com parâmetros genéricos, que posteriormente são atribuídos aos atletas;
- Um local para a consulta de dados pessoais e biométricos dos atletas, sendo ainda possível a consulta de treinos realizados e por realizar;
- Possibilidade de visualizar graficamente a evolução dos atletas.

3.3 Pressupostos de Restrições

A única restrição imposta no projeto foi o uso do [Strava](#) para o acompanhamento dos atletas. Toda a stack tecnológica ficou a cargo da empresa [Jimmy Boys](#).

3.4 Dependências

Ao longo do desenvolvimento do projeto em questão foi necessário recorrer a alguns serviços externos e packages, conseguindo assim implementar todas as funcionalidades pretendidas.

A lista que se segue apresenta algumas destas dependências, bem como uma breve explicação do seu uso.

- **Stripe:** utilizado para a realização de pagamentos e subscrições de planos de treino;
- **Strava:** utilizado para recolha de dados do atleta relacionados com atividade física (como detalhes de determinada corrida);
- **Apollo Client:** para realização de *queries GraphQL* na API;
- **React Step Wizard:** utilizado para a criação do *wizard* (formulário com vários passos) de registo de atleta;

De referir que o **TypeScript** encontra-se instalado⁴ em ambos os projetos, sendo inicialmente ambas as vertentes do projeto⁵ migradas de **JavaScript** para **TypeScript**.

⁴**Nota:** em anexo (página 53) é possível encontrar todos os detalhes sobre a instalação do **TypeScript**

⁵Vertente destinada ao atleta (*frontoffice*) e a vertente destinada ao treinador (*backoffice*)

Capítulo 4

Design & Implementação da Solução

4.1 Arquitetura Conceptual

O projeto em questão encontra-se composto por duas camadas, a parte de *back-end* e o *front-end*. A imagem que se segue representa a arquitetura, bem como a comunicação entre ambas as camadas.

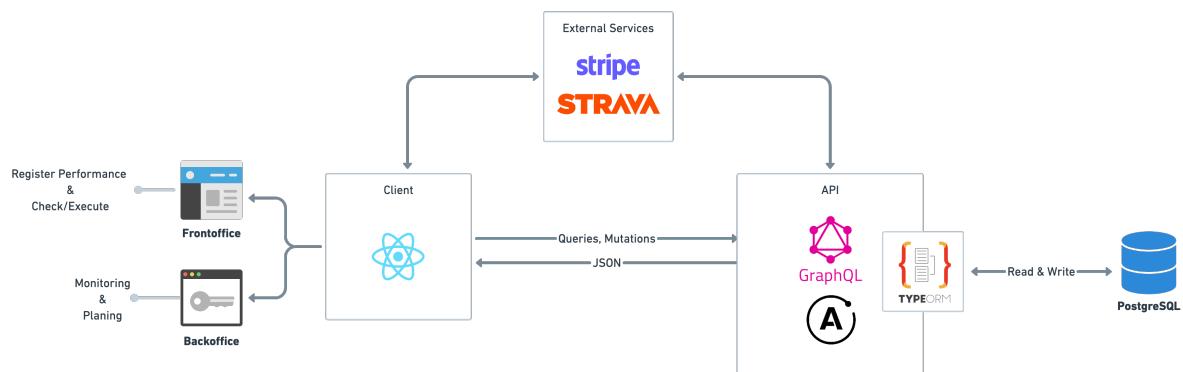


Figura 18: Arquitetura Conecptual do projeto

Como é possível analisar do lado do servidor (*back-end*) é possível encontrar uma API's composta por:

- **GraphQL;**
- **Apollo Server;**
- **TypeORM;**
- **KoaJS⁶** (não representado no diagrama);
- **PostgreSQL;**

⁶Referências Bibliográfica sobre KoaJS: [36, 25]

Já do lado do cliente (front-end), é possível encontrar como base do projeto a biblioteca **React**, utilizada nas duas vertentes do lado do cliente, o backoffice, direcionado ao administrador e treinadores da plataforma e no frontoffice, destinada aos atletas. No lado do cliente é também usado o **Apollo Client** para realizar queries no **GraphQL**.

Ambas as camadas (front-end e back-end) comunicam com serviços externos, no caso o **Stripe** para pagamentos e o **Strava** para informações relacionadas com o atleta (como corridas realizadas, entre outras).

4.2 Diagramas de Sequência

4.2.1 Gestão de Sessão

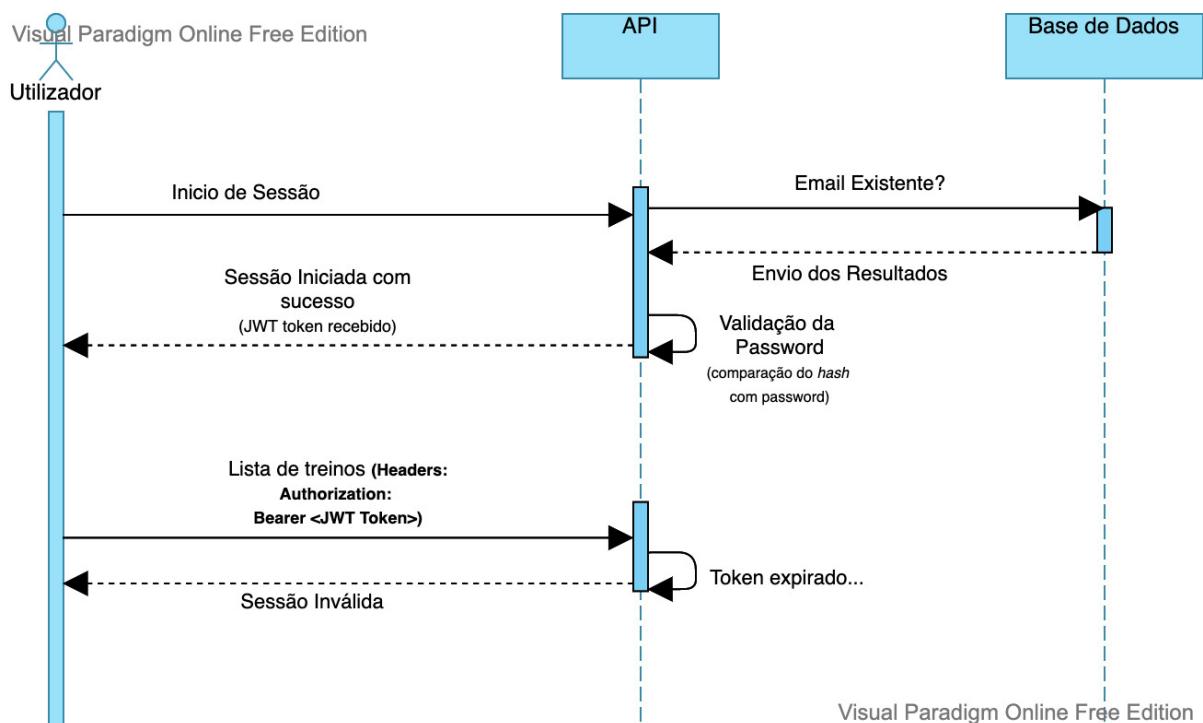


Figura 19: Diagrama de Sequência de Gestão de Sessão

A gestão de sessão é um ponto importante a abordar, pois será através da sessão que os atletas, treinadores e administrador poderão ou não executar determinadas ações. O diagrama acima apresenta o fluxo seguindo na gestão de sessão, sendo que inicialmente é realizado uma inicio de sessão no front-end do projeto, que por sua vez vai enviar os dados à **API**, onde é verificado se o email introduzido existe na base de dados, no caso de existir é então validada a password enviada, sendo comparada com o *hash* (password encriptada) armazenado na base de dados. Com todos os dados validados é devolvido um **JWT token** para utilizar nos pedidos que necessitem de autenticação.

No diagrama é apresentado o caso do pedido da lista de treinos, onde é enviado por *HTTP Headers* o token, o excerto de código que se segue apresenta um exemplo de pedido com o package Axios e o

envio do HTTP Header de autenticação.

```
1 import axios from 'axios';
2
3 axios.get('<api-url>/<route>', {
4   headers: {
5     Authorization: 'Bearer <jwt-token>'
6   }
7 });

```

Exerto de Código 5: Exemplo de pedido com o package **Axios** e autenticação por **HTTP Headers**

Assim, o token **JWT** recebido é enviado para a **API**, onde será validado se este expirou ou não. No caso deste ter expirado, o cliente será redirecionado para a página de inicio de sessão com o HTTP Status Code 401⁷ na resposta da **API**.

4.2.2 Inicio de Sessão

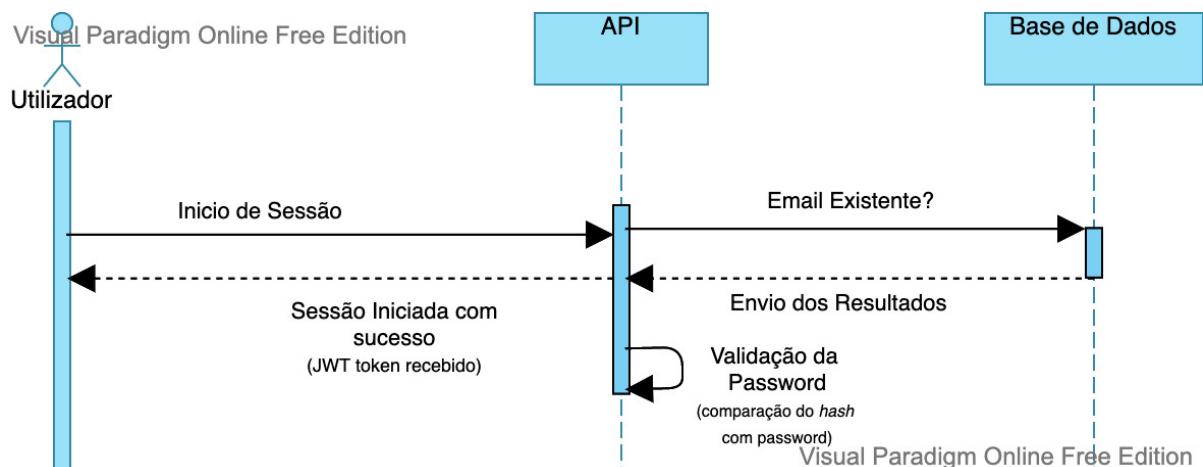


Figura 20: Diagrama de Sequência para o início de sessão

O inicio de sessão é semelhante em ambas as vertentes do projeto, porém, no frontoffice o atleta conta com a possibilidade de iniciar sessão recorrendo às redes sociais como **Facebook**, **Strava** e **Google**. Os mockups que se seguem apresentam o ecrã de inicio de sessão do backoffice e do frontoffice.

⁷HTTP Status Code 401 – Unauthorized ([Mais Informações](#))

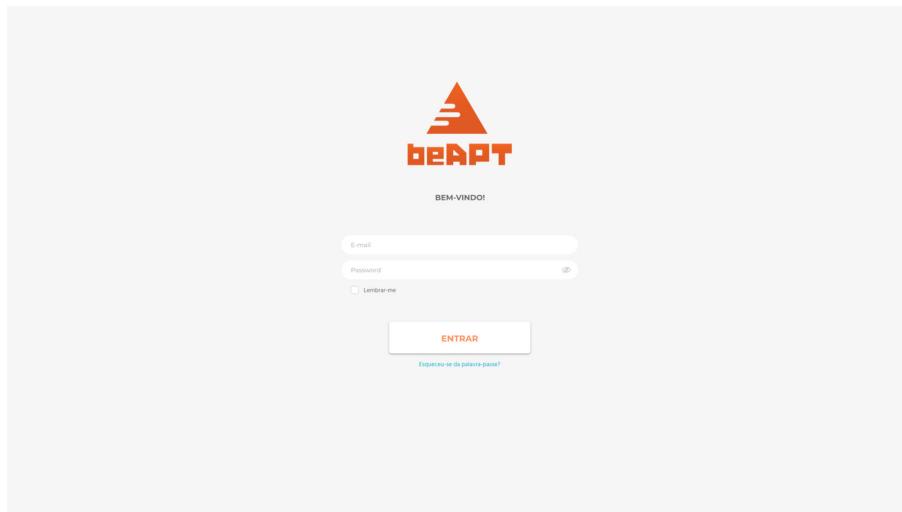


Figura 21: Mockup do ecrã de inicio de sessão – backoffice

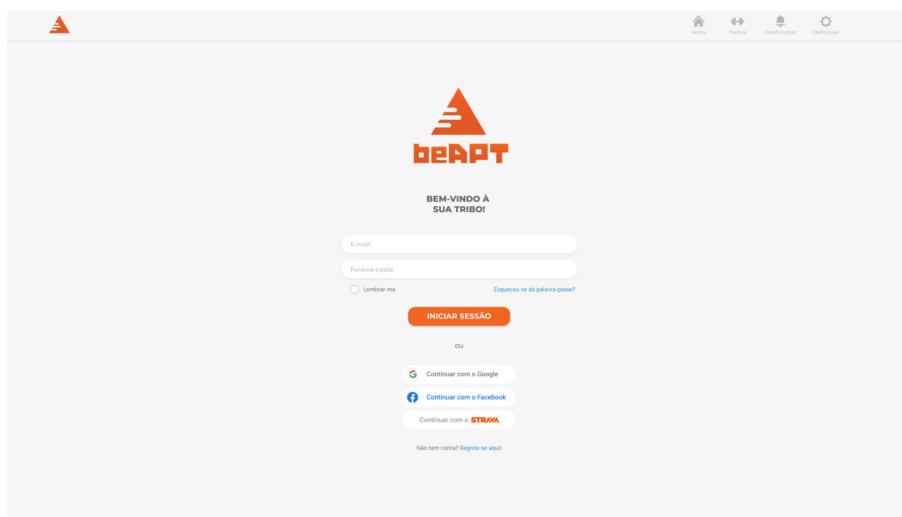


Figura 22: Mockup do ecrã de inicio de sessão com redes sociais – frontoffice

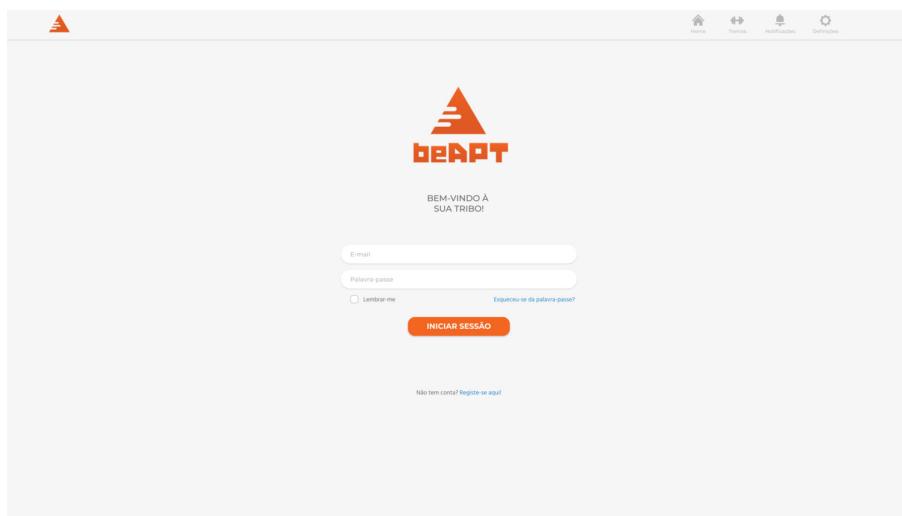


Figura 23: Mockup do ecrã de inicio de sessão – frontoffice

Importante referir que após a sessão ser iniciada com sucesso é recebido (na resposta da API) um token **JWT**, sendo este guardado e utilizado nos pedidos que necessitam de autenticação. No caso dos dados não se encontrarem corretamente inseridos, será apresentada uma mensagem de erro, não sendo recibo qualquer token na resposta.

4.2.3 Atleta – criação de conta

A criação da conta de um atleta encontra-se presente na vertente de frontoffice, sendo este não só executado num passo. Assim sendo, ao longo deste tópico serão apresentados os diagramas de sequência para cada um destes passos.

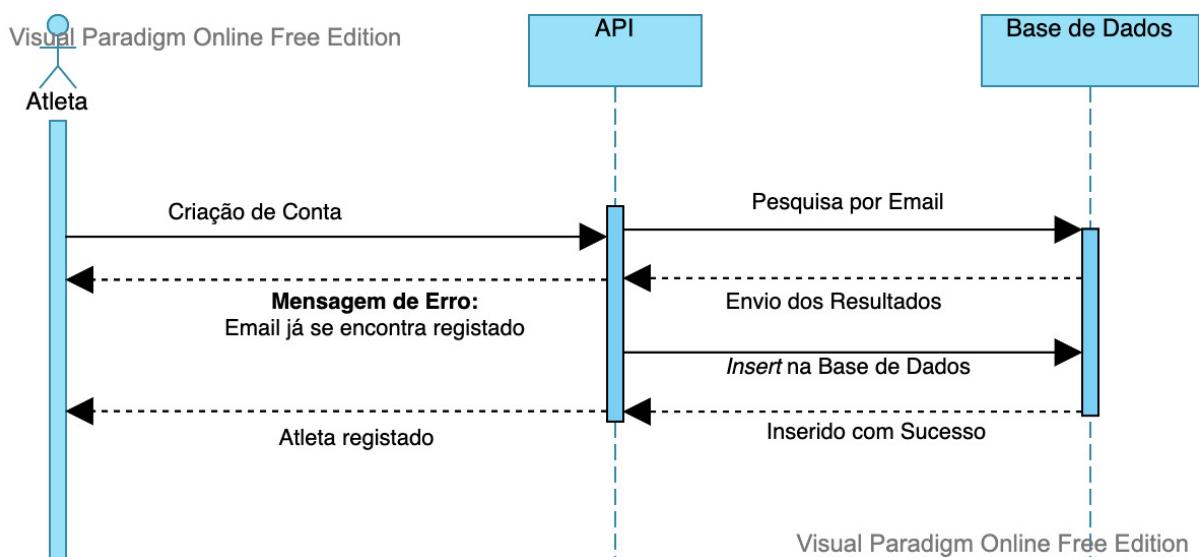


Figura 24: Diagrama de Sequência para a criação de conta de um atleta

O primeiro passo é a criação de conta básica, onde o atleta necessita de preencher os campos:

- Nome;
- Apelido;
- E-mail;
- Password.

Após o preenchimento destes campos e, a aceitação dos **Termos & Condições**, é enviado um pedido à **API**, onde por sua vez realiza determinadas validações.

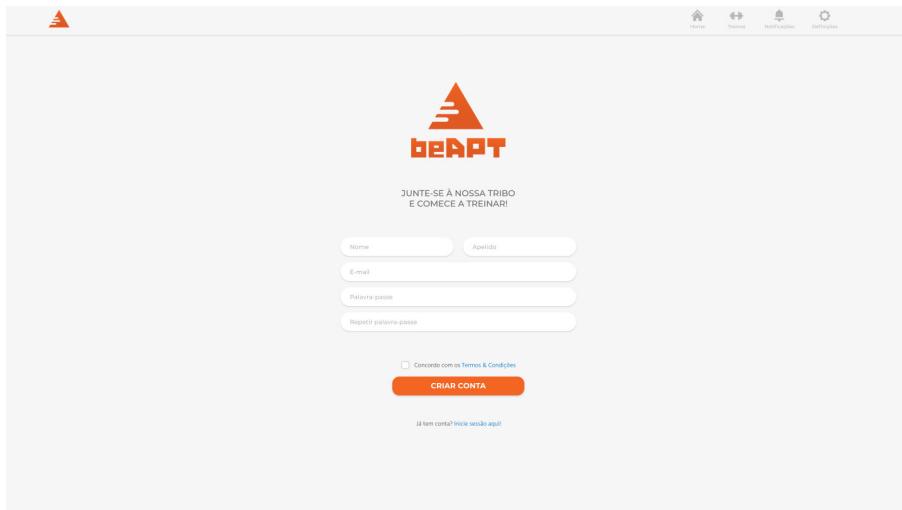


Figura 25: Mockup do ecrã de criação de conta – frontoffice

No caso, a **API** realiza uma consulta à base de dados para verificar se os dados enviados já existem, se existirem é devolvida uma mensagem ao cliente a informar que ocorreu um erro. No figura que se segue são apresentados dois erros, um deles informando que o e-mail já se encontra registado.

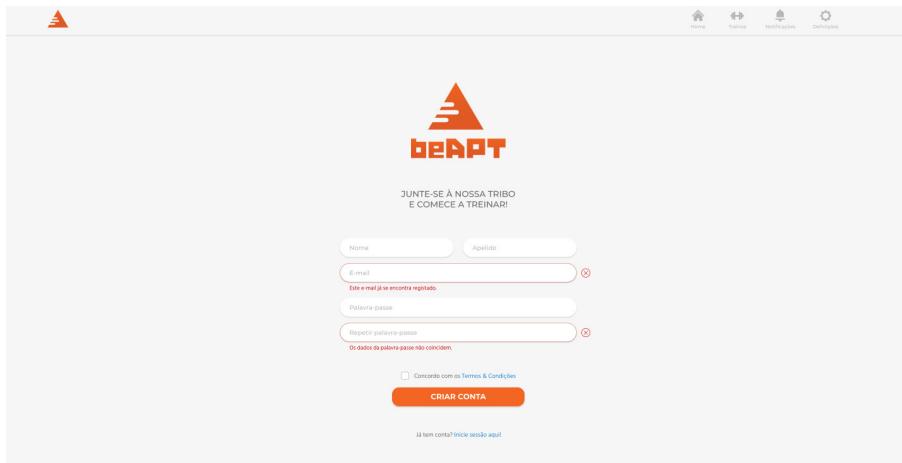


Figura 26: Mockup do ecrã de criação de conta com dados inválidos – frontoffice

No caso de não ocorrer nenhum erro e, a **API** devolver uma mensagem de sucesso, o atleta é redirecionado para o último passo do registo. Neste último passo são pedidos ao atleta outros dados pessoais como data de nascimento, contacto telefónico, morada, dados biométricos (altura, peso, frequência cardíaca máxima registada e frequência cardíaca em repouso) e o objetivo pessoal pretendido.

Figura 27: Mockup de informações de registo – frontoffice

Neste último ecrã de criação de conta é possível encontrar alguns componentes criados, sendo encontrados essencialmente neste ecrã, os *inputs* de carregamento de foto de perfil e imagem de fundo, bem como a *tooltip* que aparece ao colocar o rato no ícone de questão.

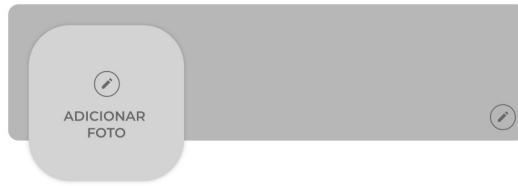


Figura 28: Inputs de carregamento de imagens



Figura 29: Componente Tooltip

O componente **Tooltip** tem como objetivo apresentar informação adicional que não esteja presente no ecrã, no caso da imagem apresentada anteriormente, indica como preencher os dados biométricos do atleta. Já os *inputs* destinados ao carregamento da imagem de perfil e imagem de fundo estes são posteriormente utilizadas na página do atleta, tal como é apresentado de seguida.



Figura 30: Imagens carregadas para o perfil do atleta – frontoffice

4.2.4 Lista de Atletas

A lista de atletas encontra-se presente na vertente de backoffice do projeto, sendo esta destinada aos treinadores e administrador da plataforma. Assim sendo, o diagrama de sequência que se segue apresenta o fluxo que acontece ao aceder a esta página.

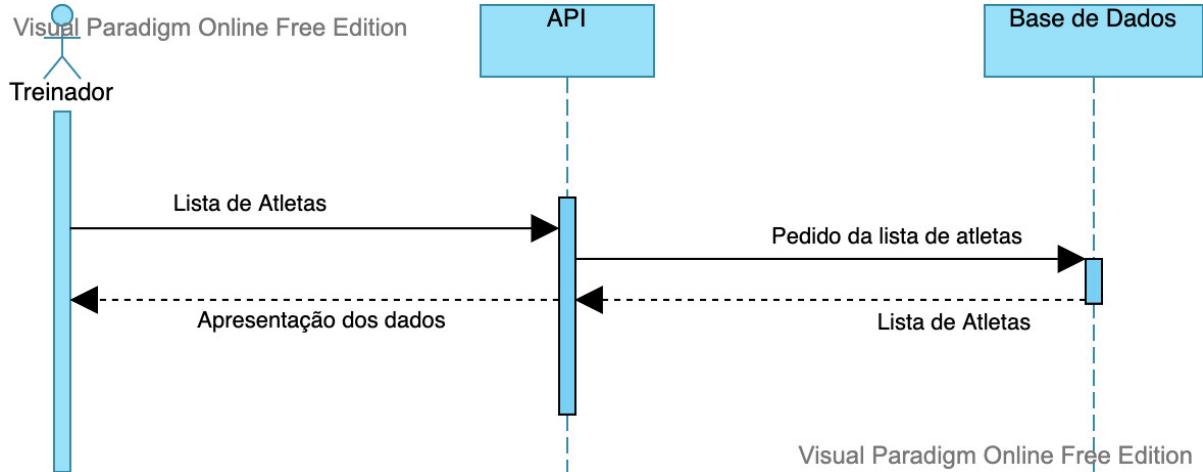


Figura 31: Diagram de Sequência para a consulta da lista de atletas

Importante referir que para ter acesso a esta página é necessário possuir sessão iniciada e um token válido (que ainda não tenho expirado). No caso de um token expirado, tal como é possível encontrar no diagrama referente à gestão da sessão, o utilizador será redirecionado para a página de inicio de sessão. Caso não tenha permissões para realizar este pedido, a API devolve me um *HTTP Status Code 403*⁸.

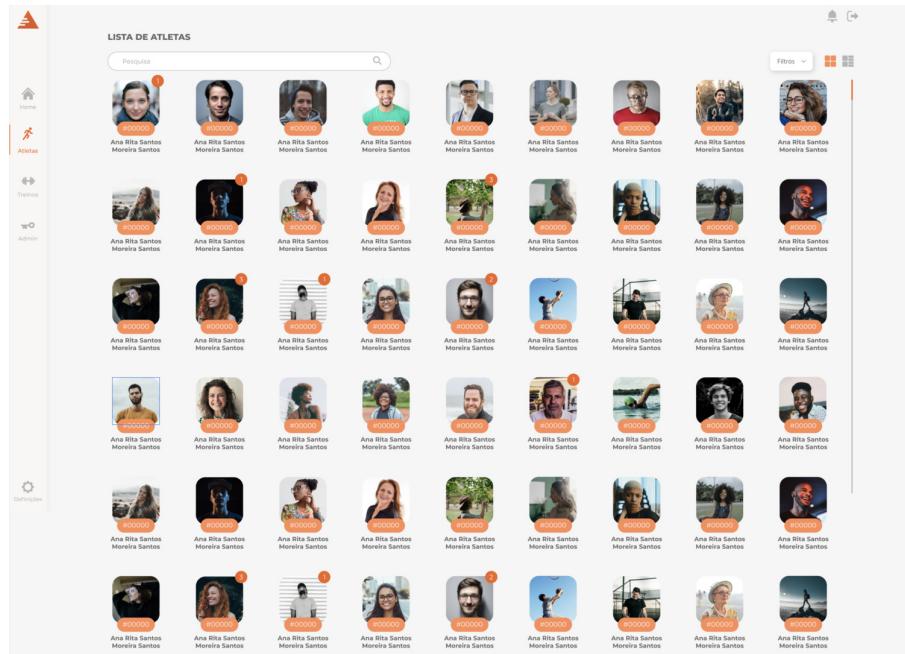


Figura 32: Lista em mosaico dos atletas – *backoffice*

⁸HTTP Status Code 403 – *Forbidden* ([Mais Informações](#))

LISTA DE ATLETAS						
	Nome	Genero	Modalidade	Tipo de Plano	Estado da Subscrição	Objectivo
#0032	Araújo, Rita Moreira dos Santos	Feminino	Natação	Base	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0020	Pedro Miguel Ferreira Borges	Masculino	Triatlo	Avançado	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0030	José Miguel António Almeida	Masculino	OCR	Base	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0029	Xavier Castanhosa Pereira	Masculino	OCR	Rendimento	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0028	António Maria Torres e Ferreira	Masculino	Triatlo	Rendimento	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0027	Maria Luísa Sá e Silva	Feminino	Triatlo	Avançado	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0026	André Silva e Costa	Masculino	Natação	Base	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0025	António Joaquim Machado	Masculino	Natação	Rendimento	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0024	Araújo, Patrícia Ferreira da Costa	Feminino	Triatlo	Rendimento	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0023	Matilda Vieira	Feminino	Triatlo	Rendimento	Suspenso	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0022	Diego Luis Castro Emanuel	Masculino	ETT	Avançado	Cancelado	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0021	Rita Alexandra Gonçalves	Feminino	Corrida	Rendimento	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0020	Mónica Simões e Corte-Real	Feminino	Corrida	Base	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0019	Maria Miguel Reis	Feminino	Corrida	Base	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0018	Inês Santos Palma	Feminino	Natação	Base	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0017	Diogo Luis Castro Emanuel	Masculino	ETT	Base	Cancelado	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0016	Rita Alexandra Gonçalves	Feminino	Corrida	Rendimento	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0015	Mónica Simões e Corte-Real	Feminino	Corrida	Base	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0014	Maria Miguel Reis	Feminino	Corrida	Base	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.
#0013	Inês Santos Palma	Feminino	Natação	Base	Ativo	Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen.

Figura 33: Lista dos atletas – backoffice

Como é possível analisar pelas figuras anteriores, a lista de atletas encontra-se disponível em duas vistas, a vista em mosaico e lista. Apesar de serem formas de visualização com estilos diferentes apenas é realizado um pedido à [API](#), sendo estes dados utilizados em cada uma destas vistas.

No que toca a componentes, a vista em mosaico utiliza essencialmente o componente **Avatar** apresentado em [anexo \(página 67\)](#), sendo este apenas repetido para cada atleta recebido no pedido à [API](#). Já a vista em lista utiliza um componente criado especificamente para esta página, o **AthleteRow**, onde consiste numa linha (como o nome indica), onde é novamente utilizado o componente **Avatar**.

Para a criação destes 2 modos apenas foi necessário recorrer a criação de um estado (`useState`) e () do **React**), onde ao clicar num dos icons é chamada uma função, tal como é possível analisar nos excertos de código que se seguem.

```

1 const renderList = () => (
2   <>
3     <div className={styles['header']}>
4       <span className={styles['avatar-col']}></span>
5       <span className={styles['tag']}>Tag</span>
6       <span className={styles['name']}>Nome</span>
7       <span>Género</span>
8       <span className={styles['sport']}>Modalidade</span>
9       <span>Tipo de Plano</span>
10      <span className={styles['subscription-status']}>
11        Estado da Subscrição
12      </span>
13      <span className={styles['objective']}>Objetivo</span>

```

```
14     </div>
15
16     {data.map((athlete, key) => (
17         <AthleteRow key={key} athlete={athlete} />
18     )));
19     </>
20 );
```

Excerto de Código 6: Função para renderizar a lista de atletas

```
1 const renderMosaic = () => (
2     <div className={styles['grid']}>
3         {data.map((athlete, key) => (
4             <AthleteCard key={key} athlete={athlete} />
5         )));
6     </div>
7 );
```

Excerto de Código 7: Função para renderizar o mosaico de atletas

Com as funções criadas, basta apenas realizar o código com as restantes informações da página.

```
1 const Athletes = () => {
2     // ...
3
4     const [isList, setIsList] = useState(true);
5
6     // ...
7
8     return (
9         <div className={styles['root']}>
10            <h2>Lista de Atletas</h2>
11            <div className={styles['top-items']}>
12                <SearchInput
13                    placeholder='Pesquisa'
14                    onChange={athletesSearch}
15                    className={styles['search']}
16                />
17
18                <div className={styles['actions-container']}>
19                    <div className={styles['icons']}>
20                        <div
21                            className={
22                                !isList ? styles['active-icon'] : undefined
23                            }
24                        >
```

```

23         }
24     >
25     <Icon
26       name='grid'
27       onClick={() => setIsList(false)}
28     />
29   </div>
30   <div
31     className={
32       isList ? styles['active-icon'] : undefined
33     }
34   >
35     <Icon name='grid' onClick={() => setIsList(true)} />
36   </div>
37   </div>
38 </div>
39 </div>
40
41   <div className={styles['content']}>
42     {isList ? renderList() : renderMosaic()}
43   </div>
44 </div>
45 );
46 }

```

Exerto de Código 8: Código da página de listagem dos atletas existentes

4.3 Estrutura de Pastas

A estrutura de pastas em ambas as componentes do projeto (frontoffice e backoffice), existindo apenas ficheiros diferentes. A figura que se segue representa a estrutura geral utilizada.

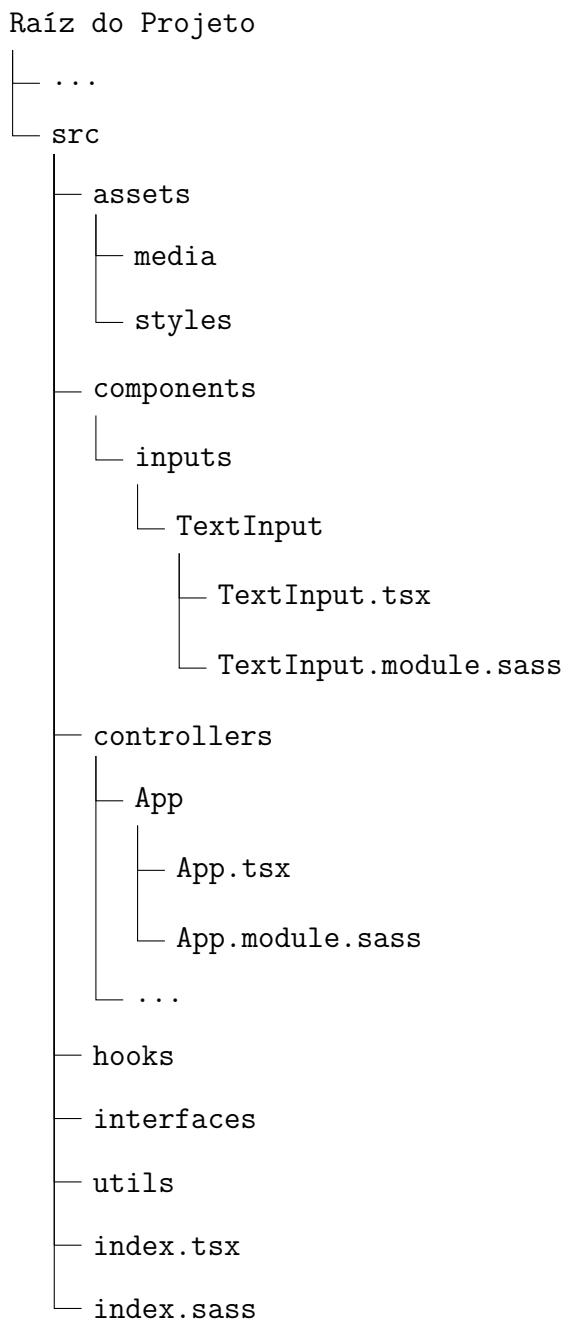


Figura 34: React – estrutura de pastas utilizada

Como é possível analisar a pasta principal do projeto é a `src/`, sendo esta que contém desde os componentes, estilos, rotas, etc.. Cada componente ou controller é constituído (na maioria dos casos) por um ficheiro `tsx` e outro `sass`, sendo o ficheiro com extensão `module.sass` responsável pelo estilo de determinado componente ou controller.

No caso dos componentes, sempre que existe mais do que um componente de uma mesma “categoria” (como por exemplo os `inputs`) é criada uma pasta principal para todos os componentes dessa “categoria”.

Capítulo 5

Resultados

Ao longo deste documento já foi possível encontrar algumas informações relativas aos resultados atingidos, bem como alguns dos componentes criados e apresentados durante os diagramas de sequência. Porém, neste ponto serão colocadas figuras da implementação dos diagramas e componentes apresentados anteriormente.

5.1 Backoffice

A vertente de *backoffice* foi a mais desenvolvida numa primeira fase, desta forma a figura que se segue apresenta as pastas **components** e **controllers**, sendo possível ter uma visão dos componentes que foram desenvolvidos.

Importante referir que a pasta **components** contém todos os componentes e, sempre que possível, organizados por categorias, ou seja, é criada uma pasta **inputs** para tudo o que são **inputs** personalizáveis (como **selects**, **checkboxes**, entre outros). No que toca à pasta **controllers** refere-se a páginas, páginas estas que podem conter rotas e/ou subrotas. Caso tenha subrotas, os ficheiros são colocados dentro da pasta da rota principal.



Figura 36: Backoffice: pasta controllers

Figura 35: Backoffice: pasta components

Como foi apresentado nos diagramas de sequência, a lista de atletas pode ser consulta de duas formas, em vista de **lista** e em **mosaico**, tal como é apresentando nas figuras que se seguem.

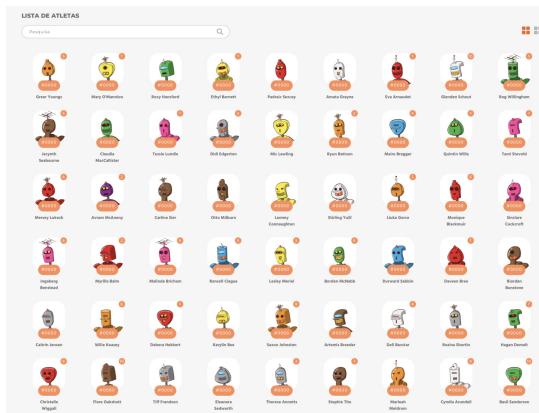


Figura 37: Backoffice: vista em mosaico dos atletas

LISTA DE ATLETAS					
Tag	Nome	Género	Motivação	Tipo de Prova	Estado de Inscrição
#0001	Grace Young	Mulher	OCA	Assinado	Verificar
#0002	Max O'Nolan	Homem	Tríatlo	Bem-vindo	Verificar
#0003	Rey Harford	Mulher	Triatlo	Base	Verificar
#0004	Emily Bennett	Fêmea	Natação	Bem-vindo	Verificar
#0005	Pedro Senna	Homem	Natação	Bem-vindo	Verificar
#0006	Amy Groves	Mulher	OCA	Base	Verificar
#0007	Eva Arnedo	Fêmea	Natação	Assinado	Verificar
#0008	Gabriel Schulz	Fêmea	OCA	Base	Verificar
#0009	Greg Wellingham	Homem	Corrida	Bem-vindo	Verificar
#0010	Claudia MacClester	Mulher	OCA	Assinado	Verificar
#0011	Teixeira Lurdes	Fêmea	OCA	Base	Verificar
#0012	Daniel Edgerton	Mulher	Natação	Assinado	Verificar
#0013	Alfie Lawing	Fêmea	OCA	Assinado	Verificar

Figura 38: Backoffice: vista em lista dos atletas

Ao clicar num atleta, o utilizador é redirecionado para o perfil do atleta, onde este conta com várias abas. Cada aba é uma subrota do perfil, alterando apenas o conteúdo de cada aba.

This screenshot shows a detailed view of an athlete's profile. It includes tabs for general information, training, and subscription status. The main content area is divided into several sections: Updates (with two recent updates), Personal Data (with fields for birthdate, phone, email, and NIF), Athlete Profile (with a dropdown menu for selecting the athlete's profile), Documents (with two uploaded files labeled 'Exame 1' and 'Exame 2'), Address (with fields for address, postal code, city, and country), General Objective (with a note and a date), and Biometric Data (with fields for age, height, weight, and various heart rate metrics). Below these are sections for speeds and distances.

Figura 39: Backoffice: página de perfil do atleta

5.2 Frontoffice

Tal como acontece na vertente do backoffice, a pasta de **components** é organizada por categoria e, na pasta **controllers** os ficheiros são organizados por rotas e subrotas, tal como no backoffice. As figuras que se seguem apresentam os componentes e controllers desenvolvidos até ao momento.

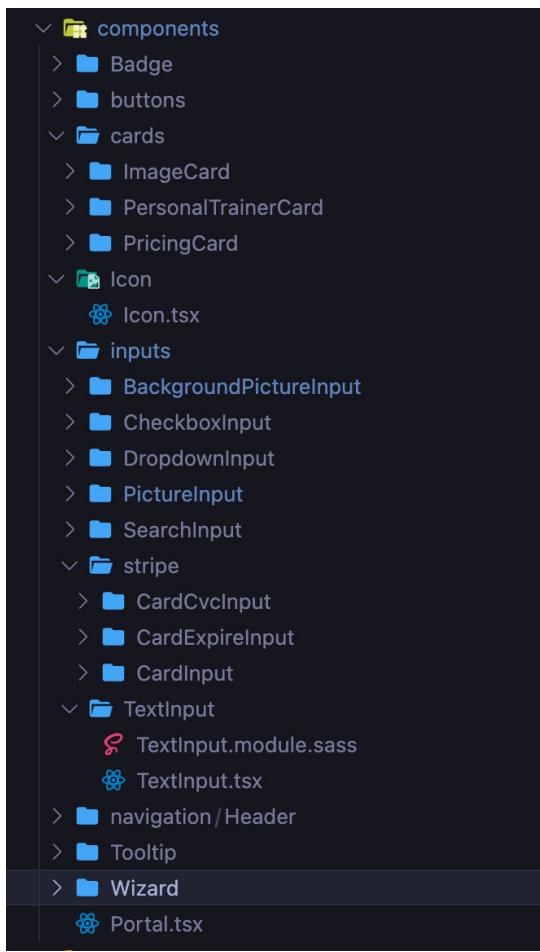


Figura 40: Frontoffice:
pasta *components*

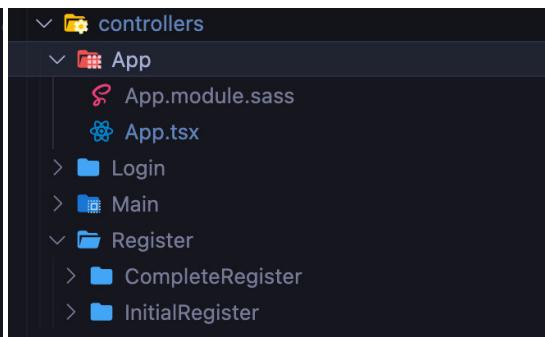


Figura 41: Frontoffice: pasta
controllers

A criação de conta do atleta no frontoffice é composta essencialmente por 2 etapas, tal como foi apresentado anteriormente. Porém após esses dois passos de criação de conta, o atleta é enviado para um wizard, algo semelhante a um formulário com múltiplos passos, onde o atleta escolhe desde a modalidade, treinador e o preçoário. As figuras que se seguem apresentam estes passos da criação de conta do atleta.

Figura 42: Frontoffice:
criação de conta do atleta

INFORMAÇÕES DE REGISTO

Dados Pessoais

Morada

Dados Biométricos

Objetivo Pessoal

Concordo com os Termos & Condições

FINALIZAR REGISTO

Figura 43: Frontoffice:
finalização da criação de
conta

Após a conclusão dos passos anteriores é apresentado o wizard com os múltiplos passos, de forma ao atleta realizar passo por passo facilmente.

Figura 44: Frontoffice:
passo inicial do wizard

Figura 45: Frontoffice:
informações pessoais e de
pagamento do wizard

No final do wizard é apresentado o resumo de todos os passos e ao concluir é dada a possibilidade do atleta conectar com o **Strava** através de um *popup* apresentado.

This screenshot shows the fifth step of a payment wizard. At the top, a progress bar indicates five steps completed. The main content area displays a summary of the purchase: a meal plan for 'Ultra Distância' (Campbell's, Classic Chia) at a monthly price of €119,02, with a 10% discount applied. Below this is a delivery address section for 'Morro de São Paulo, 10, 4700-310 Funchal, Portugal'. A 'FINALIZAR PAGAMENTO' button is at the bottom.

Figura 46: Frontoffice:
resumo do wizard

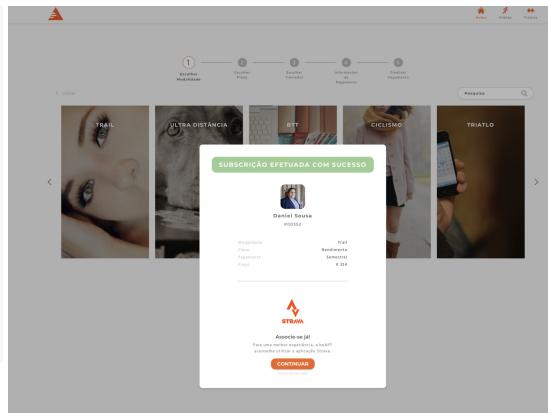


Figura 47: Frontoffice:
popup apresentado no final
do wizard

Capítulo 6

Conclusão

*"The only way to learn a new
programming language is
by writing programs in it."*

Dennis Ritchie

Apesar do projeto não se encontrar finalizado, foi possível cooperar e adquirir novos conhecimentos em ambas as vertentes (*backoffice* e *frontoffice*) do mesmo. Além disso, foi ainda possível praticar e melhorar conhecimentos já existentes, aprofundando todo o conhecimento sobre a biblioteca **React**. Porém, é necessário realçar que todos os objetivos foram atingidos, sendo estes, de uma forma geral, as funcionalidades e layout de inicio de sessão e criação de conta, bem como o *boilerplate* inicial do projeto, seguindo os *mockups* desenvolvidos para este.

Com o projeto foi ainda possível conhecer novas ferramentas e investigar algumas destas durante os tempos livres, como por exemplo o uso de **Styled Components** para definir o estilo da aplicação/-projeto.

É importante referir que apesar de determinadas funcionalidades não terem sido implementadas, surgiu o interesse pela descoberta de soluções para implementações das mesmas, bem como a descoberta de novas ferramentas e packages.

6.1 Formações Adicionais

De forma a conseguir adquirir novos e aprofundar conhecimentos, foram realizadas algumas formações adicionais, complementares ao estágio (por iniciativa própria). A lista que se segue apresenta projetos realizados durante estas formações, bem como as respetivas ligações para o código desenvolvido nestas.

- Next Level Week 04
 - [Informações](#);
 - **Repositório:** [GitHub](#);
 - **Deploy:** [Vercel](#);
- ReactJS Challenge – Slack Clone
 - [Canal do YouTube](#)
 - **Repositório:** [GitHub](#);
 - **Deploy:** [Firebase](#);
- Twitter UI Clone
 - [Vídeo](#)
 - **Repositório:** [GitHub](#);
 - **Deploy:** [Netlify](#)
- Next Level Week 05
 - [Informações](#);
 - **Repositório:** [GitHub](#);
 - **Deploy:** [Vercel](#)
- LinkedIn UI Clone
 - [Vídeo](#)
 - **Repositório:** [GitHub](#)
 - Por Concluir
- TypeGraphQL – Code/drop #74
 - [Vídeo](#)
 - **Repositório:** [GitHub](#)
 - **Temas Principais:** GraphQL e TypeGraphQL

Nos projetos apresentados acima, foi possível aplicar conhecimentos já existentes na biblioteca **React**, biblioteca esta usada para o desenvolvimento do projeto, onde permitiu aplicar novos conhecimentos sobre a mesma. O uso de **Styled Components** foi um dos conhecimentos adquiridos nestas formações. O uso de **Styled Components** permite realizar o estilo do projeto sem ser necessário criar ficheiros `.css`, ou mesmo `.sass`, recorrendo para tal a **JavaScript** ou **TypeScript** e a recorrendo a *template literals* para escrever `CSS`. Desta forma é possível introduzir variáveis passadas através de argumento para o estilo.

Outra das formações realizadas permitiu conhecer melhor o uso de **GraphQL** para a construção de uma **API**, apesar de não ser o foco deste projeto.

6.2 Trabalhos Futuros

Ao longo do projeto foram surgindo formas de melhorar o projeto, bem como pontos que seriam importantes abordar, porém o tempo era limitado, não sendo possível avaliar o esforço que implicaria realizar algumas destas e posteriormente implementar as mesmas. Nos parágrafos seguintes é possível encontrar alguns dos trabalhos que poderiam ser realizados futuramente de forma a melhorar o projeto na sua totalidade.

6.2.1 Biblioteca de Componentes

Um dos pontos seria a criação de uma biblioteca de componentes, isto deve-se essencialmente a existirem componentes iguais em ambas as vertentes (*backoffice* e *frontoffice*), evitando também assim o termo **DRY**, ou seja, usar “pedaços” de código iguais, mas em vários locais.

A criação desta biblioteca facilitaria o uso de componentes comuns a ambas as vertentes do projeto (*backoffice* e *frontoffice*), sendo esta instalada através do **NPM** ou **Yarn**. Para realizar esta biblioteca existem várias ferramentas, podendo ser criando um projeto **React**, em que posteriormente este seria publicado como package.

Referências: [21, 24]

6.2.2 Testes

Os testes são um ponto importante nos projetos, uma vez que permitem a deteção de erros que não foram aperceptíveis durante o desenvolvimento. Porém devido ao escasso tempo não foi possível abordar a realização de testes mais a fundo, assim, e tal como é possível encontrar em anexo (página 85 a 88), foram realizados alguns testes mais simples, como a testagem de valores e adição de classes **CSS** de acordo com determinada propriedade. Ainda, é possível perceber como seriam realizados os testes de simulação de inicio de sessão.

Os testes seriam outra das melhorias a implementar, auxiliando na validação do projeto. Seriam possível realizar dois tipos de testes, os *unit tests* ou testes unitários e ainda testes *end-to-end*.

Nos tópicos que se seguem é possível analisar os tipos de testes que se podem realizar, ou seja, testes *end-to-end* e testes unitários, ou *unit tests*.

Referências Adicionais: [49, 51, 15, 3, 11, 10, 40, 31]

6.2.2.1 Unit Testing

Os testes unitários, ou *unit tests*, são testes focados em partes isoladas de um projeto ou sistema, por exemplo um componente **React**. O principal objetivo deste tipo de testes é auxiliar na validação do código produzido e a encontrar bugs que não tenham surgido até então.

Uma das ferramentas frequentemente utilizada para *unit tests* é o **Jest**, podendo recorrer ou não a outros packages adicionais. O **Jest** possibilita realizar testas em diversas *frameworks* e tecnologias, contando ainda com *code coverage*, ou seja, cobertura de código, sendo apresentado no formato de tabela como na figura⁹ que se segue.

⁹Retirada do [site oficial](#)



Figura 48: Jest – cobertura de código

Um teste em **Jest** poderia ser escrito da seguinte forma:

```

1 import ReactDOM from 'react-dom'
2 import Component from 'components/Component';
3
4 it('renders without crashing', () => {
5   const div = document.createElement('div');
6   ReactDOM.render(<Component />, div);
7   ReactDOM.unmountComponentAtNode(div);
8 });

```

Excerto de Código 9: Jest – exemplo de um teste para um componente **React**

Nota

Para executar os testes e validar tanto o teste como o código produzido seria necessário executar o comando:

- **Com NPM:** `npm test;`
- **Com Yarn:** `yarn test.`

É importante ainda referir que em determinados casos pode ser necessário criar o script no ficheiro `package.json` para executar o **Jest** e os testes criados.

6.2.2.2 End-to-end Testing

Os testes *End-to-End* são o contrário dos testes unitários, ou seja, validam o projeto como um todo, sendo utilizados frequentemente para validar o fluxo seguido pelo projeto. Uma ferramenta que pode ser utilizada para realizar este tipo de testes é o [Cypress](#).

O excerto de código que se segue é um exemplo de teste¹⁰ realizado com [Cypress](#), sendo possível analisar logo de seguida o seu resultado.

```
1 it('should add a new todo to the list', () => {
2   // network stubs
3   cy.server();
4   cy.route('GET', `${serverUrl}/todos`, '@updatedJSON').as('getAllTodos');
5   cy.route('POST', `${serverUrl}/todos`, '@addTodoJSON').as('addTodo');
6 });


```

Excerto de Código 10: Cypress – exemplo de teste

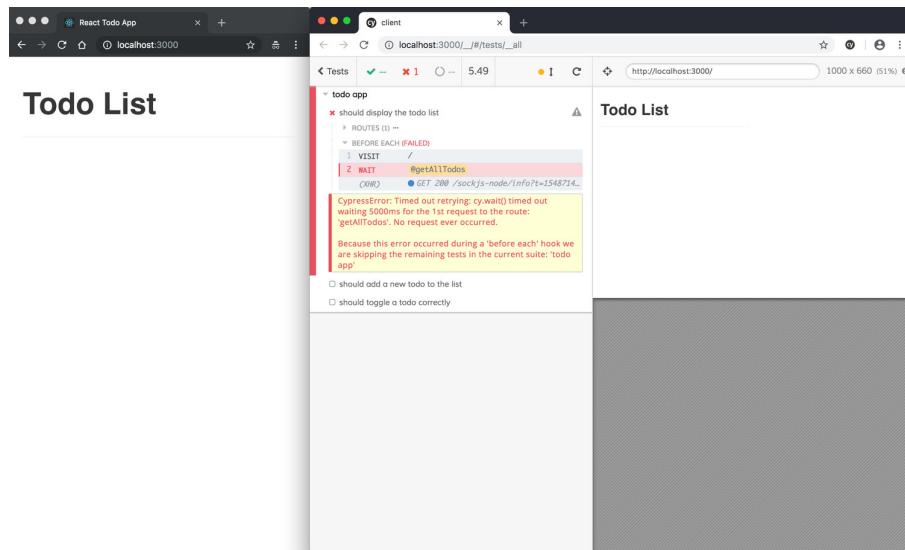


Figura 49: Cypress – resultado do teste apresentado

Também é possível executar testes *End-to-End* com [Jest](#) e [Puppeteer](#)¹¹, sendo uma alternativa ao [Cypress](#).

¹⁰ Retirado da [blog oficial](#)

¹¹ Referências: [3, 11]

Referências Bibliográficas

- [1] _CODE. *Creating your React project from scratch without create-react-app. The complete guide.* Inglês. Dev.to. Jun. de 2021. URL: <https://dev.to/underscorecode/creating-your-react-project-from-scratch-without-create-react-app-the-complete-guide-4kbc> (acedido em 06/06/2021).
- [2] Software Development Academy. *Create a React app using the command line.* Inglês. Dev.to. Jun. de 2021. URL: <https://dev.to/softwaredevacademy/create-a-react-app-using-the-command-line-3e6p> (acedido em 06/06/2021).
- [3] Benjamin Ajewole. *End to End testing in React with Puppeteer and Jest.* Inglês. Mar. de 2020. URL: <https://rexben.medium.com/end-to-end-testing-in-react-with-puppeteer-and-jest-6a0b1b8cff6b> (acedido em 06/06/2021).
- [4] Apollo Docs – Get Started (React). Inglês. Apollo. URL: <https://www.apollographql.com/docs/react/get-started/> (acedido em 24/05/2021).
- [5] Guy Bar-Gil. *NPM vs. Yarn: Which Package Manager Should You Choose?* Inglês. Ago. de 2020. URL: <https://www.whitesourcesoftware.com/free-developer-tools/blog/npm-vs-yarn-which-should-you-choose/> (acedido em 20/05/2021).
- [6] Cleber Campomori. *Precisamos falar sobre o TypeScript.* Português do Brasil. Abr. de 2017. URL: <https://www.treinaweb.com.br/blog/precisamos-falar-sobre-o-typescript/> (acedido em 17/03/2021).
- [7] Tomislav Capan. *Why The Hell Would I Use Node.js? A Case-by-Case Tutorial.* Inglês. TopTal. URL: <https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js> (acedido em 19/03/2021).
- [8] Mariana Clark. *GraphQL vs. REST | Quais são as diferenças?* Português do Brasil. URL: <https://blog.back4app.com/pt/graphql-vs-rest-quais-sao-as-diferencias/> (acedido em 04/06/2021).
- [9] Dockerize PostgreSQL. Inglês. Docker. URL: https://docs.docker.com/samples/postgresql_service/ (acedido em 02/06/2021).
- [10] Eric Elliott. *Unit Testing React Components.* Inglês. Mar. de 2019. URL: <https://medium.com/javascript-scene/unit-testing-react-components-aeda9a44aae2> (acedido em 06/06/2021).

- [11] Yomi Eluwande. *React End-to-End testing using Jest and Puppeteer*. Inglês. LogRocket. Mar. de 2021. URL: <https://blog.logrocket.com/react-end-to-end-testing-jest-puppeteer/> (acedido em 06/06/2021).
- [12] Alex Banks e Eve Porcello. *Learning React. Functional Web Development with React and Redux*. Inglês. O'Reilly, mai. de 2017. ISBN: 978-1-491-95462-1.
- [13] Diego Fernandes. *PWA: O que é? Vale a pena? Quando utilizar?* Português do Brasil. Rocketseat. 2018. URL: <https://blog.rocketseat.com.br/pwa-o-que-e-quando-utilizar/> (acedido em 20/03/2021).
- [14] Getting started with PostgreSQL on Linux. Inglês. RedHat. Mar. de 2019. URL: <https://www.redhat.com/sysadmin/postgresql-setup-use-cases> (acedido em 20/03/2021).
- [15] Abhinaba Ghosh. *React End-to-End testing made easy with Cypress*. Inglês. Mai. de 2020. URL: <https://medium.com/@abhinabaghosh.1994/react-testing-made-easy-with-cypress-262340597b08> (acedido em 06/06/2021).
- [16] GraphQL is the better REST. Inglês. How To GraphQL. URL: <https://www.howtographql.com/basics/1-graphql-is-the-better-rest/> (acedido em 04/06/2021).
- [17] Leigh Halliday. *Mocking and Testing GraphQL in React*. Inglês. Jul. de 2019. URL: <https://www.telerik.com/blogs/mock-and-test-graphql-in-react> (acedido em 11/06/2021).
- [18] How to Deploy PostgreSQL on Docker Container. Inglês. PhoenixNap. Jan. de 2020. URL: <https://phoenixnap.com/kb/deploy-postgresql-on-docker> (acedido em 02/06/2021).
- [19] João Inez. *GraphQL VS REST: Which one is better?* Inglês. Mai. de 2019. URL: <https://www.imaginarycloud.com/blog/graphql-vs-rest/> (acedido em 04/06/2021).
- [20] Introduction to Node.js. Inglês. NodeJS. URL: <https://nodejs.dev/learn> (acedido em 19/03/2021).
- [21] JB. *Publish React components as an npm package*. Inglês. Abr. de 2021. URL: <https://levelup.gitconnected.com/publish-react-components-as-an-npm-package-7a671a2fb7f> (acedido em 06/06/2021).
- [22] Pavels Jelisejevs. *Create React App: Get React Projects Ready Fast*. Inglês. SitePoint. Nov. de 2020. URL: <https://www.sitepoint.com/create-react-app/> (acedido em 15/03/2021).
- [23] Daniel Jun. *Using RESTful APIs versus GraphQL*. Inglês. Set. de 2019. URL: <https://medium.com/swlh/using-restful-apis-versus-graphql-1e6c350d56c9> (acedido em 04/06/2021).
- [24] Beingana Jim Junior. *How to create an npm library from React components*. Inglês. Dev.to. Fev. de 2021. URL: <https://dev.to/jimjunior/how-to-create-an-npm-library-from-react-components-2m2#creating-the-library> (acedido em 06/06/2021).
- [25] Peter Kayere. *Introduction to the Koa.js Framework*. Inglês. Set. de 2020. URL: <https://www.section.io/engineering-education/introduction-to-koajs/>.

- [26] Denis Kryukov. *Yarn vs. npm in 2019: Choosing the Right Package Manager for the Job*. Inglês. Jun. de 2019. URL: <https://soshace.com/yarn-package-manager-in-2019-should-we-keep-on-comparing-yarn-with-npm/> (acedido em 20/05/2021).
- [27] Jason Lai. *Building Frontend Applications By Mocking Your Entire API With Testing Tools*. Inglês. Set. de 2020. URL: <https://medium.com/swlh/building-frontend-applications-by-mocking-your-entire-api-with-testing-tools-2f050359677f> (acedido em 11/06/2021).
- [28] Gláucia Lemos. *Conhecendo TypeScript!* Português do Brasil. Jan. de 2017. URL: <https://medium.com/@glaucia86/conhecendo-typescript-8fd74e8d13fb> (acedido em 17/03/2021).
- [29] Lenon. *Node.js – O que é, como funciona e quais as vantagens*. Português do Brasil. Set. de 2018. URL: <https://www.opus-software.com.br/node-js/>.
- [30] Samuel Martins. *Por que usar TypeScript como linguagem de programação*. Português do Brasil. Nov. de 2018. URL: <https://take.net/blog/devs/por-que-usar-typescript> (acedido em 17/03/2021).
- [31] Modern Front-end Testing with Cypress. Inglês. Cypress. Fev. de 2019. URL: <https://www.cypress.io/blog/2019/02/05/modern-frontend-testing-with-cypress/> (acedido em 06/06/2021).
- [32] Node.js Introduction. Inglês. W3Schools. URL: https://www.w3schools.com/nodejs/nodejs_intro.asp (acedido em 19/03/2021).
- [33] O que é GraphQL? Português do Brasil. RedHat. URL: <https://www.redhat.com/pt-br/topics/api/what-is-graphql> (acedido em 19/03/2021).
- [34] O que é ReactJS e porque devemos usá-lo? Português. EDIT. Dez. de 2018. URL: <https://edit.com.pt/blog/o-que-e-reactjs-e-porque-devemos-usa-lo/> (acedido em 15/03/2021).
- [35] Elad Ossadon. *The Ultimate VSCode Setup for Front End/JS/React*. Inglês. Nov. de 2018. URL: <https://medium.com/productivity-freak/the-ultimate-vscode-setup-for-js-react-6a4f7bd51a2> (acedido em 20/03/2021).
- [36] Liz Parody. *Choosing the right Node.js Framework: Express, Koa, or Hapi?* Inglês. Mar. de 2019. URL: <https://nodesource.com/blog/Express-Koa-Hapi> (acedido em 26/05/2021).
- [37] Joe Previte. *How to Migrate a React App to TypeScript*. Inglês. SitePoint. Mar. de 2020. URL: <https://www.sitepoint.com/how-to-migrate-a-react-app-to-typescript/> (acedido em 15/03/2021).
- [38] Joe Previte. *React with TypeScript: Best Practices*. Inglês. SitePoint. Set. de 2020. URL: <https://www.sitepoint.com/react-with-typescript-best-practices/> (acedido em 15/03/2021).
- [39] PWA (Progressive Web App): O Que É e Como Criar o Seu (+3 Exemplos). Português do Brasil. Neil Patel. URL: <https://neilpatel.com.br/blog/pwa-o-que-e/> (acedido em 20/03/2021).

- [40] React – Testing Overview. Inglês. React. URL: <https://reactjs.org/docs/testing.html> (acedido em 06/06/2021).
- [41] React JS. Notes for Professionals. Inglês. GoalKicker.com, mai. de 2018. URL: <https://goalkicker.com/ReactJSBook/>.
- [42] Camilo Reyes. 20 Essential React Tools. Inglês. SitePoint. Out. de 2020. (Acedido em 15/03/2020).
- [43] Filipe Portela e Ricardo Queirós. *Introdução ao Desenvolvimento Moderno para Web. Do front-end ao back-end: uma visão global!* Português. FCA, 2018. ISBN: 978-972-722-897-3.
- [44] Drew Russell. GraphQL vs REST: What You Need to Know. Inglês. Nov. de 2019. URL: <https://www.rubrik.com/blog/technology/19/11/graphql-vs-rest-apis>.
- [45] Nick Scialli. Mocking Apollo GraphQL Queries in React Testing. Inglês. Fev. de 2021. URL: <https://typeofnan.dev/mockng-apollo-graphql-queries-in-react-testing/> (acedido em 11/06/2021).
- [46] Setting up a Node development environment. Inglês. Mozilla. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/development_environment (acedido em 19/03/2021).
- [47] Chris Sev. React Tools for VS Code. Inglês. Scotch. URL: <https://scotch.io/starters/react/editor-tools> (acedido em 20/03/2021).
- [48] Daniel de Souza Neris. A importância do TypeScript. Português do Brasil. Jul. de 2020. URL: <https://www.linkedin.com/pulse/import%C3%A2ncia-do-typescript-daniel-de-souza-neris/?articleId=6684939787595513856> (acedido em 17/03/2021).
- [49] Testim. React End-To-End Testing: A Guide to Getting Started. Inglês. Dez. de 2020. URL: <https://www.testim.io/blog/react-end-to-end-testing/> (acedido em 06/06/2021).
- [50] Testing. Apollo Client React testing API. Inglês. Apollo. URL: <https://www.apollographql.com/docs/react/api/react/testing/> (acedido em 11/06/2021).
- [51] Testing React Apps. Inglês. Jest. URL: <https://jestjs.io/docs/tutorial-react> (acedido em 06/06/2021).
- [52] Testing React components. Inglês. Apollo. URL: <https://www.apollographql.com/docs/react/development-testing/testing/> (acedido em 06/06/2021).
- [53] The Good and the Bad of Node.js Web App Development. Inglês. AltexSoft. Out. de 2019. URL: <https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-node-js-web-app-development/> (acedido em 19/03/2021).
- [54] The Good and the Bad of the TypeScript. Inglês. AltexSoft. Fev. de 2020. URL: <https://www.altexsoft.com/blog/typescript-pros-and-cons/> (acedido em 19/03/2021).
- [55] Michael Wanyoike. 10 Must-have VS Code Extensions for JavaScript Developers. Inglês. SitePoint. Abr. de 2020. URL: <https://www.sitepoint.com/vs-code-extensions-javascript-developers/> (acedido em 20/03/2021).

- [56] Michael Wanyoike. *Getting Started with React: A Beginner's Guide*. Inglês. SitePoint. Out. de 2020. URL: <https://www.sitepoint.com/getting-started-react-beginners-guide/> (acedido em 15/03/2021).
- [57] Michael Wanyoike. *How to Set Up VS Code for React Development*. Inglês. SitePoint. Jan. de 2021. URL: <https://www.sitepoint.com/vs-code-react-development/> (acedido em 16/03/2021).
- [58] *What is PostgreSQL?* Inglês. IBM. Ago. de 2019. URL: <https://www.ibm.com/cloud/learn/postgresql> (acedido em 20/03/2021).
- [59] *What is PostgreSQL?* Inglês. URL: <https://looker.com/databases/postgresql> (acedido em 20/03/2021).
- [60] Robien Wieruch. *The Road to GraphQL. Your journey to master pragmatic GraphQL in JavaScript with React.js and Node.js*. Inglês. Leanpub, 2018. URL: <https://leanpub.com/the-road-to-graphql> (acedido em 18/05/2021).
- [61] Prahlad Yer. *under_scores, camelCase and PascalCase - The three naming conventions every programmer should be aware of*. Inglês. Dev.to. Jul. de 2019. URL: <https://dev.to/prahladyeri/underscores-camelcasing-and-pascalcasing-the-three-naming-conventions-every-programmer-should-be-aware-of-3aed> (acedido em 13/03/2021).

Anexos

TypeScript

Instalação

A instalação do **TypeScript** pode ser realizada das seguintes maneiras:

- Globalmente:
 - **Com Yarn:** `yarn global add typescript`
 - **Com NPM:** `npm i -G typescript`
- Por Projeto:
 - **Com Yarn:** `yarn add -D typescript`
 - **Com NPM:** `npm i -D typescript`

A maneira mais comum é a instalação por projeto, visto que desta forma sempre que existir um *clone* do projeto e sejam instaladas as dependências¹², o **TypeScript** será também instalado e pronto a ser utilizado.

O uso de **TypeScript** pode implicar, em alguns casos, a instalação dos tipos (`@types`), por exemplo, no caso do **React** é necessário instalar os tipos recorrendo a `yarn add -D @types/react` ou `npm i -D @types/react`.

Nota

Como é possível analisar nos comandos de instalação do **TypeScript** por projeto, como na instalação dos tipos (`@types`), é usada a opção `-D` (tanto no uso do **Yarn** como do **NPM**), isto deve-se porque o **TypeScript** apenas será utilizado em desenvolvimento, uma vez que feito o *build* do projeto todo o código **TypeScript** é transformado em **JavaScript**.

¹² Recorrendo a `yarn install` ou `npm install`.

Configuração

O **TypeScript** permite realizar determinadas configurações no projeto, recorrendo para tal ao ficheiro **tsconfig.json**¹³. Neste ficheiro e, tal como é possível visualizar no excerto de código abaixo, é possível definir configurações relacionadas com a estrutura de pastas, qual a versão do ES a usar, entre outras configurações.

Além das configurações referidas anteriormente, entre muitas outras, é possível realizar a configuração de caminhos (*paths*), permitindo assim manter todas as importações realizadas durante o projeto mais “enxutas”. No excerto de código que se segue é possível analisar que foi criado um *path* para a pasta **components**, desta forma sempre que seja realizada a importação de um componente é possível utilizar o *path* **@components/** seguido do nome do componente.

```
1  {
2      "compilerOptions": {
3          "target": "es5",
4          "lib": [
5              "dom",
6              "dom.iterable",
7              "esnext"
8          ],
9          "allowJs": true,
10         "skipLibCheck": true,
11         "esModuleInterop": true,
12         "allowSyntheticDefaultImports": true,
13         "strict": true,
14         "forceConsistentCasingInFileNames": true,
15         "noFallthroughCasesInSwitch": true,
16         "module": "esnext",
17         "moduleResolution": "node",
18         "resolveJsonModule": true,
19         "isolatedModules": true,
20         "noEmit": true,
21         "jsx": "react",
22         "experimentalDecorators": true,
23         "baseUrl": "src",
24         "rootDir": "src",
25         "paths": {
26             "@components/*": [
27                 "src/components/*"
28             ]
29         }
30     },
31     "include": [
```

¹³[Documentação Oficial](#)

```
32     "src"  
33 ]  
34 }
```

Excerto de Código 11: TypeScript – Ficheiro `tsconfig.json`

Nota

O ficheiro `tsconfig.json` pode-se gerado automaticamente através dos comandos `npx tsc --init` ou então `yarn tsc --init`, sendo que este ficheiro gerado apenas trará todas as configurações possíveis, sendo necessário proceder posteriormente à sua correta configuração de acordo com o projeto em questão.

O ficheiro `tsconfig.json` apresentado tem como objetivo apresentar apenas uma possível estrutura de configuração. É recomendado consultar a [documentação oficial](#) relativa a este ficheiro.

É importante referir que este ficheiro deve encontrar-se na raiz do projeto para garantir o seu correto funcionamento.

React

Criação do Projeto

A criação de um projeto **React** pode ser realizada de duas formas, manualmente ou recorrendo ao `create-react-app`, porém será possível analisar abaixo como proceder à criação de ambas as formas.

Para a criação de um projeto **React** manualmente é necessário adicionar todos os packages ao ficheiro `package.json`, para isso os passos a seguir são:

1. Criação da pasta para o projeto;
2. Aceder à pasta criada anteriormente via terminal e executar o comando `npm init -y` ou `yarn init -y` (caso seja utilizado **Yarn** como package manager);
3. Adicionar todos os packages necessários, sendo eles (por norma):
 - **React** – `npm i react` ou `yarn add react`;
 - **React Dom** – `npm i react-dom` ou `yarn add react-dom`;
 - **React Scripts** – `npm i react-scripts` ou `yarn add react-scripts`.
4. Após a instalação dos packages é necessário proceder à criação dos scripts, para isso é necessário adicionar o seguinte código no ficheiro `package.json`:

```
1 "scripts": {  
2   "start": "react-scripts start",  
3   "build": "react-scripts build",  
4   "test": "react-scripts test",  
5   "eject": "react-scripts eject"  
6 },
```

Exerto de Código 12: Scripts para a execução do projeto em **React**

5. Posto isto é necessário criar todos os ficheiros necessários para a aplicação funcionar. Sendo eles:

- `index.html` (na pasta `public`)
- `index.css` (na pasta `src`)
- `index.jsx` (na pasta `src`)
- `App.jsx` (na pasta `src`)
- `App.css` (na pasta `src`)

Nota

É possível encontrar o código dos ficheiros referidos anteriormente em anexo (página 60) no ponto “**Ficheiros Iniciais**”.

É ainda importante referir que estes ficheiros são apenas a base para colocar um projeto **React** em funcionamento.

Porém como é possível analisar este processo é um pouco mais trabalhoso e implica que o programador saiba quais as dependências que necessita, para isso é possível usar o **create-react-app** que é o método recomendado pelo **React**¹⁴ para criar um projeto.

Os passos para a criação de um projeto seguindo este método são bastante simples e práticos, permitindo ainda ao programador definir se pretende usar ou não algum *template*, como por exemplo **TypeScript**. Os passos que se seguem demonstram a criação de um projeto **React** através desta “ferramenta”:

1. Em primeiro lugar é necessário instalar o **create-react-app**, isto pode ser realizado de duas formas de acordo com o *package manager* utilizado:

- **Com Yarn:** `yarn global add create-react-app`
- **Com NPM:** `npm install -g create-react-app`

2. Após a instalação é agora possível criar agora o projeto, para tal:

- **Com Yarn:** `yarn create react-app <project-name> [<options>]`
- **Com NPX:** `npx create-react-app <project-name> [<options>]`
- **Com NPM:** `npm init react-app <project-name> [<options>]`

Com isto é possível aceder à pasta do projeto (sendo a pasta o nome do projeto – `<project-name>`) e verificar que todos os packages foram adicionados, bem como os ficheiros base, inclusive o logo do **React** que irá aparecer como animação ao executar o projeto (ver figura abaixo).

¹⁴ Documentação: “[Create a new React App](#)”

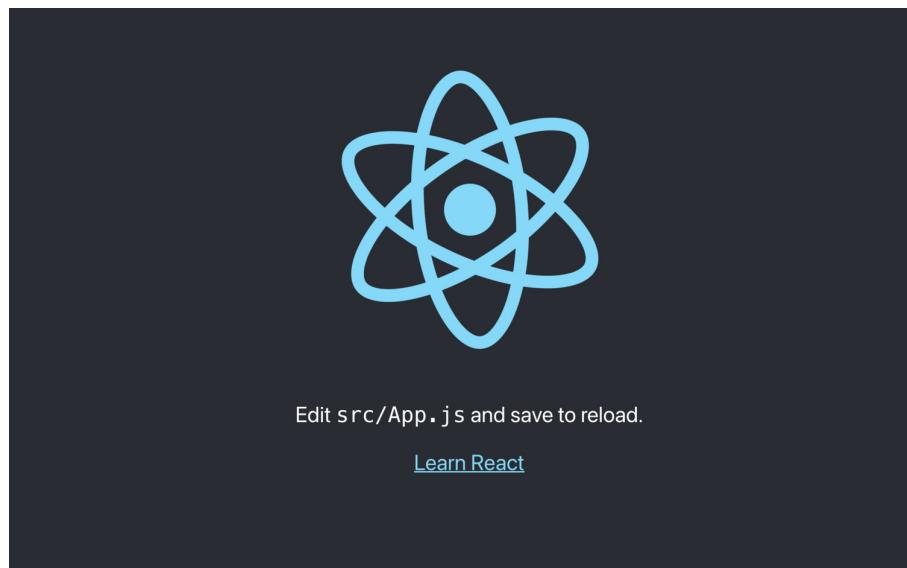


Figura 50: Página inicial do **React** após a execução do projeto

Opções Adicionais

Na criação de um projeto **React** através do `create-react-app` é possível especificar o *template* a usar, não sendo de uso obrigatório. A lista que se segue apresenta dois *templates* frequentemente utilizados:

- `--template typescript`: para gerar o projeto com **TypeScript**;
- `--template cra-template-pwa`: para gerar o projeto com a funcionalidade de PWA;
- `--template cra-template-pwa-typescript`: semelhante ao anterior, porém com **TypeScript**.

Além do *template* é ainda possível especificar o *package manager* utilizado, recorrendo à opção `--use-npm`, isto para usar o **NPM** como *package manager*¹⁵.

Referências adicionais: [1, 22, 2]

Estrutura de Pastas

A estrutura de pastas para um projeto **React** pode variar de projeto para projeto, ou da forma como o programador prefere organizar os mais diversos ficheiros do projeto. Porém e, tal como é possível analisar na figura que se segue, é comum encontrar a seguinte estrutura de pastas.

¹⁵No caso de possuir o **Yarn** instalado.

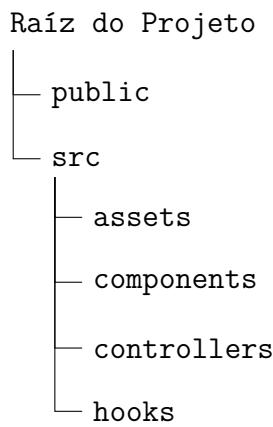


Figura 51: React – possível estrutura de pastas

Importante referir que a pasta `src/` será a pasta principal, sendo esta que irá conter todos os componentes, assets e outros ficheiros importantes para o projeto.

É importante referir que seguindo o método de criação do projeto **React** com o `create-react-app`, a estrutura de pastas e os ficheiros criados inicialmente será a seguinte:

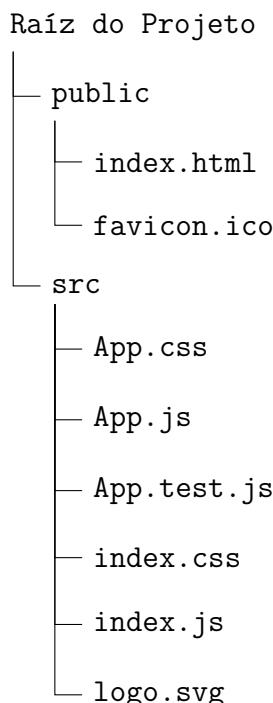


Figura 52: React – estrutura de pastas e ficheiros gerados pelo `create-react-app`

Nota

É importante relembrar que consoante o uso de **TypeScript** ou **JavaScript** será possível encontrar ficheiros `.tsx` ou `.ts`, `.jsx` ou `.js`.

Ficheiros Iniciais

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4      <head>
5          <meta charset="utf-8" />
6          <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
7          <meta name="viewport" content="width=device-width, initial-scale=1" />
8          <meta name="theme-color" content="#000000" />
9          <meta name="description" content="Web site created using create-react-app" />
10         <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
11         <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
12         <title>React App</title>
13     </head>
14
15     <body>
16         <noscript>You need to enable JavaScript to run this app.</noscript>
17         <div id="root"></div>
18     </body>
19
20 </html>
```

Excerto de Código 13: Ficheiro `index.html` de um projeto React

```
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import App from './App';
4 import './index.css';
5
6 ReactDOM.render(
7     <React.StrictMode>
8         <App />
9     </React.StrictMode>,
10    document.getElementById('root')
11 );
```

Excerto de Código 14: Ficheiro `index.jsx` de um projeto React

```
1 body {
2     margin: 0;
3     font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
4     'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
```

```
5     sans-serif;
6     -webkit-font-smoothing: antialiased;
7     -moz-osx-font-smoothing: grayscale;
8   }
9
10 code {
11   font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
12   monospace;
13 }
```

Excerto de Código 15: Ficheiro index.css de um projeto React

```
1 import React from 'react';
2 import './App.css';
3
4 function App() {
5   return (
6     <div className="App">
7       <header className="App-header">
8         <p>
9           Edit <code>src/App.jsx</code> and save to reload.
10        </p>
11        <a
12          className="App-link"
13          href="https://reactjs.org"
14          target="_blank"
15          rel="noopener noreferrer"
16        >
17           Learn React
18        </a>
19       </header>
20     </div>
21   );
22 }
23
24 export default App;
```

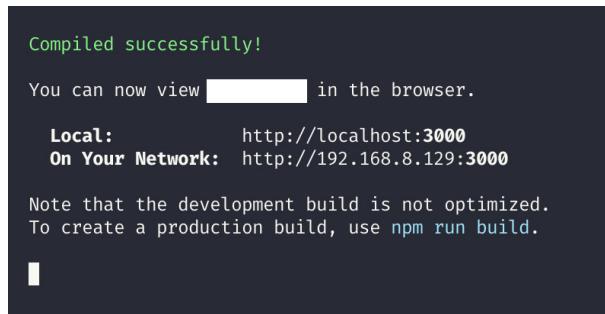
Excerto de Código 16: Ficheiro app.jsx de um projeto React

Execução do Projeto

Após a criação do projeto é agora possível executar o mesmo, para tal é possível utilizar os scripts presentes no ficheiro `package.json`, sendo apenas necessário recorrer a um dos comandos que se segue (de acordo com o package manager em uso):

- **Yarn:** `yarn start`
- **NPM:** `npm start`

Se tudo correr como esperado será apresentado a seguinte mensagem no terminal:



A screenshot of a terminal window showing the output of a successful React project build. The text in the terminal is as follows:

```
Compiled successfully!
You can now view [REDACTED] in the browser.
Local: http://localhost:3000
On Your Network: http://192.168.8.129:3000
Note that the development build is not optimized.
To create a production build, use npm run build.
```

Figura 53: Projeto **React** executado com sucesso

Nota

De notar que os comandos apresentados são para executar o projeto em modo de desenvolvimento, caso seja pretendido realizar o *build* para colocar o projeto em produção os comandos a executar são:

- **Com Yarn:** `yarn build`
- **Com NPM:** `npm run build`

Componentes Genéricos Desenvolvidos

Neste ponto é possível encontrar alguns dos componentes desenvolvidos e utilizados com mais frequência, apresentando os pontos com mais importância sobre os mesmos.

TextInput

O Componente **TextInput** é um componente destinado a ser utilizado nos diversos ecrãs do projeto. Por exemplo nos mockups apresentados no diagrama de sequência de inicio de sessão, onde este é utilizado para o campo de *email* e *password*.

Para isto ser possível o componente recebe as propriedades apresentadas no excerto de código que se segue.

```
1 const TextInput = ({
2   className,
```

```
3   disabled,
4   noValidate,
5   onChange,
6   type,
7   name,
8   placeholder,
9   multiline,
10  modifier,
11  rounded,
12  rows,
13  ...attributes
14 }: ITextInput) => {
15   // ...
16 }
```

Exerto de Código 17: Propriedades recebidas no componente `TextInput`

Assim sendo e, apesar de algumas das propriedades não serem obrigatórias, as que tem mais impacto no aspetto visual do *input* são a propriedade `rounded`, `multiline` and `modifier`.

A propriedade `rounded` tem como objetivo definir se o *input* conta com bordas redondas ou não. No caso desta propriedade ser aplicada, o *input* recebe uma *class* de **CSS** adicional (de forma a ser estilizada posteriormente), o exerto de código que segue apresenta a aplicabilidade desta propriedade, bem como o estilo adicionado.

```
1 // ...
2 <Tag
3   className={`${styles['${modifier}-style']} || ''} ${
4     rounded && styles['rounded']
5   }
6   {...attributes}
7   type={type}
8   name={name}
9   disabled={disabled}
10  placeholder='&nbsp;'
11  rows={rows ? Number(rows) : 0}
12  onChange={onChange}
13 >
14 // ...
15 </Tag>
16
17 // ...
```

Exerto de Código 18: Propriedade `rounded` aplicada ao *input*

```
1 .rounded  
2   border-radius: $inputBorderRadius // 25px
```

Exerto de Código 19: Estilo adicionado na class da propriedade `rounded`

Já no caso da propriedade `multiline`, esta destina-se a dizer se será apresentado um `input` ou `textarea`, para isso é utilizado o código abaixo:

```
1 // ...  
2  
3 const Tag: ElementType = multiline ? 'textarea' : 'input';  
4  
5 // ...
```

Exerto de Código 20: Utilização da propriedade `multiline` no componente `TextInput`

Por fim, a propriedade `modifier` tem como objetivo afetar as cores que são aplicadas ao `input`, no momento, para além do estilo padrão, a propriedade `modifier` pode receber o valor `light`, aplicando assim o seguinte estilo ao `input`.

```
1 &.light-style  
2   background-color: $whiteColor // #FFF  
3  
4   &:hover  
5     border: 1px solid $secondaryColor // #b8b7b7  
6  
7   &:focus  
8     outline: none  
9     border: 1px solid $secondaryColor // #b8b7b7
```

Exerto de Código 21: Estilo aplicado no uso da propriedade `modifier` com o valor `light`

Com estas propriedades o componente `TextInput` torna-se bastante versátil, possibilitando o seu uso nos mais diversos formulários do projeto.

Button

O componente `Button` é outros dos componentes presente em diversos ecrãs da aplicação, porém com diferentes estilos. Assim este componente recebe, tal como no componente anterior, propriedades que permitem tornar este componente mais versátil.

```
1 const Button: React.FC<Props> = ({  
2   icon,  
3   className,  
4   rounded,  
5   to,  
6   type,  
7   modifier,  
8   children,  
9   onClick,  
10  ...attributes  
11}) => {  
12   // ..  
13 }
```

Exerto de Código 22: Propriedades recebidas no componente **Button**

Neste ponto serão apenas apresentadas as propriedades **rounded** e **modifier**, visto serem as que causa maior impacto na aparência do componente. De referir que caso este receba a propriedade **to**, este utiliza um **<Link>** em vez de **<button>**, para isso:

```
1 const Tag: ElementType = to ? Link : 'button';  
2 const tagAttributes = to ? { to } : { type };
```

Exerto de Código 23: Uso de **Link** ou **button** no componente **Button**

De referir que para a propriedade **modifier** (que pode receber os valores: '**alt-primary**', '**info**', '**danger**') ou da propriedade **rounded** ter impacto, é definida uma class **CSS** adicional, no exerto de código que segue é possível analisar a aplicação dessa(s) class(es).

```
1 <Tag  
2   {...tagAttributes}  
3   {...attributes}  
4   onClick={onClick}  
5   className={`  
6     ${styles['root']} || ''}  
7     ${styles[`_${modifier}-modifier`]} || ''}  
8     ${rounded && styles['rounded']} || ''}  
9     ${className || ''}`}  
10  ></Tag>
```

Exerto de Código 24: Aplicação de classes adicionais para as propriedades **rounded** e **modifier**

Por sua vez, no estilo do componente é possível encontrar o seguinte estilo:

```
1 .alt-primary
2   background-color: $whiteColor
3   color: $buttonHighlightColor
4
5 &:disabled
6   color: $whiteColor
7   background-color: $secondaryColor
8   box-shadow: 0px 1px 2px #00000029
9
10 &:enabled
11   &:hover
12     box-shadow: 0px 5px 6px #00000029
13
14   &:focus
15     background-color: $whiteColor
16     color: $buttonHighlightColor
17     box-shadow: 0px 0px 7px #13ACCC
18
19   &:active
20     background-color: $whiteColor
21     color: $buttonHighlightColor
22     border: 1px solid #D3D3D3
23     box-shadow: none
24
25
26 .info-modifier
27   background: $appBackgroundColor
28   color: $primaryColor
29
30 &:disabled
31   background-color: $secondaryColor
32   color: $whiteColor
33
34 &:enabled
35   &:hover
36     background-color: $appBackgroundColor
37     color: #707070
38     box-shadow: 0px 5px 6px #00000029
39
40   &:focus
41     background-color: $appBackgroundColor
42     color: $primaryColor
43     box-shadow: 0px 0px 7px #13ACCC
44
45   &:active
```

```

46   color: $primaryColor
47   background-color: $appBackgroundColor
48   box-shadow: none
49
50 .danger-modifier
51   background: $dangerColor
52   color: $appForegroundColor
53
54 .rounded
55   border-radius: calc(${buttonHeight} / 2)
56   padding: 0 1em

```

Exerto de Código 25: Estilo aplicado para as propriedades `modifier` e `rounded`

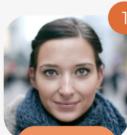
Importante referir que no caso da propriedade `modifier` não ser aplicada, o componente **Button** conta com um estilo de cores e sombras padrão.

Avatar

O componente **Avatar** é encontrado nos mais diversos ecrãs do projeto, contando com várias possibilidades de apresentação, tal como é possível analisar nas figuras que se seguem.

LISTA DE ATLETAS

Pesquisa



#00000

Ana Rita Santos
Moreira Santos



#00000

Ana Rita Santos
Moreira Santos

LISTA DE ATLETAS

Pesquisa



Maria Ferreira da Conceição

Natação Base | Base Feminino, Nível 2



DADOS PESSOAIS



MICROCICLOS

Figura 54: Componente **Avatar** na página de perfil do atleta

Figura 55: Componente **Avatar** na vista em mosaico na lista de atletas

Tag	Nome	Genero	Modalidade	Tipo de Plano	Estado da Subscrição	Objectivo
	Ana Maria Rita Moreira dos Santos	Feminino	Natação	Base	Ativo	<small> Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen </small>

Figura 56: Componente **Avatar** na lista de atletas

De uma forma resumida, o componente **Avatar** pode ser apenas a imagem do atleta, a imagem

do atleta com o número de notificações, a imagem de atleta e nome do alteta, imagem do atleta e identificador, ou então, a junção de todas estas possibilidades.

```
1 const Avatar = ({  
2   className,  
3   avatar,  
4   notifications,  
5   label,  
6   ...attributes  
7 }: IAvatarProps) => {  
8   return (  
9     <div  
10       className={`${styles["root"]} || ""} ${className || ""}`  
11       {...attributes}  
12     >  
13     <div  
14       className={styles["avatar"]}  
15       style={({  
16         backgroundImage: avatar ? `url(${avatar})` : undefined  
17       })}  
18     >  
19     {label && <span className={styles["label"]}>{label}</span>}  
20  
21     {notifications && (  
22       <span className={styles["badge"]}>{notifications}</span>  
23     )}  
24   </div>  
25 </div>  
26 );  
27 };  
28  
29 export default Avatar;
```

Exerto de Código 26: Código desenvolvido para o componente **Avatar**

Como é possível analisar pelo excerto de código anterior, o componente **Avatar** recebe determinadas propriedades, podendo algumas ser ou não recebidas, permitindo assim criar as várias vertentes apresentadas nas figuras.

SearchInput

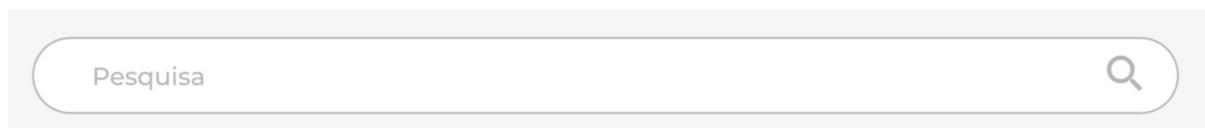


Figura 57: Aparência do componente **SearchInput**

O componente **SearchInput** é outro dos componentes que é visível nos demais ecrãs do projeto, sendo este um *input* de **HTML** normal, com um estilo adicional e o uso de um ícone no formato **SVG**.

De seguida é possível analisar o código produzido para este componente e, é possível notar que, em comparação aos componentes anteriores, este recebe menos propriedades, sendo a propriedade **onChange** responsável por executar a pesquisa sempre que existe uma alteração no valor do mesmo.

```
1 const SearchInput = ({  
2   className,  
3   onChange,  
4   ...attributes  
5 }: ISearchInputProps) => {  
6   return (  
7     <label className={`${styles["root"]} ${className || ""}`}>  
8       <input type="text" {...attributes} onChange={onChange} />  
9  
10      <Icon name="search" width="20px" height="20px" />  
11    </label>  
12  );  
13};  
14  
15 export default SearchInput;
```

Exerto de Código 27: Código do componente **SearchInput**

Para facilitar a realização da pesquisa foi utilizado um package adicional, o **Fuse.js**. A utilização deste package é bastante simples e pode ser analisada no excerto de código que se segue.

```
1 import SearchInput from 'components/inputs/SearchInput/SearchInput';  
2 import Fuse from 'fuse.js';  
3  
4 // ...  
5  
6 const [data, setData] = useState(athletes);  
7  
8 // Search  
9 const options = {  
10   keys: ['name'],  
11};  
12  
13 const fuse = new Fuse(athletes, options);  
14  
15 const onSearchChange = (evt: ChangeEvent<HTMLInputElement>) => {  
16   if (evt.target.value !== '') {  
17     setData(fuse.search(evt.target.value).map((fuseItem) => fuseItem.item));  
18   }  
19};
```

```
18     } else {
19         setData(athletesData);
20     }
21 };
22
23 return (
24     // ...
25     <SearchInput placeholder='Pesquisa' onChange={onSearchChange} />
26 );
```

Exerto de Código 28: Uso do componente **SearchInput** em conjunto com o package **Fuse.js**

Como é possível analisar, o package **Fuse.js** recebe inicialmente uma lista de valores e um objeto com as opções. Neste objeto de opções são definidas as "keys" (chaves) nas quais serão realizada a pesquisa.

É ainda possível encontrar a função que realiza a pesquisa sempre que o valor do **SearchInput** muda, executando o método **setData** (do **useState()** do **React**), mas com os valores filtrados pelo **Fuse.js**.

Sass

Instalação

A instalação do **Sass** normalmente é realizada através do **NPM** ou **Yarn**, porém existem outras formas de realizar a Instalação, por exemplo em **macOS** é possível instalar recorrendo ao **HomeBrew**.

Ainda assim, com o **NPM** ou **Yarn**, este pode ser instalado globalmente ou apenas instalado em cada projeto de forma individual. Os comandos que se seguem apresentam as duas formas de Instalação com **Yarn** e **NPM**.

- **Globalmente:**
 - **Com NPM:** `npm install -g sass;`
 - **Com Yarn:** `yarn global add sass;`
- **Por Projeto:**
 - **Com NPM:** `npm install sass;`
 - **Com Yarn:** `yarn add sass;`

Mixins

As mixins no **Sass** permitem reutilizar estilo, poupando tempo e aplicando o conceito de **DRY**, ou seja, *Don't Repeat Yourself*. No excerto de código que se segue é possível analisar a definição de uma mixin, bem como a utilização da mesma.

```
1 =flex-settings($direction: row)
2   display: flex
3   flex-direction: $direction
4
5 .my-row
6   +flex-settings()
7
8 .my-column-row
9   +flex-settings(column)
```

Exerto de Código 29: Definição e uso de mixins no **Sass**

Tal como é possível analisar no excerto de código anterior, as mixins podem receber parâmetros, sendo que estes parâmetros podem assumir um valor por defeito. Ou seja, a mixin **flex-settings** pode receber ou não a direção (**direction**), sendo o valor por defeito **row**.

Na linha 6 e 9 é possível analisar a utilização desta *mixin*, bem como a alteração do valor do parâmetro `direction` para `column`.

Nota

A declaração de uma *mixin* em **Sass** é realizada através do símbolo `=`, seguido do nome da *mixin* e entre parêntesis o(s) parâmetro(s). O uso desta é realizado através do símbolo `+`, seguido do nome e parâmetros caso possua.

Caso seja usada a variante `.scss`, a declaração de *mixins* é feita através de `@mixin` e o seu uso através de `@include`.

Herança

No **Sass** também é possível realizar herança, neste caso herança de estilos. O excerto de código¹⁶ que é apresentado de seguida demonstra a utilização da herança no **Sass**.

```
1 .error
2   border: 1px #f00
3   background-color: #fdd
4
5 &--serious
6   @extend .error
7   border-width: 3px
```

Exerto de Código 30: Demonstração de herança no **Sass**

No excerto de código abaixo é possível analisar o resultado final após este ser compilado para um ficheiro **CSS**.

```
1 .error,
2 .error--serious {
3   border: 1px #f00;
4   background-color: #fdd;
5 }
6
7 .error--serious {
8   border-width: 3px;
9 }
```

Exerto de Código 31: Código **CSS** resultante da compilação do excerto de código anterior

¹⁶Retirado da [documentação oficial](#).

Além da herança através de class's é possível recorrer a placeholders. Placeholders funcionam como uma class, porém começa com % e não são incluídos no código [CSS](#) resultante.

```
1  %toolbelt
2    box-sizing: border-box
3    border-top: 1px rgba(#000, .12) solid
4    padding: 16px 0
5    width: 100%
6
7    &:hover
8      border: 2px rgba(#000, .5) solid
9
10   .action-buttons
11     @extend %toolbelt
12     color: #4285f4
13
14
15   .reset-buttons
16     @extend %toolbelt
17     color: #cddc39
```

Exerto de Código 32: Demonstração de placeholders em [Sass](#)¹⁷

Sendo que após este código ser compilado, o placeholder apresentado não estará no código [CSS](#) resultante, tal como é possível analisar abaixo.

```
1  .action-buttons, .reset-buttons {
2    box-sizing: border-box;
3    border-top: 1px rgba(0, 0, 0, 0.12) solid;
4    padding: 16px 0;
5    width: 100%;
6  }
7
8  .action-buttons:hover, .reset-buttons:hover {
9    border: 2px rgba(0, 0, 0, 0.5) solid;
10 }
11
12 .action-buttons {
13   color: #4285f4;
14 }
15
16 .reset-buttons {
17   color: #cddc39;
18 }
```

Exerto de Código 33: Código **CSS** resultante da compilação do excerto de código com placeholder

Variáveis

No **Sass** é possível declarar variáveis recorrendo ao símbolo \$ seguido do nome pretendido. Nos excertos de código que se seguem é possível analisar a declaração de variáveis, o seu uso e qual o resultado após este ser compilado para **CSS**.

```
1 $padding: 10px 20px
2 $defaultColor: #ca4d24
3
4 .alert
5   background-color: $defaultColor
6   padding: $padding
```

Exerto de Código 34: Utilização de variáveis em **Sass**

No **CSS** estas variáveis não são visíveis, uma vez que o valor destas serão apresentadas diretamente na linha da sua utilização, ou seja:

```
1 .alert {
2   background-color: #ca4d24;
3   padding: 10px 20px;
4 }
```

Exerto de Código 35: Código **CSS** resultante do excerto de código com variáveis em **Sass**

Porém em **CSS** também é possível utilizar variáveis, porém estas são definidas recorrendo a :root {}.

No excerto de código que se segue é apresentado um exemplo de variáveis em **CSS**.

```
1 :root {
2   --padding: 10px 20px;
3   --default-color: #ca4d24
4 }
5
6 .alert {
7   padding: var(--padding);
8   background-color: var(--default-color);
9 }
```

Exerto de Código 36: Declaração e uso de variáveis em **CSS**

GraphQL

Instalação

A instalação do **GraphQL** pode ser realizada através do **NPM** ou do **Yarn**, para isso basta recorrer a um dos seguintes comandos:

- **Com Yarn:** `yarn add graphql`
- **Com NPM:** `npm install graphql`

Desta forma o **GraphQL** está disponível para utilizar ao longo do projeto recorrendo a uma das seguintes formas apresentadas abaixo.

```
1 const { graphql, buildSchema } = require('graphql');
```

Excerto de Código 37: Importação do GraphQL em JavaScript

```
1 import { graphql, buildSchema } from 'graphql';
```

Excerto de Código 38: Importação do GraphQL em TypeScript

Apollo Client

O **Apollo Client** permite realizar queries no lado do servidor (*back-end*), mas sendo estas executadas no lado do cliente, o *front-end*.

Em primeiro lugar é necessário realizar a instalação do **Apollo Client**, para isso:

- **Com Yarn:** `yarn add @apollo/client;`
- **Com NPM:** `npm i @apollo/client`

Nos tópicos que se seguem é possível analisar em mais detalhe a criação de um cliente, bem como a realização de uma query.

Criação de um client

A criação do *client* do **Apollo Client** tem como princípio definir a que *url* serão realizadas as *queries*, no caso o *url* da API.

O exemplo que se segue foi retirado da [documentação oficial do Apollo Client](#).

```
1 import { ApolloClient, InMemoryCache } from '@apollo/client';
2
3 const client = new ApolloClient({
4   uri: '<api-url>',
5   cache: new InMemoryCache(),
6 });
```

Exerto de Código 39: Criação de um client recorrendo ao **Apollo Client** no **React**

Execução de queries

A execução de uma query no **Apollo Client** implica que já exista um *client* criado, tal como foi apresentado no tópico anterior. Desta forma, é utilizada a variável criada (**const client**) e o método **query**. Novamente este exemplo pode ser encontrado na [documentação oficial do Apollo Client](#).

```
1 import { gql } from '@apollo/client';
2
3 // const client = ...
4
5 client
6   .query({
7     query: gql`  

8       query GetRates {  

9         rates(currency: "USD") {  

10           currency  

11         }  

12       }  

13     `,
14   })
15   .then((result) => console.log(result));
```

Exerto de Código 40: Execução de uma query recorrendo ao **Apollo Client** no **React**

Comparação entre GraphQL e REST

O **GraphQL** foi desenvolvido em 2015 pelo **Facebook**, tendo como principal motivação a resolução de problemas que surgiram com o aumento do uso de dispositivos móveis.

As principais diferenças entre **REST** e **GraphQL** são verificadas na figura¹⁸ que segue, uma vez que em **REST**, os dados são obtidos através de múltiplos endpoints. Já em **GraphQL** apenas é necessário realizar um único pedido para obter todos os dados, combatendo assim o problema de *over-fetching* e *under-fetching* de dados.

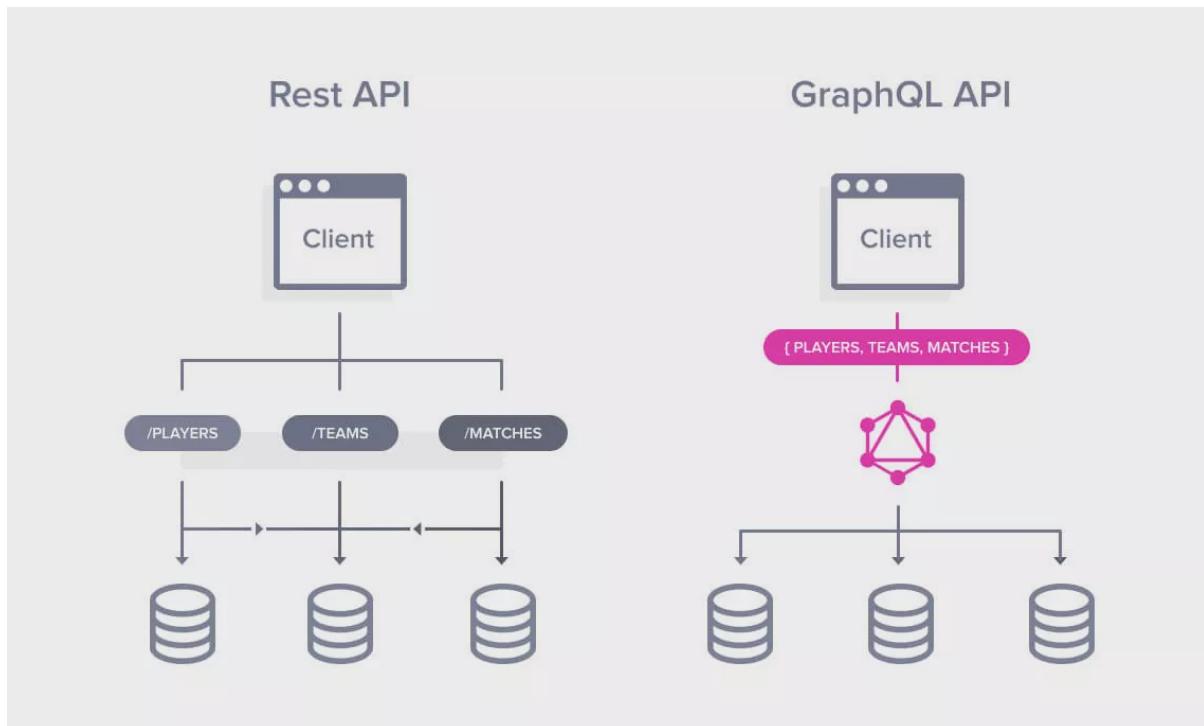


Figura 58: Comparação entre **GraphQL** e **REST**

Over-fetching consiste na obtenção de dados em excesso, o que é provável acontecer num pedido **REST**, uma vez que a informação devolvida por esta possa conter dados que não sejam necessários. Já o *under-fetching* é o inverso, ou seja, a falta de dados.

Em **GraphQL** estes problemas são combatidos, uma vez que o cliente tem o poder de escolher apenas os dados que necessita que a **API** lhe devolva. O excerto de código que se segue apresenta uma query¹⁹ em **GraphQL** onde é possível analisar a seleção dos campos escolhidos para serem devolvidos.

```
1 {  
2   hero {
```

¹⁸ Retirado de: [23]

¹⁹ Retirado da [documentação oficial](#)

```
3     name
4     # Queries can have comments!
5   friends {
6     name
7   }
8 }
9 }
```

Excerto de Código 41: GraphQL – Exemplo de query com seleção de campos

Como é possível analisar no exemplo apresentado são selecionados os campos **name** do schema **hero**, bem como ainda o campo **name** do schema **friends** que se encontram relacionados com o **hero**. Ou seja, o funcionamento é como um **JOIN** em base de dados.

Esta é apenas uma das vantagens em comparação a **REST** e, importa referir, que os problemas de *over-fetching* e *under-fetching* também podem ser resolvidos, sendo necessário recorrer a outras metodologias.

Porém, o **GraphQL** também conta com pontos negativos, como por exemplo:

- **Error Handling:** em **GraphQL** independentemente de a query ser realizado com sucesso ou não, é sempre devolvido o *HTTP Status Code 200*²⁰;
- **Complexidade:** em **GraphQL** a complexidade é maior, exigindo logo de inicio o desenvolvimento sobre **SDL**, ou seja, **Schema Definition Language**. De uma forma geral, a curva de aprendizagem em **GraphQL** é mais acentuada;
- **Web Caching:** realizar *caching* em **GraphQL** torna-se mais complexo, visto não usar *HTTP Caching*. É possível implementar *caching* recorrendo ao **Apollo Server**, porém a persistência de queries pode ser um problema.

Estes são apenas alguns pontos comparativos em relação a **REST** e **GraphQL**, existindo outros. Como tal, são deixadas algumas referências relativas a este tópico: [16, 19, 8, 44, 33, 23]

²⁰ [HTTP Status Code 200 – Ok \(Mais Informações\)](#)

Yarn

Instalação

A instalação do **Yarn** requer que o **NodeJS** esteja já instalado no dispositivo em questão. É possível encontrar no anexo relativo à instalação [página 81 e 82](#) nos diversos sistemas operativos.

Linux/Windows/macOS

O **Yarn** pode ser instalado em qualquer dos sistemas operativos recorrendo ao **NPM**, bastando apenas executar o comando `npm install --global yarn`.

Com este comando o **Yarn** será instalado globalmente no dispositivo sendo possível verificar se a instalação foi bem sucedida recorrendo ao comando `yarn --version`, apresentando assim a versão do **Yarn** instalada no dispositivo.

macOS

A instalação do **Yarn** no **macOS** pode ser realizada das seguintes formas:

- **Via HomeBrew:** `brew install yarn`;
- **Via Script:** `curl -o- -L https://yarnpkg.com/install.sh | bash`;

Nota

No caso da execução do **Yarn** apresentar `yarn command not found`, significa que é necessário realizar a configuração do path/caminho no `.bash_profile`, `.bashrc` ou `.zshrc`.

Para definir o path basta adicionar a seguinte linha num dos ficheiros mencionados: `export PATH="$PATH:$yarn global bin"`.

Linux

A instalação do **Yarn** no **Linux**²¹ pode ser realizada através dos *Debian Packages*, para isso:

- `curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -`
`echo "deb https://dl.yarnpkg.com/debian/ stable main";`
- `sudo tee /etc/apt/sources.list.d/yarn.list;`

²¹Os comandos apresentados são para distribuições com base **Debian**

```
· sudo apt update && sudo apt install yarn.
```

Nota

No caso da execução do **Yarn** apresentar **yarn command not found**, significa que é necessário realizar a configuração do *path/caminho* no **.bash_profile**, **.bashrc** ou **.zshrc**.

Para definir o *path* basta adicionar a seguinte linha num dos ficheiros mencionados:
export PATH="\$PATH:\$yarn global bin".

NodeJs

Instalação

macOS

A instalação do **NodeJS** no macOS pode ser realizada de diversas formas, as que são apresentadas de seguida são apenas algumas das soluções existentes.

- **Via HomeBrew:** `brew install node;`
- **Via NVM:** `nvm install --lts`

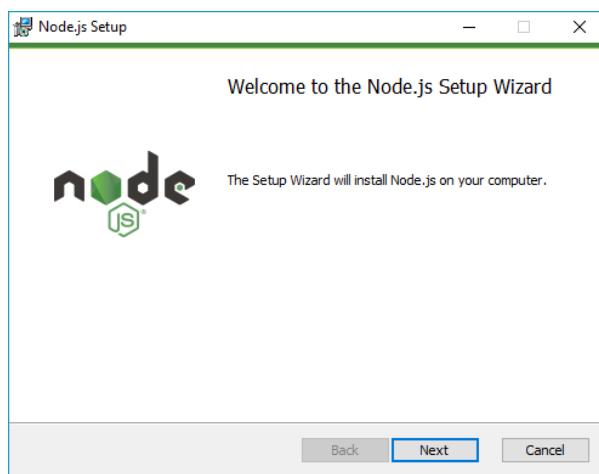
De referir que o NVM²² é uma ferramenta bastante útil para os utilizadores que necessitam de mais do que uma versão do **NodeJS**, conseguindo posteriormente realizar a atribuição de uma versão específica a cada projeto, recorrendo para tal ao comando `nvm use <version>`

Linux

Para realizar a instalação do **NodeJS** em Linux é²³ bastante simples, recorrendo para tal ao comando `sudo apt install nodejs`, ou novamente, recorrendo ao NVM como foi apresentado anteriormente.

Windows

Para realizar a instalação do **NodeJS** no Windows basta fazer o download do instalador (ficheiro `.exe`) e seguir os passos apresentados, a imagem que se segue apresenta o ecrã inicial da instalação do **NodeJS**.



²²[Repositório oficial do NVM](#)

²³**Nota:** os comandos apresentados são para distribuições baseadas em Debian

Figura 59: Ecrã inicial da instalação do **NodeJS** no Windows

Arquitetura

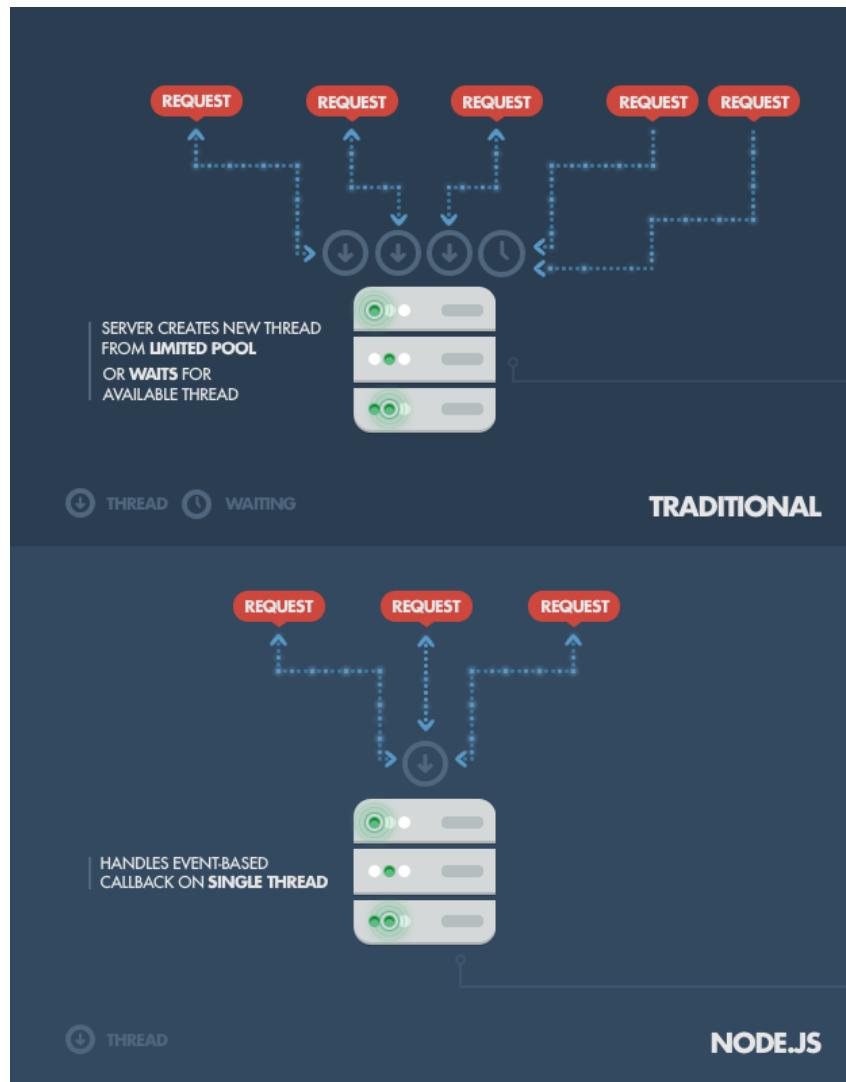


Figura 60: Arquitetura do **NodeJS** em comparação com a arquitetura tradicional

PostgreSQL

Instalação

Container Docker

Uma das formas de instalar o **PostgreSQL** é através de um container **Docker**²⁴, sendo apenas necessário possuir este instalado na máquina em questão.

Após possuir o **Docker**, basta executar no terminal o seguinte comando: `docker run --name postgres --network=postgres-network -e "POSTGRES_PASSWORD=<your-password>" -p 5432:5432 -v ~/Documents/containers/postgres:/var/lib/postgresql/data -d postgres`. Importante referir que o caminho `/Documents/containers/postgres/` é uma pasta criada para armazenar informações do container.

Nota

A porta pela qual é possível aceder ao **PostgreSQL** é também definida no comando de criação do container. Caso seja pretendido o uso de uma porta diferente basta alterar a porta depois dos dois pontos (:), por exemplo:

- **Porta padrão:** ... -p 5432:5432 ...;
- **Porta personalizada:** ... -p 5432:1234

Garantir que a porta personalizada desejada não se encontra já em uso por outra aplicação.

Windows

A instalação do **PostgreSQL** no **Windows** resume-se essencialmente ao download do ficheiro `.exe` no [site oficial](#), iniciando o executável posteriormente e seguir todos os passos apresentados semelhante à instalação do **NodeJS** neste sistema operativo.

macOS

No **macOS** o **PostgreSQL** pode ser instalado através da imagem `.dmg` baixada através do [site oficial](#), ou então através do **HomeBrew**. Para instalar através do **HomeBrew** basta executar o comando: `brew install postgresql`.

²⁴Referências recomendadas: [9,18]

Linux

Para realizar a instalação do **PostgreSQL** no **Linux**²⁵ é necessário executar os seguintes comandos²⁶:

- wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -
- sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt/ `lsb_release -cs`-pgdg main" >> /etc/apt/sources.list.d/pgdg.list'
- sudo apt-get update
- sudo apt-get -y install postgresql

²⁵Os comandos apresentados são para distribuições com base **Debian**

²⁶Comandos retirados do [site oficial](#)

Testes

Devido ao tempo limitado não foi possível realizar testes a todo o projeto, porém foi possível testar o seu funcionamento recorrendo ao **Jest** e o package incluído no **React** para testes. Desta forma foi possível realizar alguns testes unitários a dois componentes, realizando algumas validações através das propriedades que estes recebem. Além destes dois testes, foi possível testar como realizar o preenchimento de *inputs*, simulando a interação por parte do utilizador.

```
1 import "@testing-library/jest-dom";
2 import { cleanup, render } from "@testing-library/react";
3 import ReactDOM from "react-dom";
4 import Button from "./Button";
5
6 beforeEach(cleanup);
7
8 it("Renders Button without crashing", () => {
9   const div = document.createElement("div");
10  ReactDOM.render(<Button>Test Button</Button>, div);
11 });
12
13 it("Renders Button with info modifier", () => {
14   const { getByText } = render(
15     <Button modifier="info">Test with Modifier</Button>
16   );
17
18   // Verify from class added from property
19   expect(getByText("Test with Modifier")).toHaveClass("info-modifier");
20 });
```

Excerto de Código 42: Testes realizados ao componente *Button*

```
1 import "@testing-library/jest-dom";
2 import { cleanup, render } from "@testing-library/react";
3 import ReactDOM from "react-dom";
4 import StatusBadge from "./StatusBadge";
5
6 beforeEach(cleanup);
7
8 it("Renders Status Badge without crashing", () => {
9   const div = document.createElement("div");
10  ReactDOM.render(<StatusBadge status="Ativo" />, div);
11});
```

```

13 it("Renders Status Badge with active class text", () => {
14   const { getByTestId } = render(<StatusBadge status="Ativo" />);
15
16   // Have class
17   expect(getByTestId("status-badge")).toHaveClass("active-status");
18 });
19
20 it("Renders Status Badge with Ativo text", () => {
21   const { getByTestId } = render(<StatusBadge status="Ativo" />);
22
23   // Have Ativo text
24   expect(getByTestId("status-badge")).toHaveTextContent("Ativo");
25 });

```

Excerto de Código 43: Testes realizados ao componente **Status Badge**

O resultado dos testes apresentados acima seria:

```

PASS  src/components/status/StatusBadge/StatusBadge.test.tsx
PASS  src/components/buttons/Button/Button.test.tsx

Test Suites: 2 passed, 2 total
Tests:       5 passed, 5 total
Snapshots:  0 total
Time:        1.787 s, estimated 2 s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.

```

Figura 61: Execução dos testes com sucesso

Se no caso do teste realizado ao componente **Status Badge**, na linha 17 (que se encontra sombreada), o valor "Ativo" fosse alterado, o teste iria dar erro, visto este não conter assim a class **active-status**. Para esse erro seria apresentado a mensagem presente na figura que se segue.

```

PASS  src/components/status/StatusBadge/StatusBadge.test.tsx
FAIL  src/components/buttons/Button/Button.test.tsx
● Renders Button with info modifier

  expect(element).toHaveClass("info-modifier")

  Expected the element to have class:
    info-modifier
  Received:
    root alt-primary-modifier

    16 |     );
    17 |
    > 18 |     expect(getByText("Test with Modifier")).toHaveClass("info-modifier");
    19 |   );
    20 |

      at Object.<anonymous> (src/components/buttons/Button/Button.test.tsx:18:42)

Test Suites: 1 failed, 1 passed, 2 total
Tests:       1 failed, 4 passed, 5 total
Snapshots:  0 total
Time:        1.891 s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.

```

Figura 62: Execução dos testes com erro no componente **Status Badge**

Outro ponto que é necessário ter em consideração e, tal como é possível analisar nos testes apresentados, a “pesquisa” do elemento é feito pelo `getById`, sendo que para tal é necessário adicionar no código do componente o `data-testid="<id>"`:

```
1 return (
2   <div
3     data-testid="status-badge"
4     {...attributes}
5     className={`${styles["root"]} ${className} ${`${
6       styles[`#${statusColor}-status`]
7     }`}
8   >
9     {status}
10    </div>
11 );
```

Exerto de Código 44: Código com o `data-testid` definido no componente **Status Badge**

O exerto de código que se segue, apresenta um teste onde é realizada a verificação do valor do componente **TextInput** no ecrã de inicio de sessão, ou seja, numa primeira fase é inserido o valor no campo de texto (no caso no campo `email`) e posteriormente é validado o seu valor, simulando assim a interação que é realizada pelo utilizador.

```
1 import { render } from "@testing-library/react";
2 import userEvent from "@testing-library/user-event";
3 import Login from "./Login";
4
5 it("Input have email", () => {
6   const onSuccess = (token: string, remember: boolean) => {
7     console.log(token);
8   };
9
10  const div = document.createElement("div");
11  const component = render(<Login onLoginSuccess={onSuccess}> /);
12
13  const emailInput = component.getByTestId("email-input");
14  userEvent.paste(emailInput, "email@example.pt");
15
16  expect(emailInput.getAttribute("value")).toBe("email@example.pt");
17});
```

Exerto de Código 45: Teste com preenchimento de inputs

Seria ainda possível realizar testes com dados falsos, ou chamados de *mock data*, recorrendo para tal ao package `@testing-library/react`. Para tal, seria simulado um pedido a [API](#), no caso do projeto seria através do **Apollo Client**, onde é esperada uma resposta.

Na documentação do **Apollo Client**²⁷ é possível encontrar exemplos de como utilizar as ferramentas já incluídas na instalação do mesmo, recorrendo para tal ao **MockedProvider**, um elemento que recebe uma lista de **mocks**, ou seja, respostas e dados factícios.

No caso do inicio de sessão existiriam duas possibilidades, o inicio de sessão com sucesso, onde era necessário testar o redirecionamento por parte do **React Router**, ou então, o inicio de sessão inválido, onde é apresentada uma mensagem de erro ao utilizador.

A pergunta pode ser, mas qual a razão pela qual usar dados factícios? Num caso real de um inicio de sessão onde é utilizado *email* e *password*, basta o utilizador alterar a *password* para os testes falharem, porém a funcionalidade de inicio de sessão continua a funcionar. Com os *mock data* isto não acontece, visto simular um pedido factício à [API](#).

²⁷ Documentação oficial e outras referências: [52, 45, 50, 27, 17]

Visual Studio Code

Configurações

```
1  {
2    "[javascript)": {
3      "editor.defaultFormatter": "esbenp.prettier-vscode",
4      "editor.formatOnPaste": true,
5      "editor.formatOnType": true,
6      "editor.tabSize": 4,
7      "editor.detectIndentation": false,
8      "editor.insertSpaces": false,
9      "editor.formatOnSave": true
10     },
11     "javascript.suggest.enabled": true,
12     "javascript.updateImportsOnFileMove.enabled": "never",
13     "javascript.suggest.autoImports": true,
14     "[typescript)": {
15       "editor.formatOnPaste": true,
16       "editor.defaultFormatter": "esbenp.prettier-vscode",
17       "editor.tabSize": 4,
18       "editor.detectIndentation": false,
19       "editor.formatOnType": false,
20       "editor.formatOnSave": true
21     },
22     "[typescriptreact)": {
23       "editor.formatOnPaste": true,
24       "editor.defaultFormatter": "esbenp.prettier-vscode",
25       "editor.tabSize": 4,
26       "editor.detectIndentation": false,
27       "editor.formatOnType": false,
28       "editor.formatOnSave": true
29     },
30     "[javascriptract)": {
31       "editor.defaultFormatter": "esbenp.prettier-vscode",
32       "editor.formatOnPaste": true,
33       "editor.formatOnType": true,
34       "editor.tabSize": 4,
35       "editor.detectIndentation": false,
36       "editor.insertSpaces": false,
37       "editor.formatOnSave": true
38     },
39     "typescript.suggest.enabled": true,
40     "typescript.autoClosingTags": true,
41     "typescript.preferences.quoteStyle": "single",
42     "typescript.updateImportsOnFileMove.enabled": "never",
```

```

43  "typescript.tsserver.log": "verbose",
44  "typescript.suggest.autoImports": true,
45  "eslint.validate": [
46    "javascript",
47    "typescript"
48  ],
49  "[html)": {
50    "editor.defaultFormatter": "vscode.html-language-features",
51    "editor.formatOnPaste": true,
52    "editor.formatOnType": true
53  },
54  "html.autoClosingTags": true,
55  "html.format.indentInnerHtml": true,
56  "[sass)": {
57    "editor.formatOnSave": false,
58    "editor.formatOnPaste": true,
59    "editor.insertSpaces": true,
60    "editor.detectIndentation": true,
61    "editor.autoIndent": "full",
62    "editor.tabSize": 4,
63    "editor.quickSuggestions": {
64      "other": true,
65      "comments": false,
66      "strings": true
67    }
68  },
69  "[json)": {
70    "editor.defaultFormatter": "vscode.json-language-features",
71    "editor.formatOnPaste": true,
72    "editor.formatOnType": true,
73    "editor.tabSize": 4,
74    "editor.detectIndentation": false,
75    "editor.insertSpaces": false
76  },
77  "emmet.syntaxProfiles": {
78    "javascript": "jsx"
79  },
80  "emmet.includeLanguages": {
81    "javascript": "javascriptreact"
82  },
83  "files.associations": {
84    ".stylelintrc": "json",
85    ".prettierrc": "json"
86  },
87  "editor.wordWrapColumn": 80,
88  "editor.codeActionsOnSave": {
89    "source.fixAll.eslint": true,
90    "source.organizeImports": true

```

```

91 },
92 "editor.insertSpaces": false,
93 "editor.autoIndent": "full",
94 "editor.wordWrap": "on",
95 "editor.autoClosingBrackets": "always",
96 "editor.autoClosingQuotes": "always",
97 "editor.tabSize": 4,
98 "editor.tabCompletion": "on",
99 "editor.minimap.enabled": false,
100 "editor.quickSuggestionsDelay": 0,
101 "editor.snippetSuggestions": "top",
102 "editor.formatOnSave": true,
103 "editor.quickSuggestions": {
104     "other": true,
105     "comments": true,
106     "strings": true
107 }
108 }

```

Exerto de Código 46: Configurações utilizadas no **Visual Studio Code**

Nota

Para utilizar as Configurações apresentadas devem ser seguidos os passos abaixo:

1. Aceder às configurações do **Visual Studio Code** no formato **JSON**, para isso utilizar a tecla de atalho apresentada abaixo de acordo com o sistema operativo e pesquisar pela opção “*Preferences: Open Settings (JSON)*”;
 - **No macOS:** CMD + SHIFT + P;
 - **No Windows/Linux:** CTRL + SHIFT + P.
2. Copiar as configurações apresentadas e colar no ficheiro **settings.json** (ficheiro que abriu no passo anterior).
 - **Nota:** caso já possua configurações neste ficheiro, basta remover as chavetas iniciais (**{}**) no código apresentado e colocar as restantes configurações.

Extensões

Como referido anteriormente, o **Visual Studio Code** é rico em extensões, tornando-o bastante versátil e capaz de ser utilizado para qualquer linguagem ou finalidade. Abaixo são apresentadas algumas das extensões usadas no desenvolvimento deste projeto.

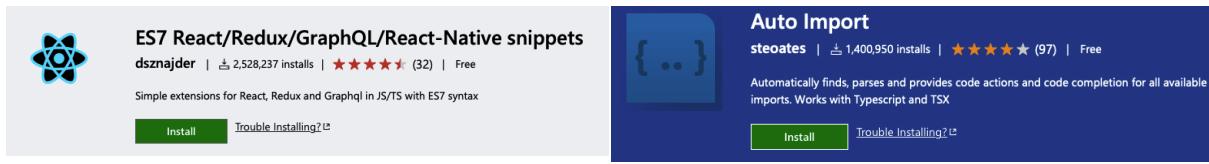


Figura 63: Extensão **ES7 React/Redux/GraphQL/React-Native snippets**
React/Redux/GraphQL/React-Native snippets

[Install](#) [Trouble Installing?](#)

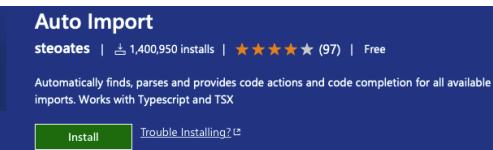
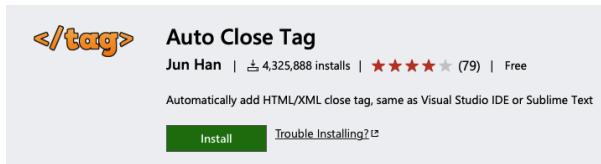


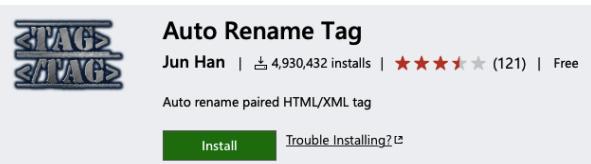
Figura 64: Extensão **Auto Import**

[Link](#)

[Link](#)



[Install](#) [Trouble Installing?](#)



Auto Rename Tag
Jun Han | 4,930,432 installs | ★★★★★★ (121) | Free
Auto rename paired HTML/XML tag

[Install](#) [Trouble Installing?](#)

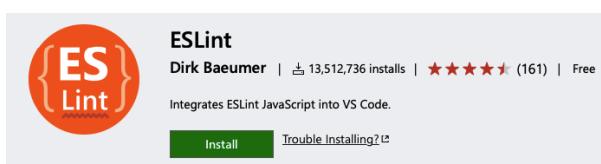
Figura 65: Extensão **Auto Close Tag**

[Link](#)

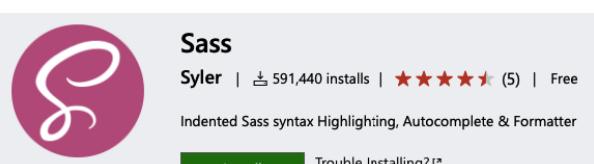
Figura 66: Extensão **Auto Rename Tag**

[Link](#)

[Link](#)



[Install](#) [Trouble Installing?](#)



Sass
Syler | 591,440 installs | ★★★★★★ (5) | Free
Indented Sass syntax Highlighting, Autocomplete & Formatter

[Install](#) [Trouble Installing?](#)

Figura 67: Extensão **ESLint**

[Link](#)

Figura 68: Extensão **Sass**

[Link](#)

[Link](#)

Nota

As extensões apresentadas têm apenas a finalidade oferecer mais funcionalidades ou snippets ao IDE em questão, o **Visual Studio Code**.

Prettier

Configuração

```
1  {
2    "semi": true,
3    "useTabs": true,
4    "tabWidth": 4,
5    "singleQuote": true,
6    "jsxSingleQuote": true,
7    "bracketSpacing": true,
8    "jsxBracketSameLine": false,
9    "trailingComma": "none"
10 }
```

Exerto de Código 47: Configurações utilizadas no **Prettier**

O código apresentado acima é do ficheiro `.prettierrcc`, onde é possível definir várias configurações para o **Prettier**. Além destas configurações é possível ainda definir um ficheiro parecido com o `.gitignore`, no caso `.prettierignore`, onde tal como no `.gitignore` são definidos os ficheiros ou pastas nas quais não será executado o **Prettier**.

```
1 package.json
2 dist/
```

Exerto de Código 48: Exemplo do conteúdo do ficheiro `.prettierignore`

Husky e Git hooks

Juntamente com o **Prettier** pode ser utilizado o **Husky** e o **Lint Staged** para formatar todo o código produzido ao realizar um *commit*.

```
1 {
2   "husky": {
3     "hooks": {
4       "pre-commit": "lint-staged"
5     }
6   },
7   "lint-staged": {
8     "*.{js,jsx,ts,tsx,json,css,scss,md)": [
```

```
9     "prettier --write",
10    "git add ."
11  ]
12 }
13 }
```

Exerto de Código 49: Configurações utilizadas no **Husky**, **Lint Staged** e **Prettier**

Como é possível analisar, ao realizar um *commit* será executado o **Lint Staged**, que por sua vez apenas aplica os comandos aos ficheiros que terminem com as extensões definidas.

Desta forma todo o código é formatado seguindo as regras definidas no ficheiro **.prettier**.