ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

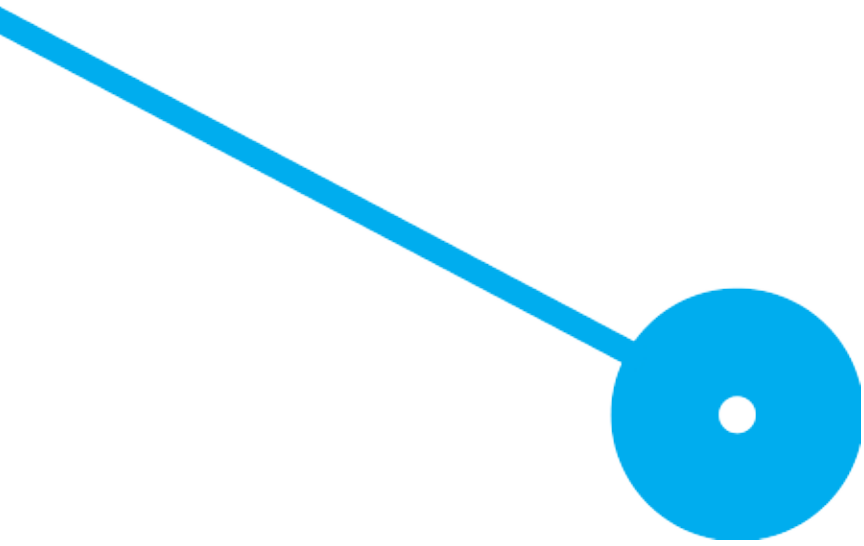
P.PORTO

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

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

Lista de Tabelas

Lista de Excertos de Código

Abreviaturas

LEI Licenciatura em Egenharia 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 Prótotipo 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** disponibilizados 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 TODO

Build TODO

Script TODO

Open Source TODO

Query TODO

Framework TODO

Browser TODO

Workflow TODO

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

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

Agradecimentos

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 de 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* extenos é utilizada a cor azul.

Contudo e, sempre que seja pertinente realçar uma determinada nota, será utilizado o formado 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");
```

Excerto 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 Introdução

1.2 Âmbito/Contextualização

1.3 Objetivos

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:

- **Capítulo 2 —Enquadramento Tecnológico:** neste capítulo é possível encontrar todas as tecnologias utilizadas no decorrer do projeto;
- **Capítulo 3 —Ferramentas & Ambiente de Desenvolvimento:** neste capítulo são abordadas todas as ferramentas utilizadas, bem como a preparação do ambiente de desenvolvimento;
- **Capítulo X —XXX:**

Capítulo 2
Fundamentação Teórica

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, estando em primeiro lugar **Rust**, 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 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

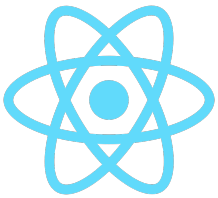


Figura 3: ***React*** *– logo*

Existem quem considere que o **React** é uma *framework* de **JavaScript**, 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:

- Fácil Aprendizagem;
- Rápidez;
- 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 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**, possuíndo 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.

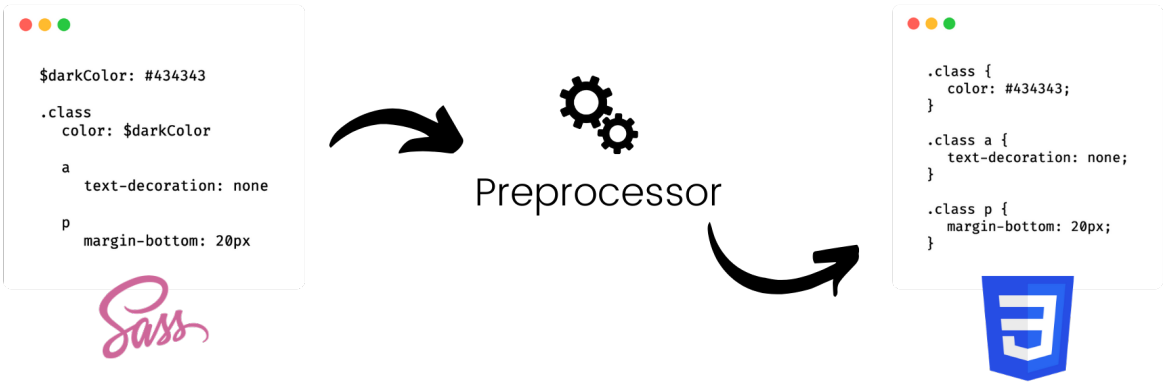


Figura 5: *Ficheiro* `.sass` *compilado para um ficheiro* `.css`

A imagem anterior representa a transformação que é realizada após a compilação de um código **Sass** (`.sass`), onde é possível analisar a declaração de uma variável (`$darkColor`), bem como o seu uso. Além disso, como é possível analisar, o **Sass** permite o uso de *nesting*, ou seja, ... TODO

Os pontos que se seguem demonstram algumas das vantagens em utilizar **Sass**, demonstrando com exemplos práticos.

2.1.3.1 *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)
```

Excerto de Código 3: *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`.

2.1.3.2 Herança

No **Sass** também é possível realizar herança, nesta caso herança de estilos. O excerto de código[1] 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
```

Excerto de Código 4: *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  }
```

Excerto de Código 5: *Código* ***CSS*** *resultante da compilação do excerto de código anterior*

[1] Retirado 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.

```
%toolbelt
  box-sizing: border-box
  border-top: 1px rgba(#000, .12) solid
  padding: 16px 0
  width: 100%

  &:hover
    border: 2px rgba(#000, .5) solid

.action-buttons
  @extend %toolbelt
  color: #4285f4


.reset-buttons
  @extend %toolbelt
  color: #cddc39
```

Excerto de Código 6: *Demonstração de placeholders em* ***Sass***[2]

Sendo que após este código ser compilado, o *placeholder* apresentado não estará no código **CSS** resultante, tal como é possível analisar abaixo.

```
.action-buttons, .reset-buttons {
  box-sizing: border-box;
  border-top: 1px rgba(0, 0, 0, 0.12) solid;
  padding: 16px 0;
  width: 100%;
}

.action-buttons:hover, .reset-buttons:hover {
  border: 2px rgba(0, 0, 0, 0.5) solid;
}

.action-buttons {
  color: #4285f4;
}

.reset-buttons {
  color: #cddc39;
}
```

Excerto de Código 7: *Código* ***CSS*** *resultante da compilação do excerto de código com placeholder*

2.1.3.3 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**.

```
$padding: 10px 20px
$defaultColor: #ca4d24

.alert
  background-color: $defaultColor
  padding: $padding
```

Excerto de Código 8: *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:

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

Excerto de Código 9: *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**.

```
:root {
  --padding: 10px 20px;
  --default-color: #ca4d24
}

.alert {
  padding: var(--padding);
  background-color: var(--default-color);
}
```

Excerto de Código 10: *Declaração e uso de variáveis em* ***CSS***

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 é possível ecnontrar a imagem que representa a arquitetura do **NodeJS** em comparação com a arquitetura tradicional[3].

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 como realizar a instalação do **NodeJS** nos diversos sistemas opertivos.

2.1.5 GraphQL

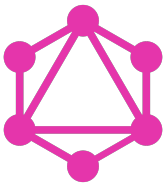


Figura 7: GraphQL — logo

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.

[3] Imagem retirada de [caseByCaseNode]

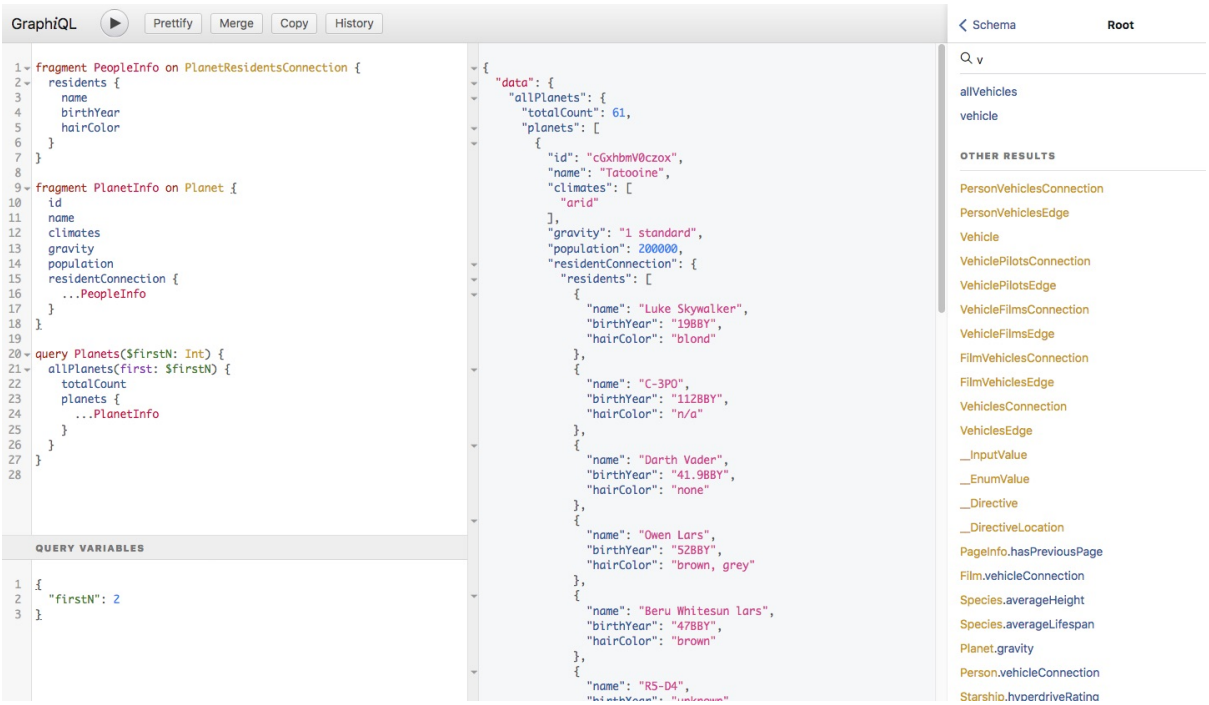


Figura 8: *Demonstração do* ***GraphiQL***

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`).

```
{
  hero {
    name
    # Queries can have comments!
    friends {
      name
    }
  }
}
```

Excerto de Código 11: 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**, contento a propriedade `data`, possuindo todos os resultados obtidos.

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {"name": "Luke Skywalker"},
        {"name": "Han Solo"},
        {"name": "Leia Organa"}
      ]
    }
  }
}
```

Excerto de Código 12: GraphQL *— Exemplo de resposta à query realizada*

Em anexo 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

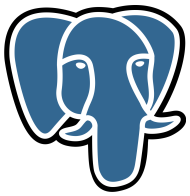


Figura 9: PostgreSQL *— logo*

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

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 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**: [**ultimateVSReact**, **reactToolsVS**, **spVSExtensions**, **vscodeReactSP**]

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 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 ambos[4].

	Sem Cache	Com Cache	Reinstalar
NPM 6.13.4	67 segudos	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*;
- Facilidade de uso | determinados comandos tornam-se mais intuitivos com o **Yarn**;
- Possibilidade de reinstalar packages sem conexão à Internet.

Em anexo 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 conjuto com o **GitLab**, sendo seguido o *workflow* apresentado na imagem que se segue.

[4] Retirado de [**yarnVSNpm**]

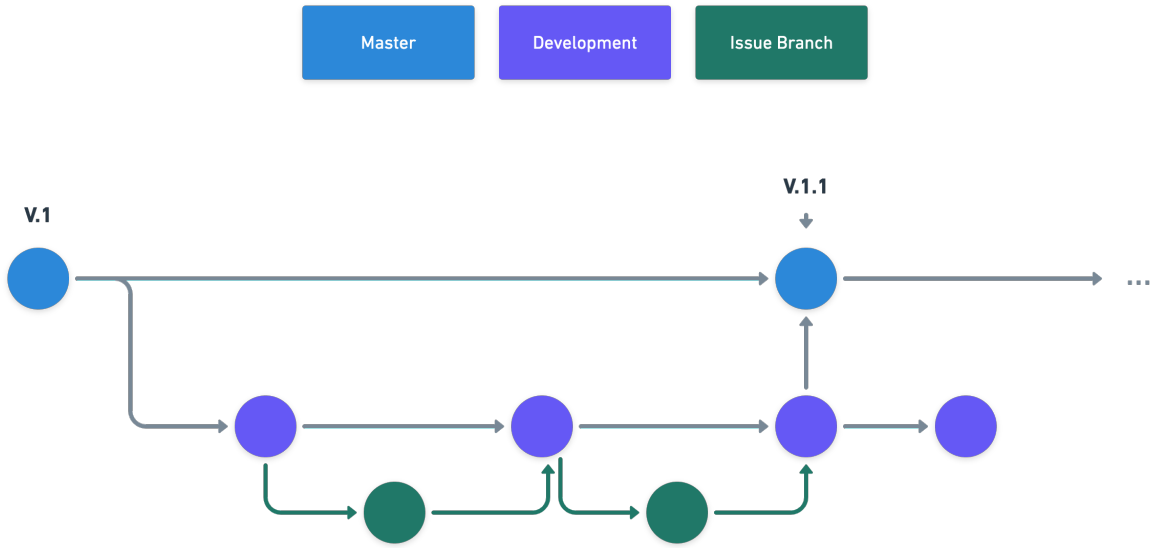


Figura 13: *Workflow seguido no* ***GitLab*** *durante o desenvolvimento*

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 *branchs* 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 estao 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* utilizandos ao longo do projeto encontram-se listdos de seguida.

2.4.1 GitLab

A primeira ferramente utilizada para controlar as tarefas existentes para o projeto foi o **GitLab**, criando para tal *issues*, sendo apresentadas numa *board*. Posteriormente, era através destsa *issues*

também criado um *merge request*, ficando assim associados.

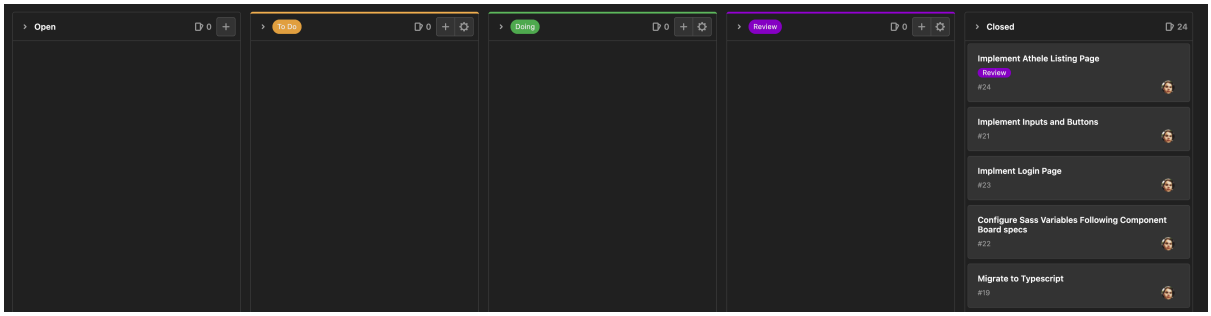


Figura 14: *Board utilizada no* ***GitLab***

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.

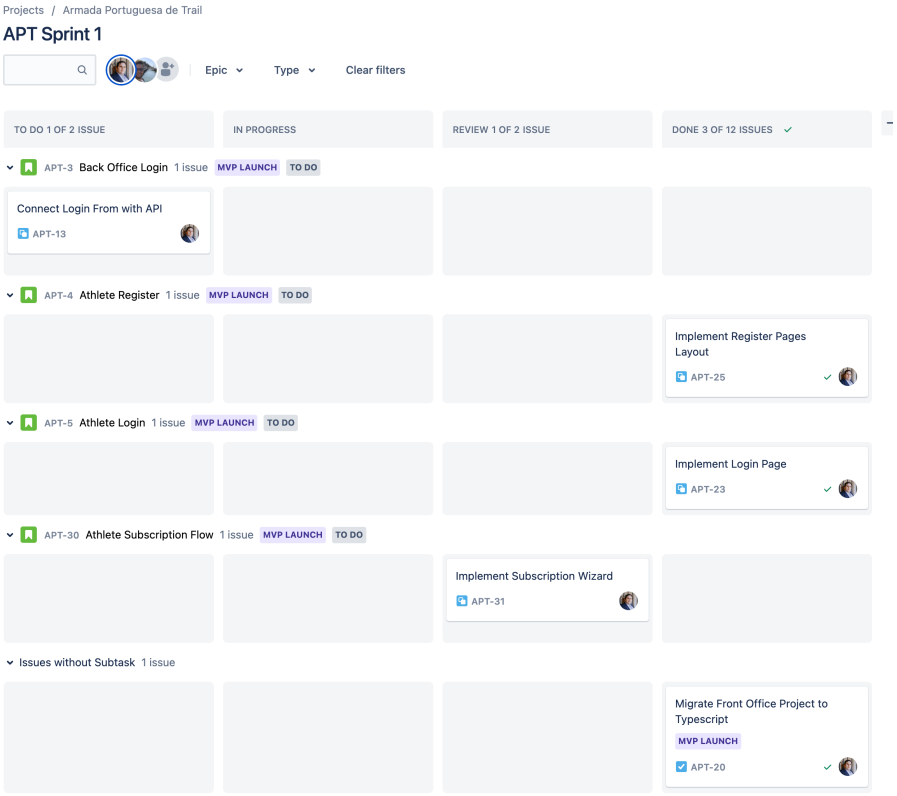


Figura 15: *Board utilizada no* ***Jira***

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*.

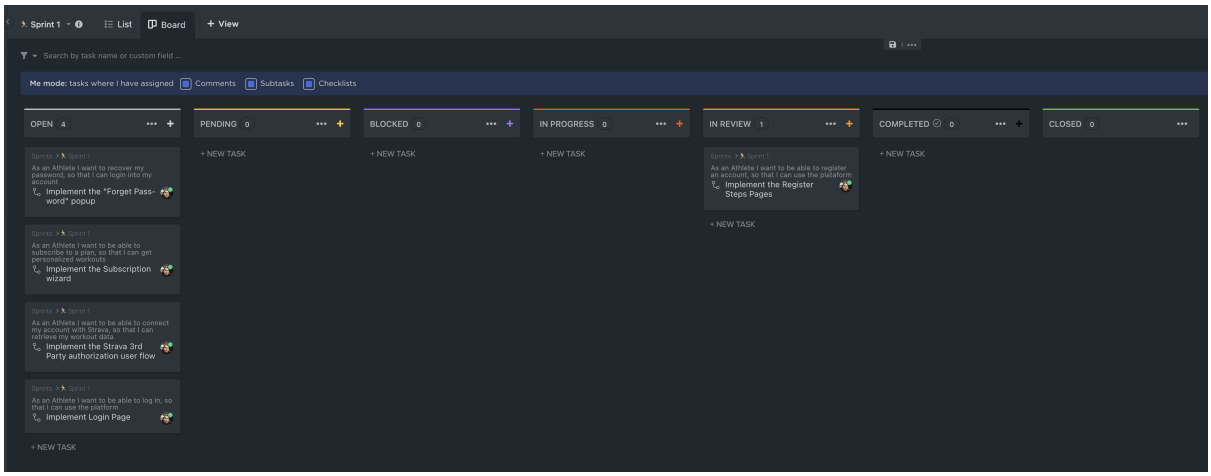


Figura 16: *Vista em board no* ***ClickUp***

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 componentes (*backoffice* e *frontoffice*), analisando de forma resumida o *workflow* executado em cada componente do projeto (*backoffice* e *frontoffice*). As principais depêndencias 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 treinar 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 interpretaçaõ dos treinos, bem como o registo dos resultados obtidos.

A imagem que se segue é um pequeno exemplo do fluxo destas duas componentes. De referir que os atletas irão sempre executar ações na componente de *frontoffice* do projeto e, por sua vez, o administrador e treinadores irão executar na componente de *backoffice*.

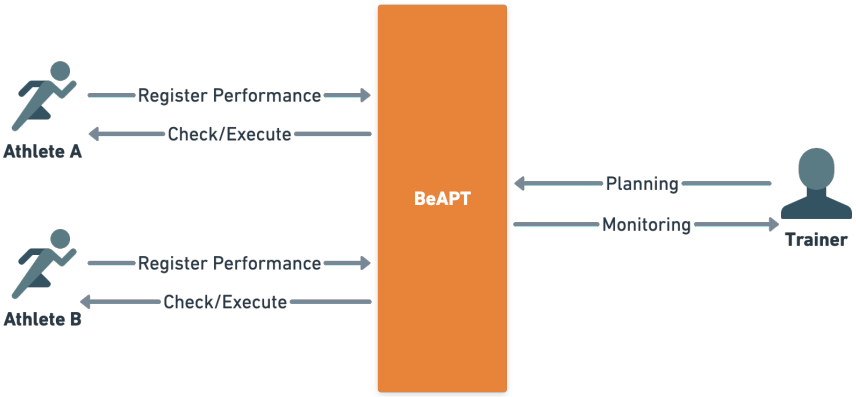


Figura 17: *Exemplo de utilização por parte dos atletas e treinadores*

A lista que se segue apresenta, com mais detalhe, as funcionalidades que estão ao dispor para os atletas na componente 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 componente 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.

3.2 Perspetiva do Utilizador

3.3 Pressupostos de Restrições

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 desta 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 instalado[1] em ambos os projetos, sendo inicialmente ambas as components do projeto[2] migrados de **JavaScript** para **TypeScript**.

[1] **Nota:** em anexo é possível encontrar todos os detalhes sobre a instalação do **TypeScript**
[2] Componente destinada ao atleta (*frontoffice*) e a componente destinada ao *personal trainer* (*backoffice*)

Capítulo 4
Requisitos Específicos

4.1 Requisitos Funcionais

4.2 Requisitos Não Funcionais

Capítulo 5
Design & Implmenetação da Solução

5.1 Arquitetura Conceptual

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

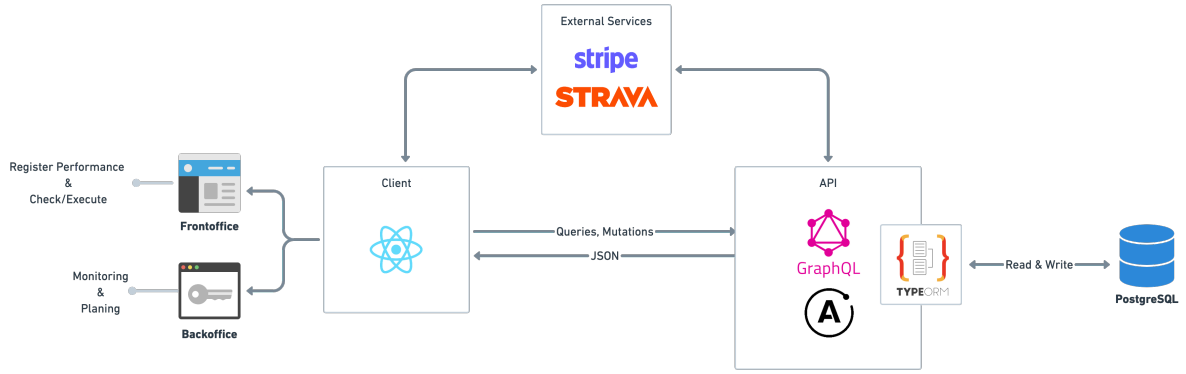


Figura 18: *Arquitetura Conecptual do projeto*

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

- **GraphQL**;
- **Apollo Server**;
- **TypeORM**;
- **KoaJS**[1] (não representado no diagrama);
- **PostgreSQL**;

[1]Refêrencia Bibliográfica sobre **KoaJS**: [**expressVsKoa**, **introkoa**]

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 *personal trainers* 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 compoentens (*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).

5.2 Diagramas de Sequência

5.2.1 Gestão de Sessão

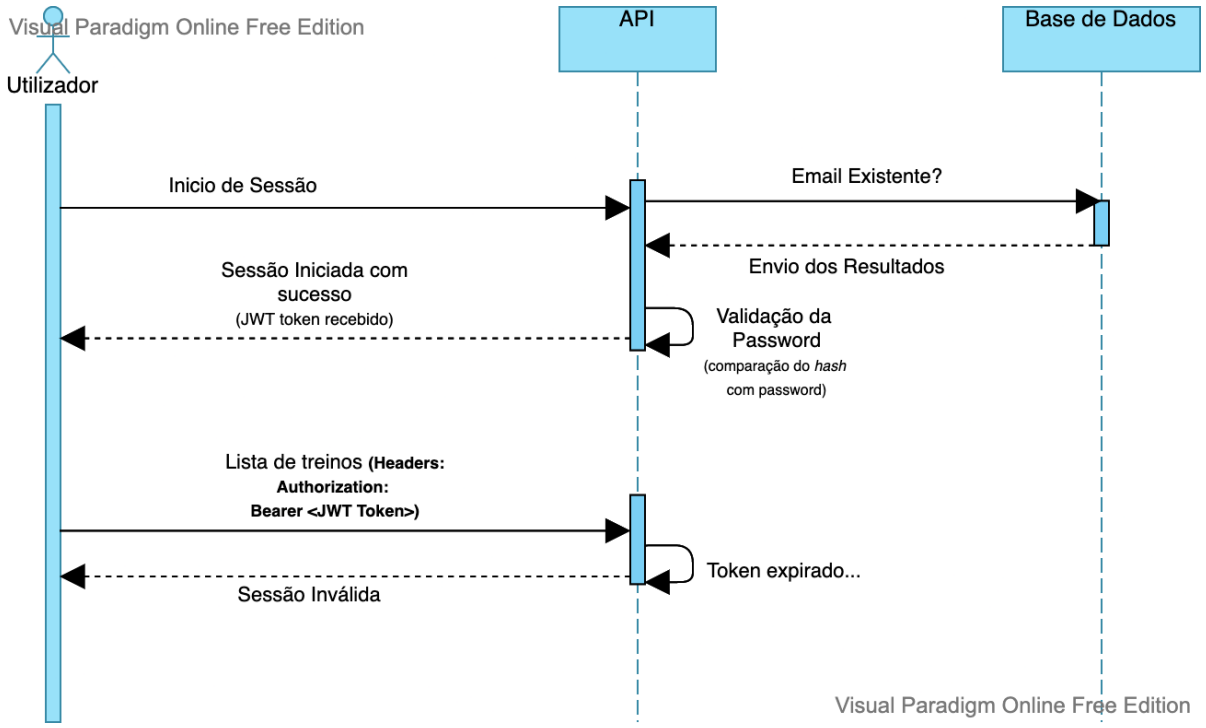


Figura 19: *Diagrama de Sequência de Gestão de Sessão*

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 inicalmente é realizado uma inicio de sessão no *front-end* do projeot, 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.

```
import axios from 'axios';

axios.get('<api-url>/<router>', {
  headers: {
    Authorization: 'Bearer <jwt-token>'
  }
});
```

Excerto de Código 13: *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 erro *HTTP* 401[2] na resposta da **API**.

5.2.2 Inicio de Sessão

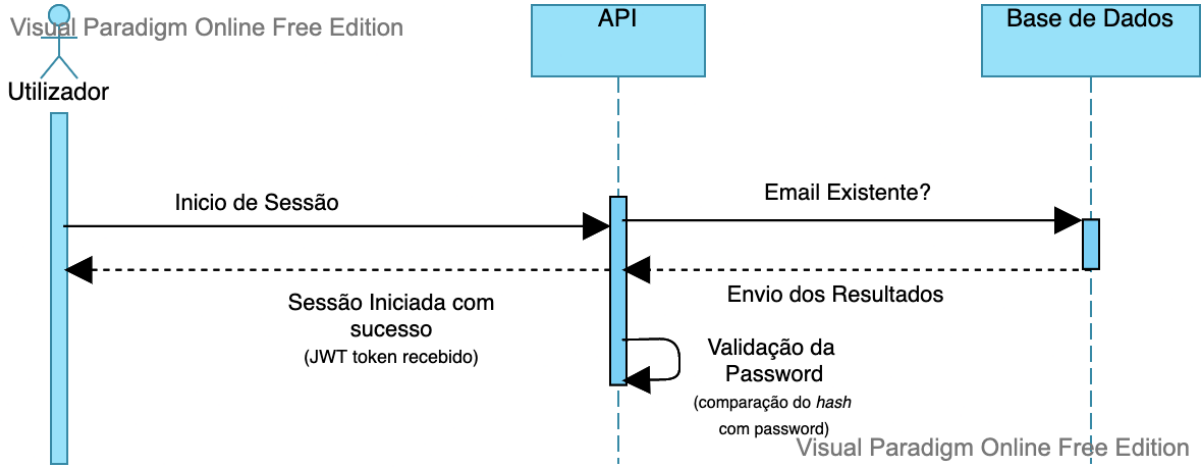


Figura 20: *Diagrama de Sequência para o inicio de sessão*

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

[2] **Erro HTTP 401 —** *Unauthorized* (Mais Informações)

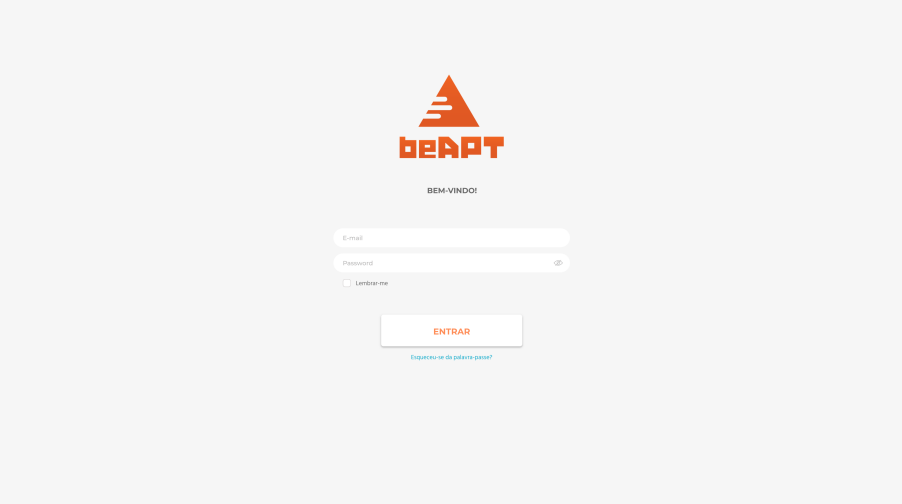


Figura 21: *Mockup do ecrã de inicio de sessão – backoffice*

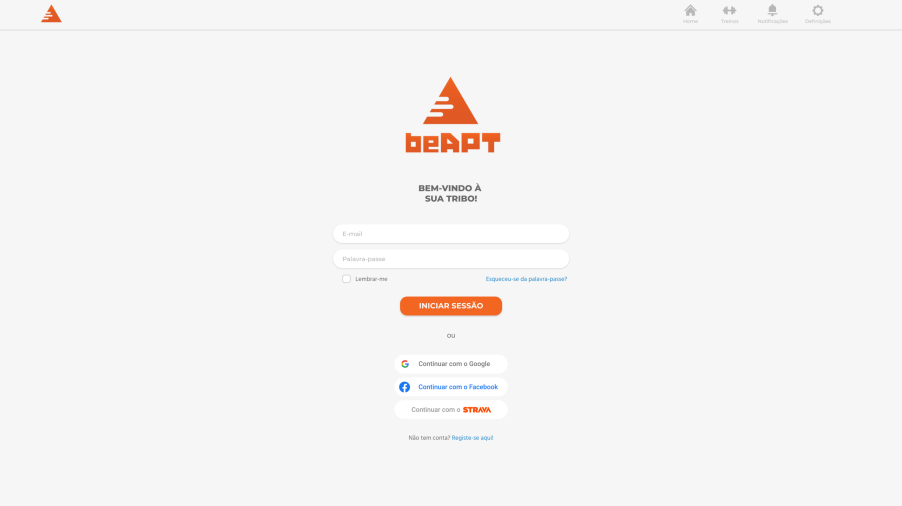


Figura 22: *Mockup do ecrã de inicio de sessão com redes sociais – frontoffice*

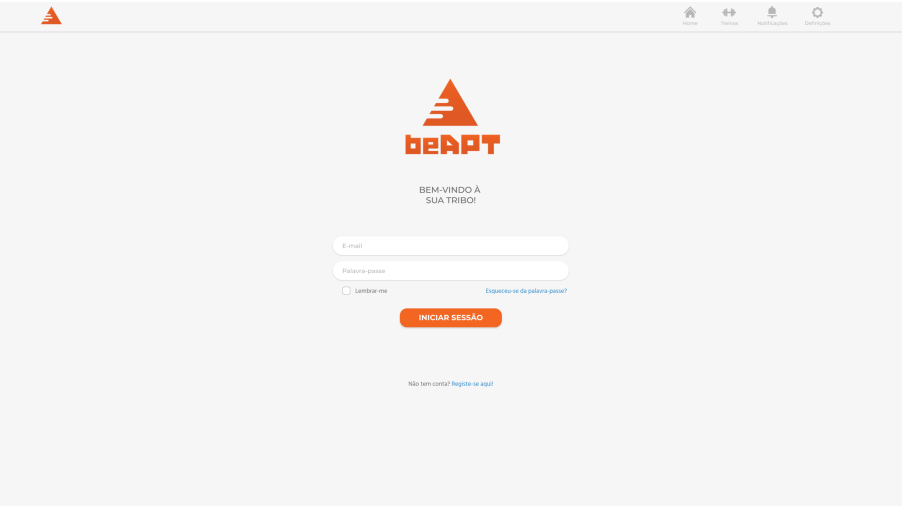


Figura 23: *Mockup do ecrã de inicio de sessão – frontoffice*

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.

5.2.3 Atleta – criação de conta

A criação da conta de um atleta encontra-se presente na componente 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.

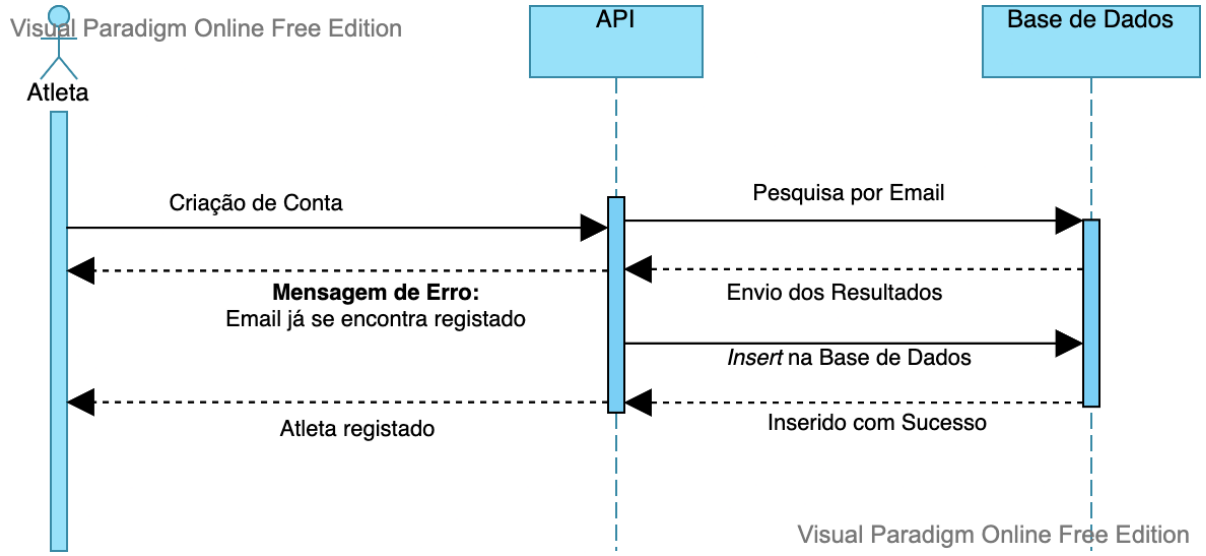


Figura 24: *Diagrama de Sequência para a criação de conta de um atleta*

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.

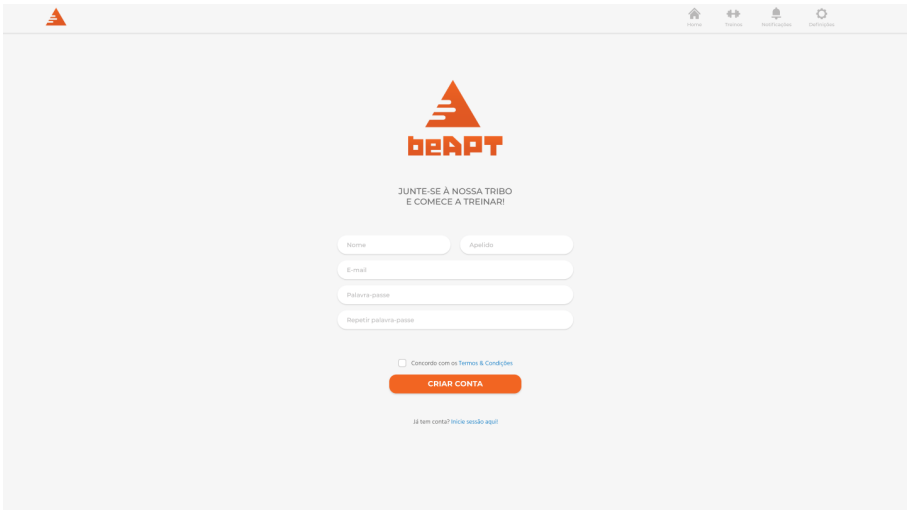


Figura 25: *Mockup do ecrã de criação de conta – frontoffice*

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.

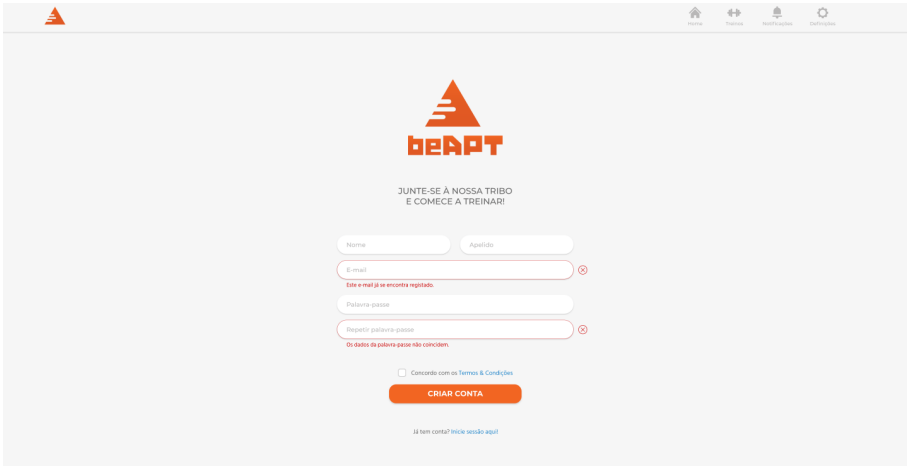


Figura 26: *Mockup do ecrã de criação de conta com dados inválidos – frontoffice*

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.

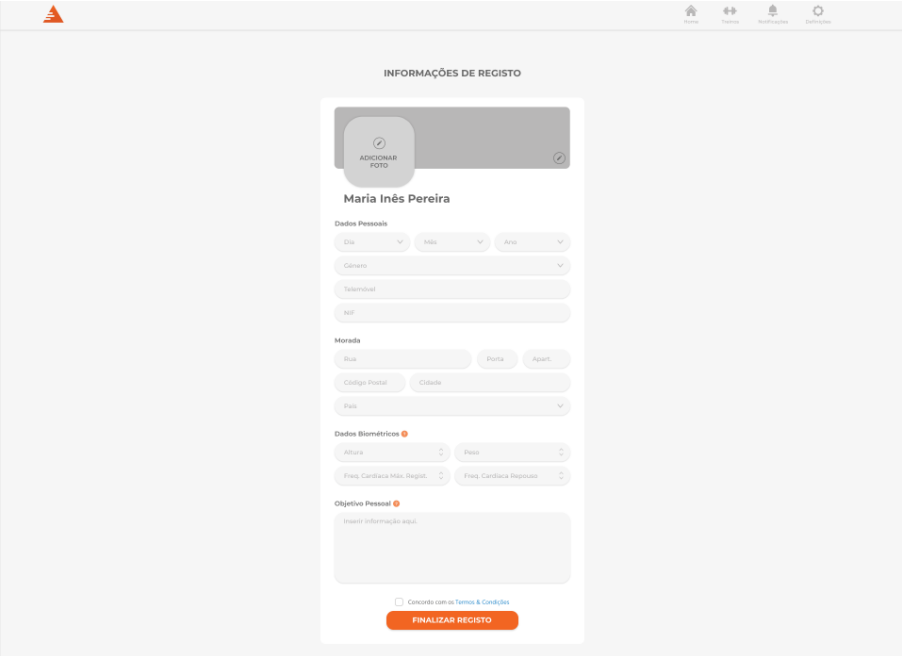


Figura 27: *Mockup de informações de registo – frontoffice*

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

Figura 28: *Inputs de carregamento de imagens*

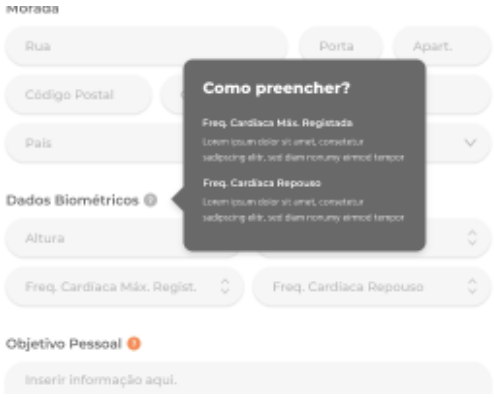


Figura 29: *Componente* ***Tooltip***

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.

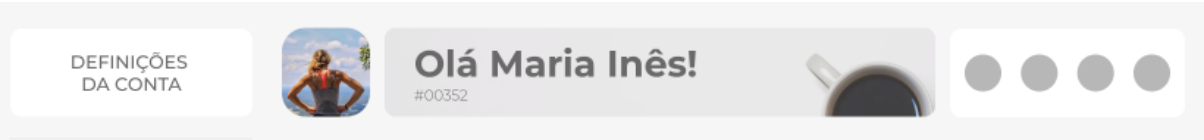


Figura 30: *Imagens carregadas para o perfil do atleta – frontoffice*

5.3 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.

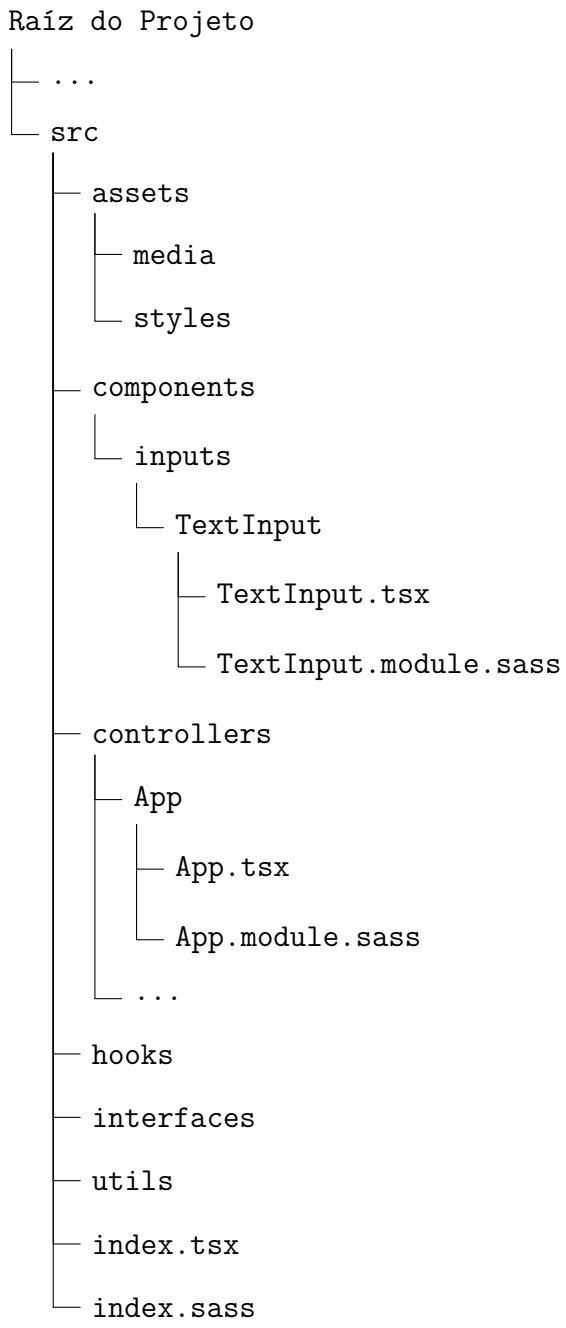


***Figura 31: React** — estrutura de pastas utilizada*

Como é possível analisar a pasta principal do projeto é a para `src/`, sendo esta que contêm 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` reponsável 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 6
Gestão e Acompanhamento do Projeto

Capítulo 7
Resultados

7.1 Cenário de Validação

7.2 Análise de Resultados

Capítulo 8
Conclusões

8.1 Formações Adicionais

De forma a conseguir adquirir novos conhecimentos e aprofundar conhecimentos já existentes, foram realizadas algumas formações adicionais durante os tempos livres (como fins de semana, por exemplo). A lista que se segue apresenta alguns dos projetos realizados durante estas formações, bem como as respetivas ligações para os mesmos.

- **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 os principais conhecimentos aplicados foram relacionados com a biblioteca **React**, mais precisamente na utilização de contextos, o uso de *Styled Components* para realizar todo o *layout* da aplicação (em vez de **CSS** ou **Sass**) e o uso de dois temas, no caso *dark* e *light mode*.

8.2 Trabalhos Futuros

- Testes com JEST – para *unit test*
- Testes com Cypress – para *end-to-end*
- Biblioteca de componentes

Referências Bibliográficas

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 dependências[1], 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**.

[1] Recorrendo a `yarn install` ou `npm install`.

Configuração

O **TypeScript** permite realizar determinadas configurações no projeto, recorrendo para tal ao ficheiro `tsconfig.json`[2]. 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.

```
{
  "compilerOptions": {
    "target": "es5",
    "lib": [
      "dom",
      "dom.iterable",
      "esnext"
    ],
    "allowJs": true,
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "noFallthroughCasesInSwitch": true,
    "module": "esnext",
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react",
    "experimentalDecorators": true,
    "baseUrl": "src",
    "rootDir": "src",
    "paths": {
      "@components/*": [
        "src/components/*"
      ]
    }
  },
  "include": [
```

[2] Documentação Oficial

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

Excerto de Código 14: 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 encontra-se na raíz 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 issó é necessário adicionar o seguinte código no ficheiro `package.json`:

```
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
```

Excerto de Código 15: *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** 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**[3] 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).

[3] **Documentação:** "Create a new React App"

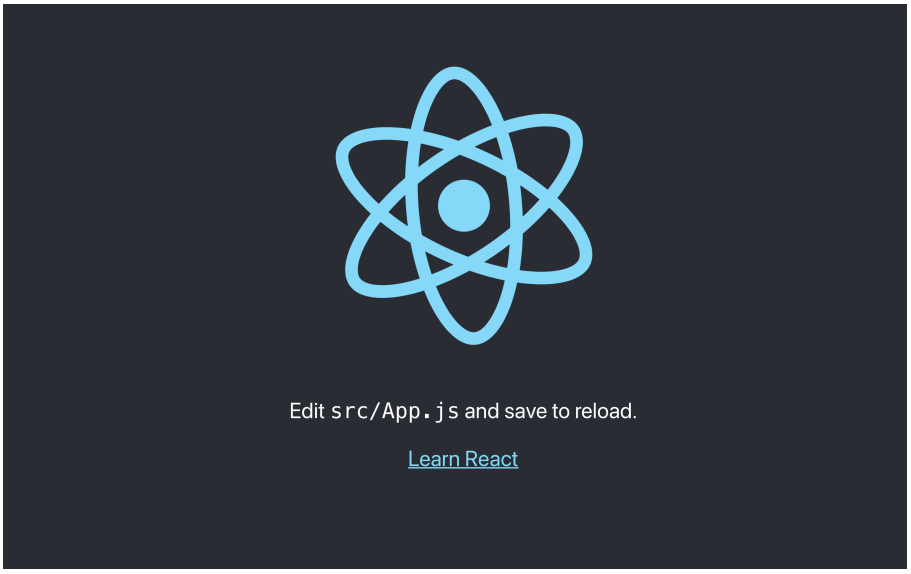


Figura 32: *Página inicial do* **React** *após a execução do projeto*

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*[4].

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.

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.

[4] No caso de possuir o **Yarn** instalado.

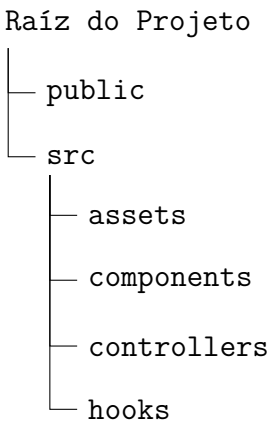


Figura 33: React — possível estrutura de pastas

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:

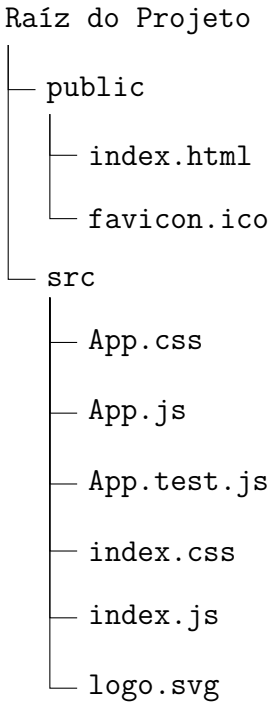


Figura 34: React — estrutura de pastas e ficheiros gerados pelo `create-react-app`

Nota

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

Ficheiros Iniciais

```html
<!DOCTYPE html>
<html lang="en">

  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta name="description" content="Web site created using create-react-app" />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>React App</title>
  </head>

  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>

</html>
```

Excerto de Código 16: *Ficheiro* ***`index.html`*** *de um projeto* ***React***

```jsx
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import './index.css';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

Excerto de Código 17: *Ficheiro* ***`index.jsx`*** *de um projeto* ***React***

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

```
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

Excerto de Código 18: *Ficheiro* ***`index.css`*** *de um projeto* ***React***

```
import React from 'react';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <p>
          Edit <code>src/App.jsx</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

Excerto de Código 19: *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:

Figura 35: *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,
```

```
  disabled,
  noValidate,
  onChange,
  type,
  name,
  placeholder,
  multiline,
  modifier,
  rounded,
  rows,
  ...attributes
}: ITextInput) => {
  // ...
}
```

Excerto de Código 20: *Propriedades recebidas no componente* ***TextInput***

Assim sendo e, apesar de algumas das propriedades não serem obrigatórias, as que tem mais impacto no aspeto 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 excerto de código que segue apresenta a aplicabilidade desta propriedade, bem como o estilo adicionado.

```
// ...
<Tag
  className={`${styles[`${modifier}-style`] || ''} ${
    rounded && styles['rounded']
  }`}
  {...attributes}
  type={type}
  name={name}
  disabled={disabled}
  placeholder=' '
  rows={rows ? Number(rows) : 0}
  onChange={onChange}
>
// ...
</Tag>

// ...
```

Excerto de Código 21: *Propriedade* ***`rounded`*** *aplicada ao input*

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

Excerto de Código 22: *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:

```tsx
// ...

const Tag: ElementType = multiline ? 'textarea' : 'input';

// ...
```

Excerto de Código 23: *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*.

```sass
&.light-style
  background-color: $whiteColor // #FFF

  &:hover
    border: 1px solid $secondaryColor // #b8b7b7

  &:focus
    outline: none
    border: 1px solid $secondaryColor  // #b8b7b7
```

Excerto de Código 24: *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, proprieades que permitem tornar este componente mais versátil.

```tsx
const Button: React.FC<Props> = ({
  icon,
  className,
  rounded,
  to,
  type,
  modifier,
  children,
  onClick,
  ...attributes
}) => {
  // ..
}
```

Excerto de Código 25: *Propriedades recebidas no componente* ***Button***

Neste ponto serão apenas apresentadas as proprieades `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:

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

Excerto de Código 26: *Uso de* ***`Link`*** *ou* ***`button`*** *no componente* ***Button***

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

```tsx
<Tag
  {...tagAttributes}
  {...attributes}
  onClick={onClick}
  className={`
    ${styles['root'] || ''}
    ${styles[`${modifier}-modifier`] || ''}
    ${(rounded && styles['rounded']) || ''}
    ${className || ''}`}
></Tag>
```

Excerto de Código 27: *Aplicação de classes adicionais para as proprieades* ***`rounded`*** *e modifier*

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

```
.alt-primary
  background-color: $whiteColor
  color: $buttonHighlightColor

  &:disabled
    color: $whiteColor
    background-color: $secondaryColor
    box-shadow: 0px 1px 2px #00000029

  &:enabled
    &:hover
      box-shadow: 0px 5px 6px #00000029

    &:focus
      background-color: $whiteColor
      color: $buttonHighlightColor
      box-shadow: 0px 0px 7px #13ACCC

    &:active
      background-color: $whiteColor
      color: $buttonHighlightColor
      border: 1px solid #D3D3D3
      box-shadow: none


.info-modifier
  background: $appBackgroundColor
  color: $primaryColor

  &:disabled
    background-color: $secondaryColor
    color: $whiteColor

  &:enabled
    &:hover
      background-color: $appBackgroundColor
      color: #707070
      box-shadow: 0px 5px 6px #00000029

    &:focus
      background-color: $appBackgroundColor
      color: $primaryColor
      box-shadow: 0px 0px 7px #13ACCC

    &: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
```

Excerto de Código 28: *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 possiblidades de apresentação, tal como é possível analisar nas figuras que se seguem.

Figura 36: *Componente* ***Avatar*** *na página de perfil do atleta*

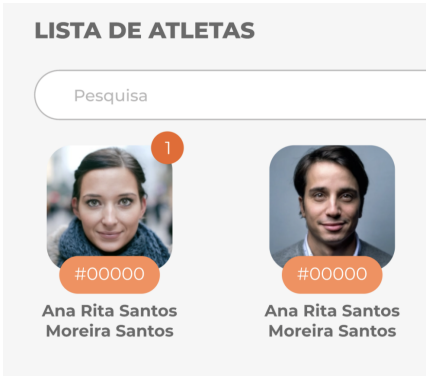


Figura 37: *Componente* ***Avatar*** *na vista em mosaico na lista de atletas*

Figura 38: *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.

```
const Avatar = ({
  className,
  avatar,
  notifications,
  label,
  ...attributes
}: IAvatarProps) => {
  return (
    <div
      className={`${styles["root"] || ""} ${className || ""}`}
      {...attributes}
    >
      <div
        className={styles["avatar"]}
        style={{
          backgroundImage: avatar ? `url(${avatar})` : undefined
        }}
      >
        {label && <span className={styles["label"]}>{label}</span>}

        {notifications && (
          <span className={styles["badge"]}>{notifications}</span>
        )}
      </div>
    </div>
  );
};

export default Avatar;
```

Excerto de Código 29: *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

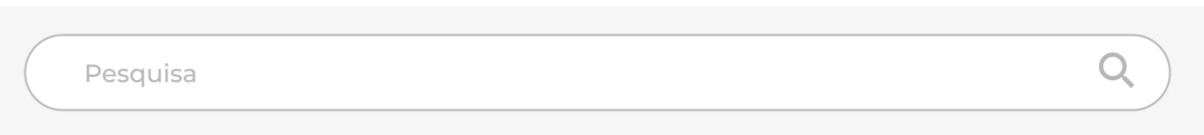


Figura 39: *Aparência do componente* ***SearchInput***

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 icone 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.

```
const SearchInput = ({
  className,
  onChange,
  ...attributes
}: ISearchInputProps) => {
  return (
    <label className={`${styles["root"] || ""} ${className || ""}`}>
      <input type="text" {...attributes} onChange={onChange} />

      <Icon name="search" width="20px" height="20px" />
    </label>
  );
};

export default SearchInput;
```

Excerto de Código 30: *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.

```
import SearchInput from 'components/inputs/SearchInput/SearchInput';
import Fuse from 'fuse.js';

// ...

const [data, setData] = useState(athletes);

// Search
const options = {
  keys: ['name'],
};

const fuse = new Fuse(athletes, options);

const oneSearchChange = (evt: ChangeEvent<HTMLInputElement>) => {
  if (evt.target.value !== '') {
    setData(fuse.search(evt.target.value).map((fuseItem) => fuseItem.item));
```

```
  } else {
    setData(athletesData);
  }
};

return (
  // ...
  <SearchInput placeholder='Pesquisa' onChange={oneSearchChange} />
);
```

Excerto de Código 31: *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**.

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 32: *Importação do* ***GraphQL*** *em* ***JavaScript***

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

Excerto de Código 33: *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 tópcios 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**.

```
import { ApolloClient, InMemoryCache } from '@apollo/client';

const client = new ApolloClient({
  uri: '<api-url>',
  cache: new InMemoryCache(),
});
```

Excerto de Código 34: *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 variavél criada (`const client`) e o método `query`. Novamente este exemplo pode ser encontrado na documentação oficial do **Apollo Client**.

```
import { gql } from '@apollo/client';

// const client = ...

client
  .query({
    query: gql`
      query GetRates {
        rates(currency: "USD") {
          currency
        }
      }
    `,
  })
  .then((result) => console.log(result));
```

Excerto de Código 35: *Execução de uma query recorrendo ao* ***Apollo Client*** *no* ***React***

Comparação entre GraphQL e REST

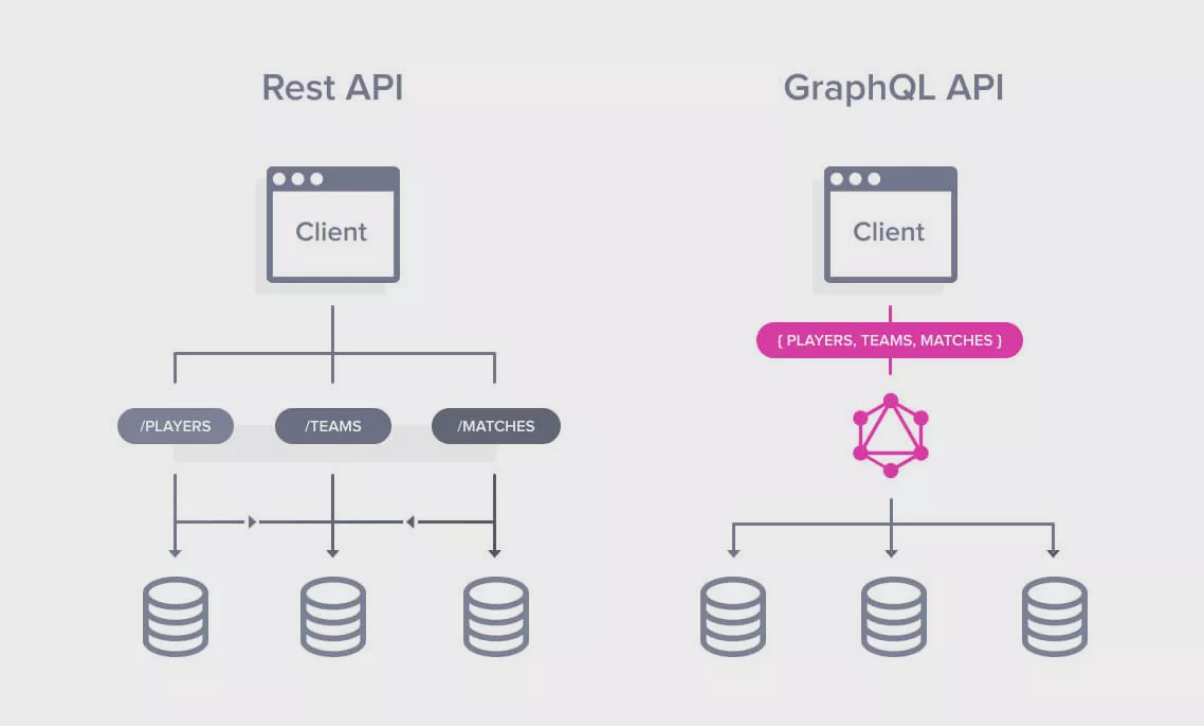


*Figura 40: Comparação entre **GraphQL** e **REST***

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 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**[5] 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.l
  ```
- `sudo apt update && sudo apt install yarn`.

[5] Os comandos apresentados são para distribuições com base **Debian**
  ```

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 apresentas de seguida são apenas algumas das soluções existentes.

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

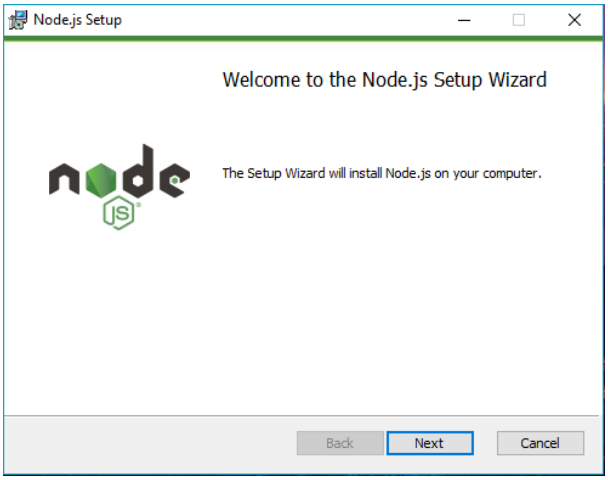
De referir que o**NVM**[6] é uma ferramenta bastante útil para os utilizadores que necessitam de mais do que uma versão do **NodeJS**, conseguindo posteriormente realizar a atribuição de uma versão específica a cada projeto, recorrendo para tal ao comando `nvm use <version>`

Linux

Para realizar a instalação do **NodeJS** em Linux é[7] 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**.



[6] Repositório oficial do NVM
[7] **Nota:** os comandos apresentados são para distribuições baseadas em Debian

*Figura 41: Ecrã inicial da instalação do **NodeJS** no Windows*

Arquitetura

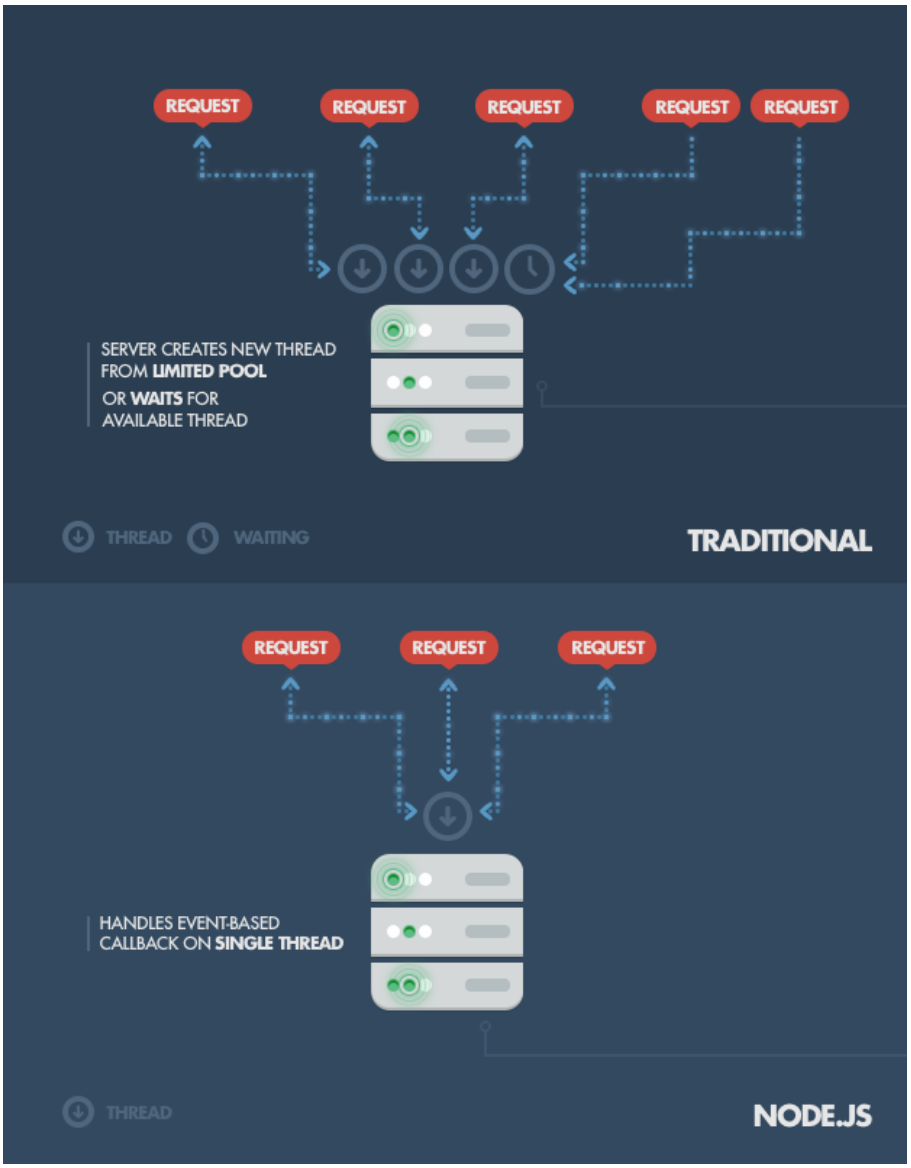


*Figura 42: Arquitetura do **NodeJS** em comparação com a arquitetura tradicional*

PostgreSQL

Instalação

Container Docker

Uma das formas de instalar o **PostgreSQL** é através de um *container* **Docker**[8], 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`.

[8] **Referências recomendadas:** [**postgresDocker**, **postgresContainer**]

Linux

Para realizar a instalação do **PostgreSQL** no **Linux**[9] é necessário executar os seguintes comandos[10]:

- `sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)`
- `wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -`
- `sudo apt-get update`
- `sudo apt-get -y install postgresql`

[9] Os comandos apresentados são para distribuições com base **Debian**
[10] Comandos retirados do site oficial

Visual Studio Code

Configurações

```
{
  "[javascript]": {
    "editor.defaultFormatter": "esbenp.prettier-vscode",
    "editor.formatOnPaste": true,
    "editor.formatOnType": true,
    "editor.tabSize": 4,
    "editor.detectIndentation": false,
    "editor.insertSpaces": false,
    "editor.formatOnSave": true
  },
  "javascript.suggest.enabled": true,
  "javascript.updateImportsOnFileMove.enabled": "never",
  "javascript.suggest.autoImports": true,
  "[typescript]": {
    "editor.formatOnPaste": true,
    "editor.defaultFormatter": "esbenp.prettier-vscode",
    "editor.tabSize": 4,
    "editor.detectIndentation": false,
    "editor.formatOnType": false,
    "editor.formatOnSave": true
  },
  "[typescriptreact]": {
    "editor.formatOnPaste": true,
    "editor.defaultFormatter": "esbenp.prettier-vscode",
    "editor.tabSize": 4,
    "editor.detectIndentation": false,
    "editor.formatOnType": false,
    "editor.formatOnSave": true
  },
  "[javascriptreact]": {
    "editor.defaultFormatter": "esbenp.prettier-vscode",
    "editor.formatOnPaste": true,
    "editor.formatOnType": true,
    "editor.tabSize": 4,
    "editor.detectIndentation": false,
    "editor.insertSpaces": false,
    "editor.formatOnSave": true
  },
  "typescript.suggest.enabled": true,
  "typescript.autoClosingTags": true,
  "typescript.preferences.quoteStyle": "single",
  "typescript.updateImportsOnFileMove.enabled": "never",
```

```
  "typescript.tsserver.log": "verbose",
  "typescript.suggest.autoImports": true,
  "eslint.validate": [
    "javascript",
    "typescript"
  ],
  "[html]": {
    "editor.defaultFormatter": "vscode.html-language-features",
    "editor.formatOnPaste": true,
    "editor.formatOnType": true
  },
  "html.autoClosingTags": true,
  "html.format.indentInnerHtml": true,
  "[sass]": {
    "editor.formatOnSave": false,
    "editor.formatOnPaste": true,
    "editor.insertSpaces": true,
    "editor.detectIndentation": true,
    "editor.autoIndent": "full",
    "editor.tabSize": 4,
    "editor.quickSuggestions": {
      "other": true,
      "comments": false,
      "strings": true
    }
  },
  "[json]": {
    "editor.defaultFormatter": "vscode.json-language-features",
    "editor.formatOnPaste": true,
    "editor.formatOnType": true,
    "editor.tabSize": 4,
    "editor.detectIndentation": false,
    "editor.insertSpaces": false
  },
  "emmet.syntaxProfiles": {
    "javascript": "jsx"
  },
  "emmet.includeLanguages": {
    "javascript": "javascriptreact"
  },
  "files.associations": {
    ".stylelintrc": "json",
    ".prettierrc": "json"
  },
  "editor.wordWrapColumn": 80,
  "editor.codeActionsOnSave": {
    "source.fixAll.eslint": true,
    "source.organizeImports": true
```

```
  },
  "editor.insertSpaces": false,
  "editor.autoIndent": "full",
  "editor.wordWrap": "on",
  "editor.autoClosingBrackets": "always",
  "editor.autoClosingQuotes": "always",
  "editor.tabSize": 4,
  "editor.tabCompletion": "on",
  "editor.minimap.enabled": false,
  "editor.quickSuggestionsDelay": 0,
  "editor.snippetSuggestions": "top",
  "editor.formatOnSave": true,
  "editor.quickSuggestions": {
    "other": true,
    "comments": true,
    "strings": true
  }
}
```

Excerto de Código 36: *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.

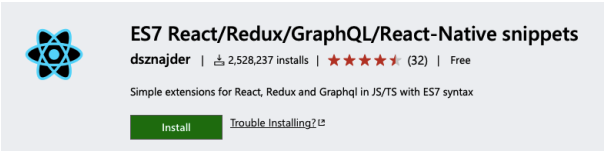


*Figura 43: Extensão **ES7 React/Redux/GraphQL/React-Native snippets***

Link

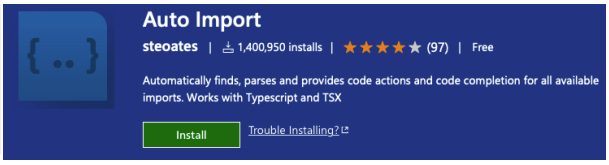


*Figura 44: Extensão **Auto Import***

Link

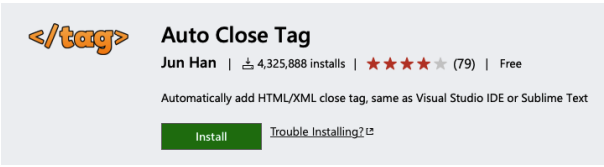


*Figura 45: Extensão **Auto Close Tag***

Link

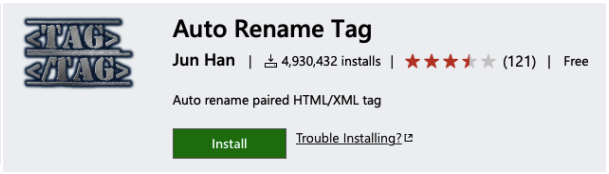


*Figura 46: Extensão **Auto Rename Tag***

Link

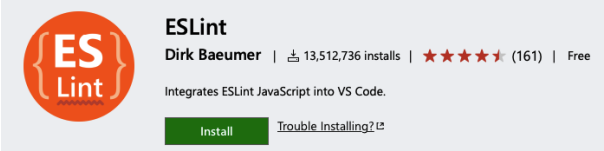


*Figura 47: Extensão **ESLint***

Link

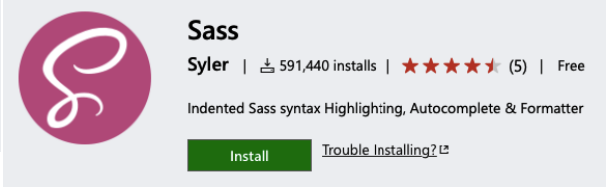


*Figura 48: Extensão **Sass***

Link

Nota

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

Prettier

Configuração

```
{
  "semi": true,
  "useTabs": true,
  "tabWidth": 4,
  "singleQuote": true,
  "jsxSingleQuote": true,
  "bracketSpacing": true,
  "jsxBracketSameLine": false,
  "trailingComma": "none"
}
```

Excerto de Código 37: *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**.

```
package.json
dist/
```

Excerto de Código 38: *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*.

```
{
  "husky": {
    "hooks": {
      "pre-commit": "lint-staged"
    }
  },
  "lint-staged": {
    "*.{js,jsx,ts,tsx,json,css,scss,md}": [
```

```
      "prettier --write",
      "git add ."
    ]
  }
}
```

Excerto de Código 39: *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`.