ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

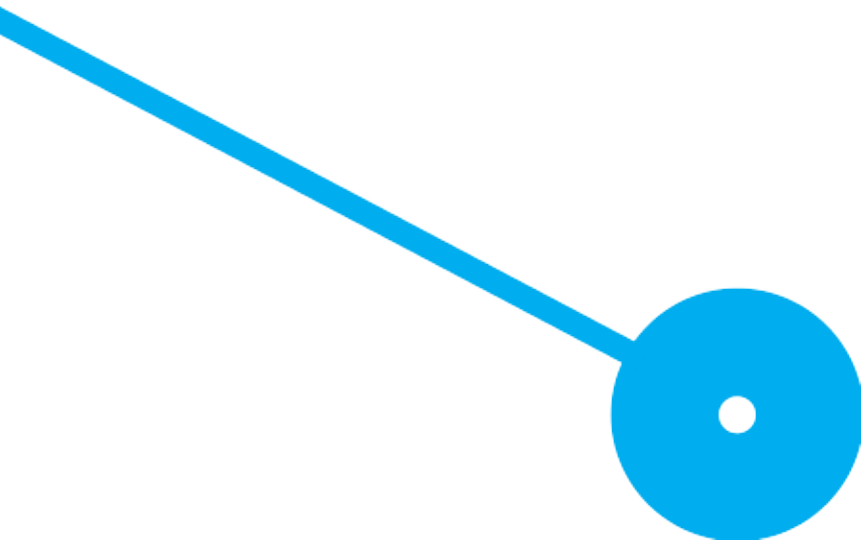
P.PORTO

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*

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*

Web *World Wide Web*

RGPD Regulamento Geral sobre a Proteção de Dados

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 TODO

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*

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.

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..

Já em casos de textos sublinhados, por norma, referem-se a ligações no documento, por exemplo a ligação para uma determinada definição no glossário.

Além disso, sempre que seja pretendido realçar uma nota será utilizado o exemplo abaixo.

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 Objetivos

1.3 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

Neste capítulo

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

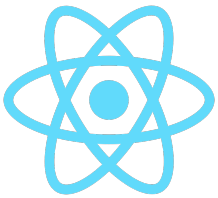


Figura 3: ***React*** *– logo*

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.

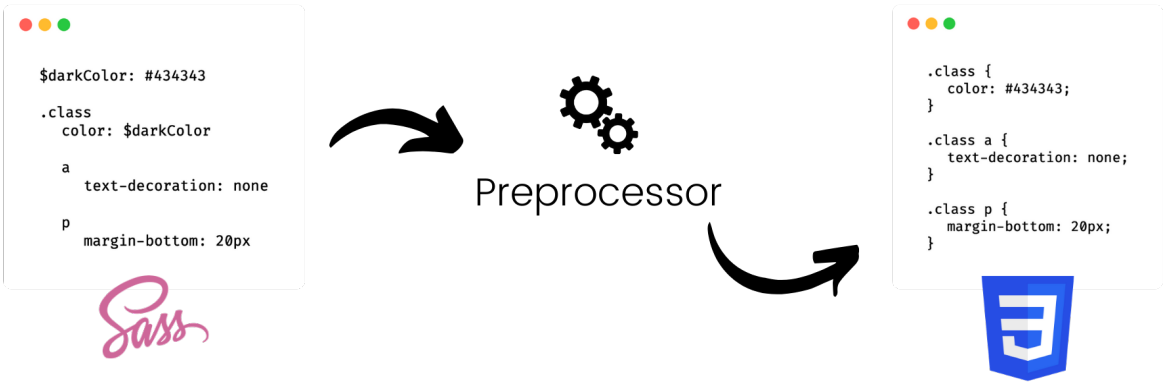


Figura 5: *Ficheiro* `.sass` *compilado para um ficheiro* `.css`

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.

```sass
%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.

```css
.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**.

```sass
$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:

```css
.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**.

```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**.

A imagem que se segue representa a arquitetura do **NodeJS** em comparação com a arquitetura tradicional[3].

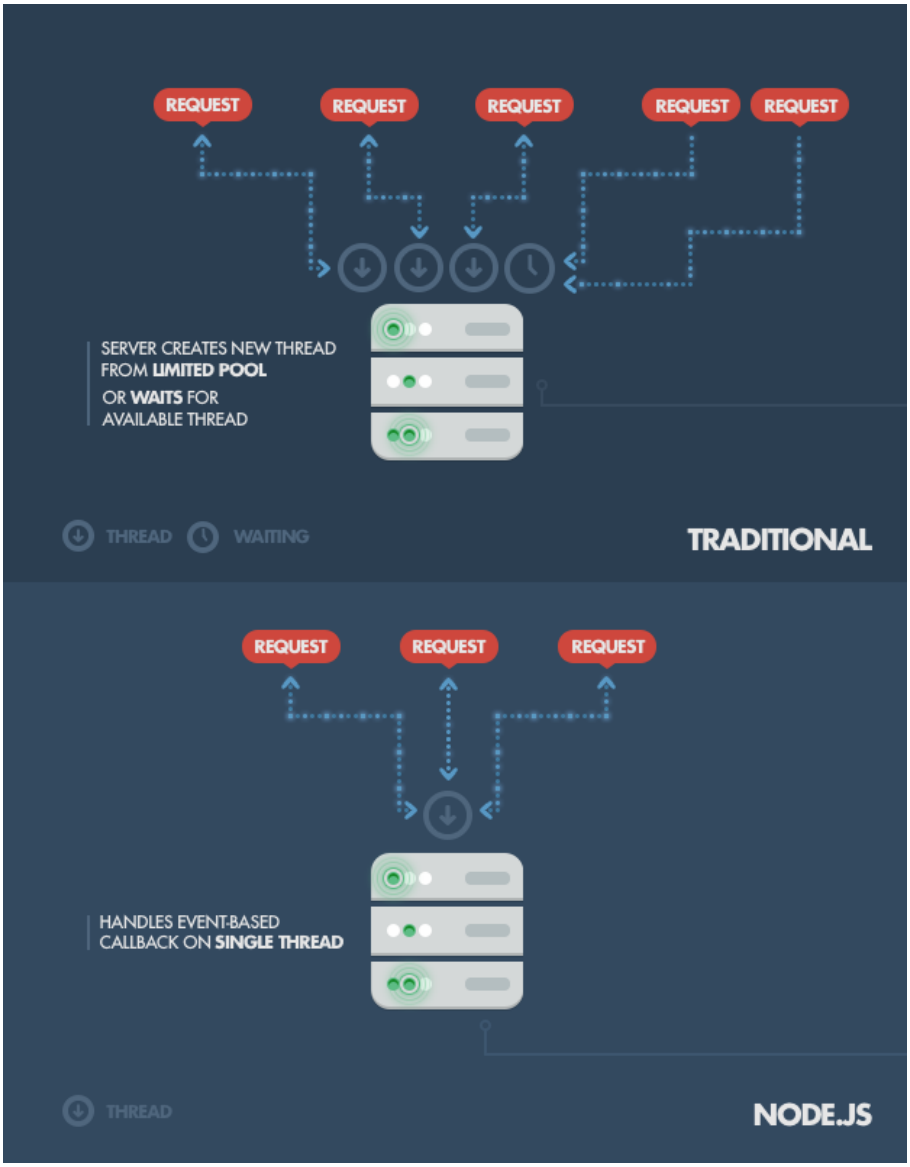


*Figura 7: Arquitetura do **NodeJS** em comparação com a arquitetura tradicional*

[3] Imagem retirada de [caseByCaseNode]

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.

2.1.5 GraphQL

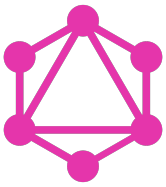


Figura 8: ***GraphQL*** *– logo*

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.

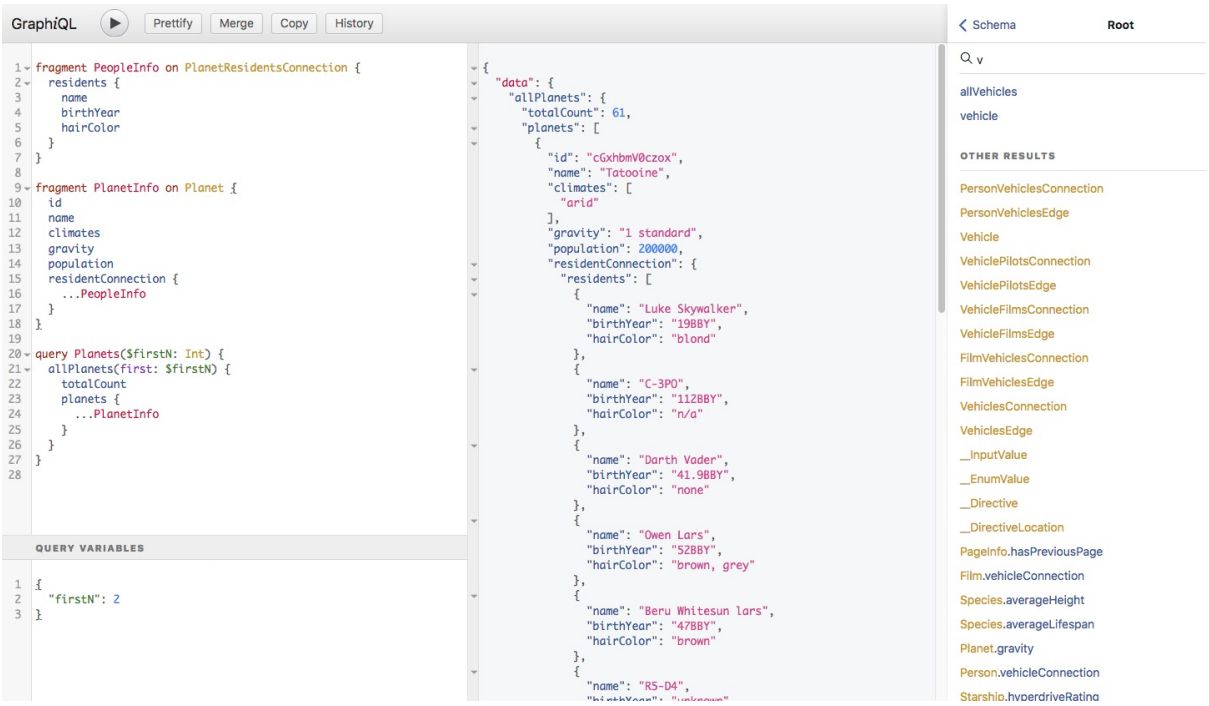


Figura 9: *Demonstração do* ***GraphiQL***

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**.

2.1.6 PostgreSQL

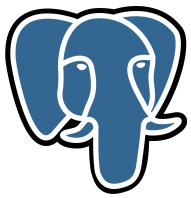


Figura 10: PostgreSQL — logo

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 11: 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 12: WebStorm – logo

O **WebStorm** é outro IDE bastante conhecido e "poderoso", não sendo v 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 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**.

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

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[5], 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**.

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

Configuração

O **TypeScript** permite realizar determinadas configurações no projeto, recorrendo para tal ao ficheiro `tsconfig.json`[6]. 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": [
```

[6] Documentação Oficial

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

Excerto de Código 13: 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 14: *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**[7] 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).

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

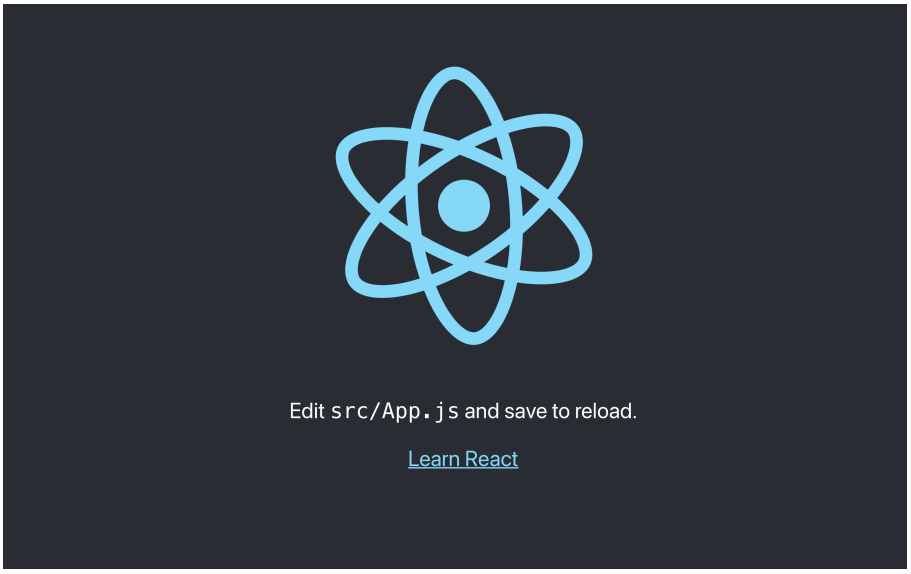


Figura 13: *Página inicial do* **React** *após a execução do projeto*

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*[8].

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.

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

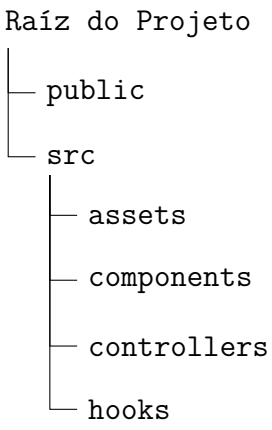


***Figura 14: React** — possível estrutura de pastas*

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:

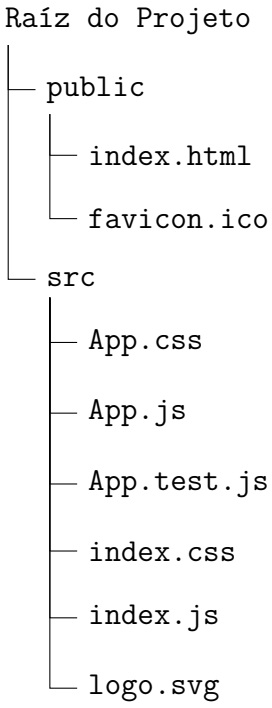


***Figura 15: React** — estrutura de pastas e ficheiros gerados pelo `create-react-app`*

Nota

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

Ficheiros Iniciais

```
<!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" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
↪   https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head>

  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>

</html>
```

Excerto de Código 15: *Ficheiro `index.html` de um projeto* **React**

```
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 16: *Ficheiro `index.jsx` de um projeto* **React**

```
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 17: *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 18: *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 16: *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`.

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 19: *Importação do* ***GraphQL*** *em* ***JavaScript***

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

Excerto de Código 20: *Importação dwo* ***GraphQL*** *em* ***TypeScript***

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 21: *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.

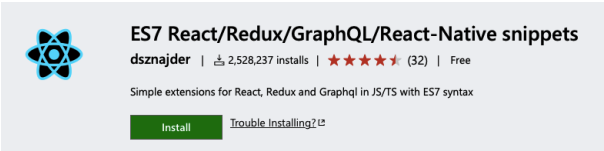


*Figura 17: Extensão **ES7 React/Redux/GraphQL/React-Native snippets***

Link

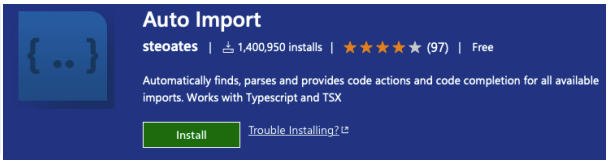


*Figura 18: Extensão **Auto Import***

Link

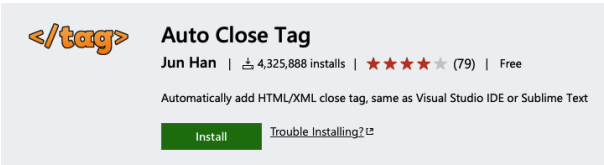


*Figura 19: Extensão **Auto Close Tag***

Link

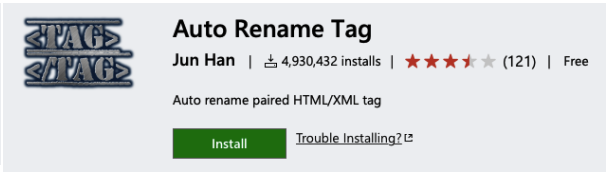


*Figura 20: Extensão **Auto Rename Tag***

Link

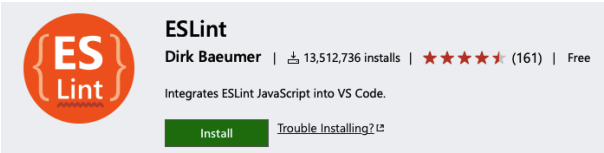


*Figura 21: Extensão **ESLint***

Link

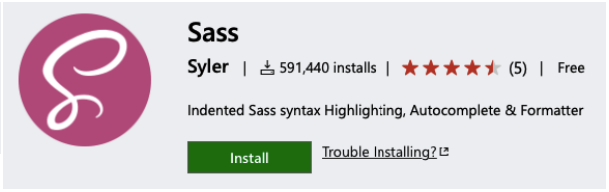


*Figura 22: Extensão **Sass***

Link

Nota

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