
ESCOLA

SUPERIOR

DE TECNOLOGIA

E GESTÃO

POLITÉCNICO

DO PORTO

P.PORTO



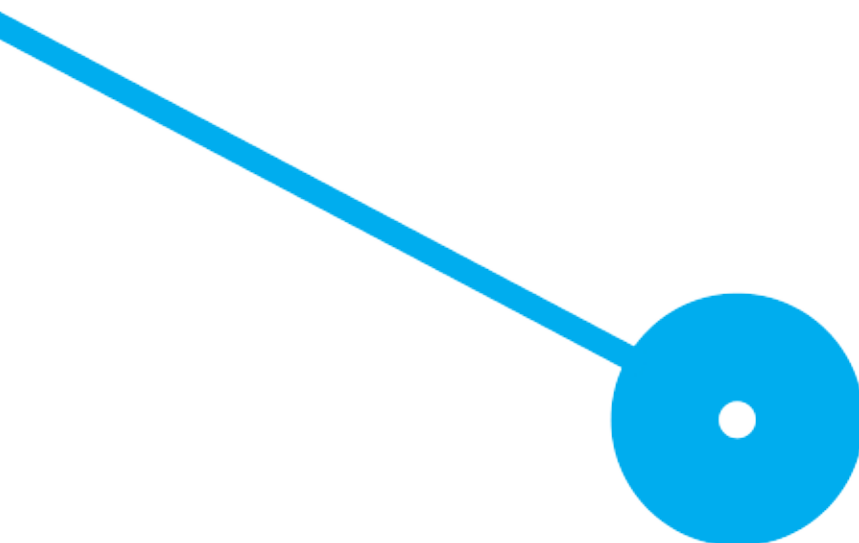
LICENCIATURA

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

L

LICENCIATURA

ENGENHARIA INFORMÁTICA

Plataforma de gestão de treinos para atletas de alta competição

Daniel Sousa

Professor Ricardo Jorge Santos

Este trabalho não inclui as críticas e sugestões feitas pelo Júri

[PÁGINA PROPOSITADAMENTE DEIXADA EM BRANCO]

Conteúdo

Lista de Figuras	v
Lista de Tabelas	vi
Lista de Excertos de Código	vii
Abreviaturas	viii
Glossário	x
Resumo	1
Apresentação do Autor	2
Apresentação da Entidade de Acolhimento	2
Convenções e Nomenclatura	3
1 Contextualização e Motivação	6
1.1 Introdução	6
1.2 Objetivos	6
1.3 Organização do Documento	6
2 Enquadramento Tecnológico	7
2.1 TypeScript	7
2.2 React	8

2.3	Sass	9
2.3.1	<i>Mixins</i>	10
2.3.2	Herança	10
2.3.3	Variáveis	12
2.4	NodeJS	13
2.5	GraphQL	15
2.6	PostgreSQL	17
3	Ambiente de Desenvolvimento	18
3.1	IDE	18
3.1.1	Visual Studio Code	18
3.1.2	WebStorm	18
3.2	Controlo de Versões	19
3.2.1	<i>Board</i>	19
3.2.2	<i>Issues</i>	19
3.2.3	<i>Merge Requests</i>	19
	Referências Bibliográficas	20
	Anexos	23
	TypeScript	23
	Instalação	23
	Configuração	24
	React	26
	Criação do Projeto	26
	Opções Adicionais	28
	Estrutura de Pastas	28
	Ficheiros Iniciais	30
	Execução do Projeto	32

GraphQL	34
Instalação	34
Visual Studio Code	35
Configurações	35
Extensões	37

Lista de Figuras

1	Jimmy Boys — Icon	3
2	TypeScript — logo	7
3	React — logo	8
4	Sass — logo	9
5	Ficheiro .sass compilado para um ficheiro .css	9
6	NodeJS — logo	13
7	Arquitetura do NodeJS em comparação com a arquitetura tradicional	14
8	GraphQL — logo	15
9	Demonstração do GraphiQL	15
10	PostgreSQL — logo	17
11	Visual Studio Code — logo	18
12	WebStorm — logo	18
13	Página inicial do React após a execução do projeto	28
14	React — possível estrutura de pastas	29
15	React — estrutura de pastas e ficheiros gerados pelo create-react-app	29
16	Projeto React executado com sucesso	32
17	Extensão ES7 React/Redux/GraphQL/React-Native snippets	38
18	Extensão Auto Import	38
19	Extensão Auto Close Tag	38
20	Extensão Auto Rename Tag	38
21	Extensão ESLint	38

22	Extensão Sass	38
----	--------------------------------	----

Lista de Tabelas

1	Principais diferenças entre TypeScript e JavaScript	8
---	---	---

Lista de Excertos de Código

1	Demonstração de excerto de código	5
2	Excerto de código com validação TypeScript	8
3	Definição e uso de <i>mixins</i> no Sass	10
4	Demonstração de herança no Sass	11
5	Código CSS resultante da compilação do excerto de código anterior	11
6	Demonstração de <i>placeholders</i> em Sass	12
7	Código CSS resultante da compilação do excerto de código com <i>placeholder</i>	12
8	Utilização de variáveis em Sass	12
9	Código CSS resultante do excerto de código com variáveis em Sass	13
10	Declaração e uso de variáveis em CSS	13
11	GraphQL — Exemplo de <i>query</i>	16
12	GraphQL — Exemplo de resposta à <i>query</i> realizada	16
13	TypeScript — Ficheiro tsconfig.json	25
14	Scripts para a execução do projeto em React	26
15	Ficheiro index.html de um projeto React	31
16	Ficheiro index.jsx de um projeto React	31
17	Ficheiro index.css de um projeto React	31
18	Ficheiro app.jsx de um projeto React	32
19	Importação do GraphQL em JavaScript	34
20	Importação do GraphQL em TypeScript	34
21	Configurações utilizadas no Visual Studio Code	37

Abreviaturas

LEI Licenciatura em Engenharia Informática

CTeSP Curso Técnico Superior Profissional

ESTG Escola Superior de Tecnologia e Gestão

ES *ECMAScript*

JSON *JavaScript Object Notation*

YAML *YAML Ain't Markup Language*

SQL *Structured Query Language*

HTML *Hyper Text Markup Language*

CSS *Cascading Style Sheets*

Sass *Syntactically Awesome Style Sheets*

JS *JavaScript*

TS *TypeScript*

NoSQL *No SQL –Not Only SQL*

NVM *Node Version Manager*

NPM *Node Package Manager*

JWT *JSON Web Token*

UI *User Interface*

UX *User Experience*

HTTP *HyperText Transfer Protocol*

CDN *Content Delivery Network*

CMS *Content Management System*

CRUD *Create, Read, Update, Delete*

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 Protótipo 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 contém 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 **ne-grito** 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 formato que é apresentado de seguida.

Nota

Informação da nota

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

! Erro Apresentado

Mensagem ou informações sobre o erro.

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

“Citação”

Autor ou Referência da citação

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

```
1 // Exemplo de Excerto de Código
2 console.log("Hello World");
```

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

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 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 exemplo 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 <i>Scripting</i>
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**.

2.2 React

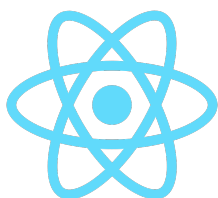


Figura 3:
React
— logo

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**.

2.3 Sass



Figura 4:
Sass – logo

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

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

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

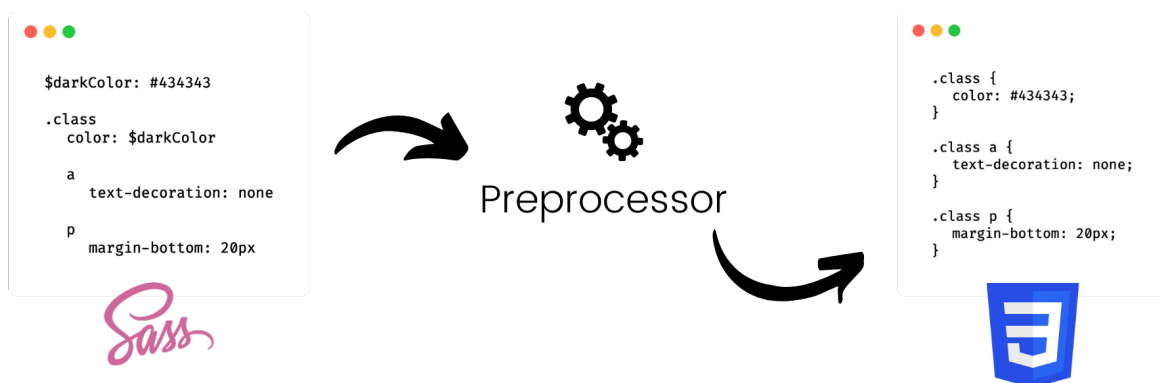


Figura 5: Ficheiro **.sass** compilado para um ficheiro **.css**

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.

2.3.1 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**.

2.3.2 Herança

No **Sass** também é possível realizar herança, nesta caso herança de estilos. O excerto de código¹ que é apresentado de seguida demonstra a utilização da herança no **Sass**.

```
1 .error
2     border: 1px #f00
```

¹Retirado da [documentação oficial](#).

```
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

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

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

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.

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

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

2.3.3 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**.

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

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

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**.

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

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

2.4 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 tradicional³.

³Imagem retirada de [3]

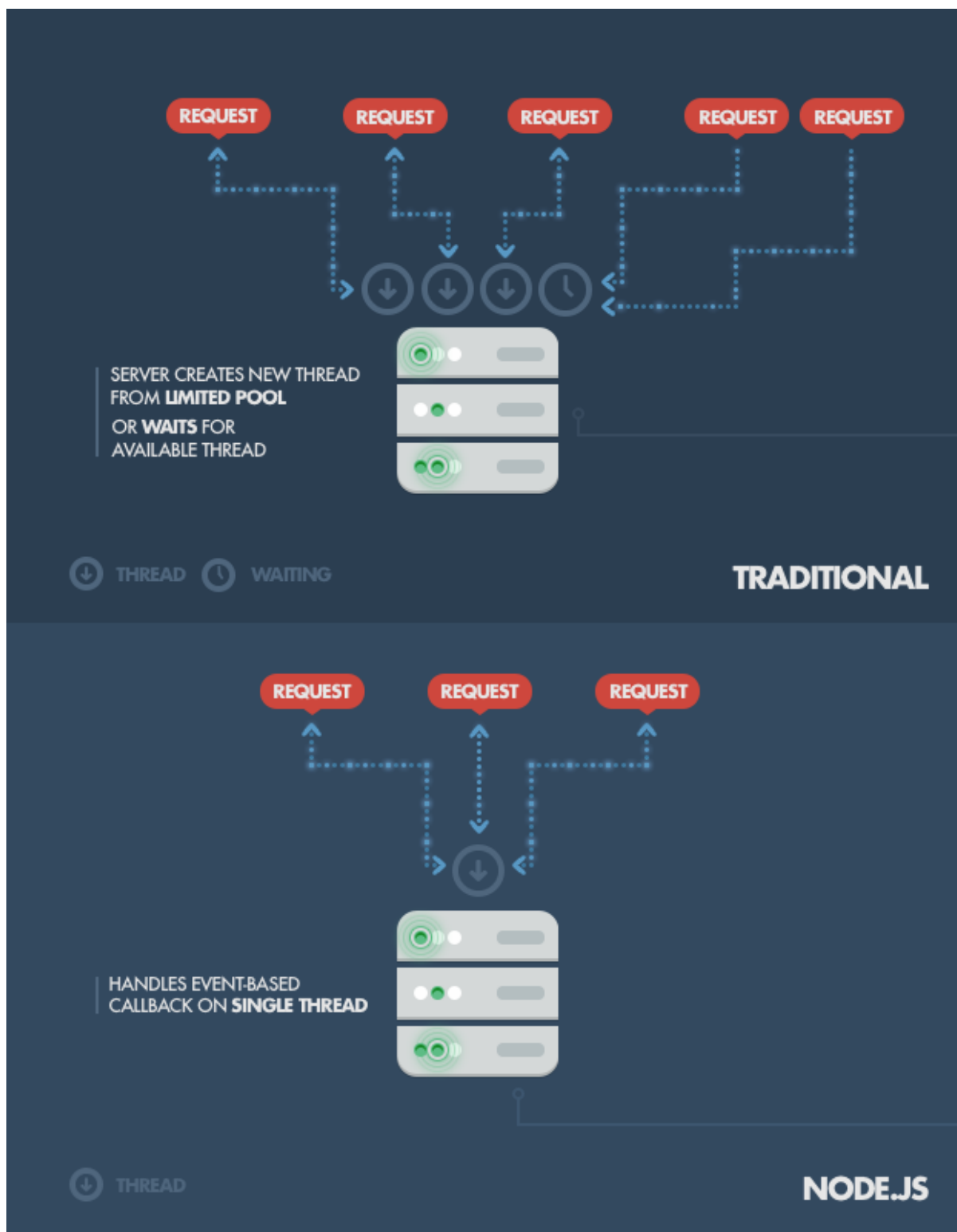


Figura 7: Arquitetura do **NodeJS** em comparação com a arquitetura tradicional

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.

2.5 GraphQL

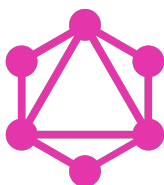


Figura 8:
GraphQL
— logo

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.

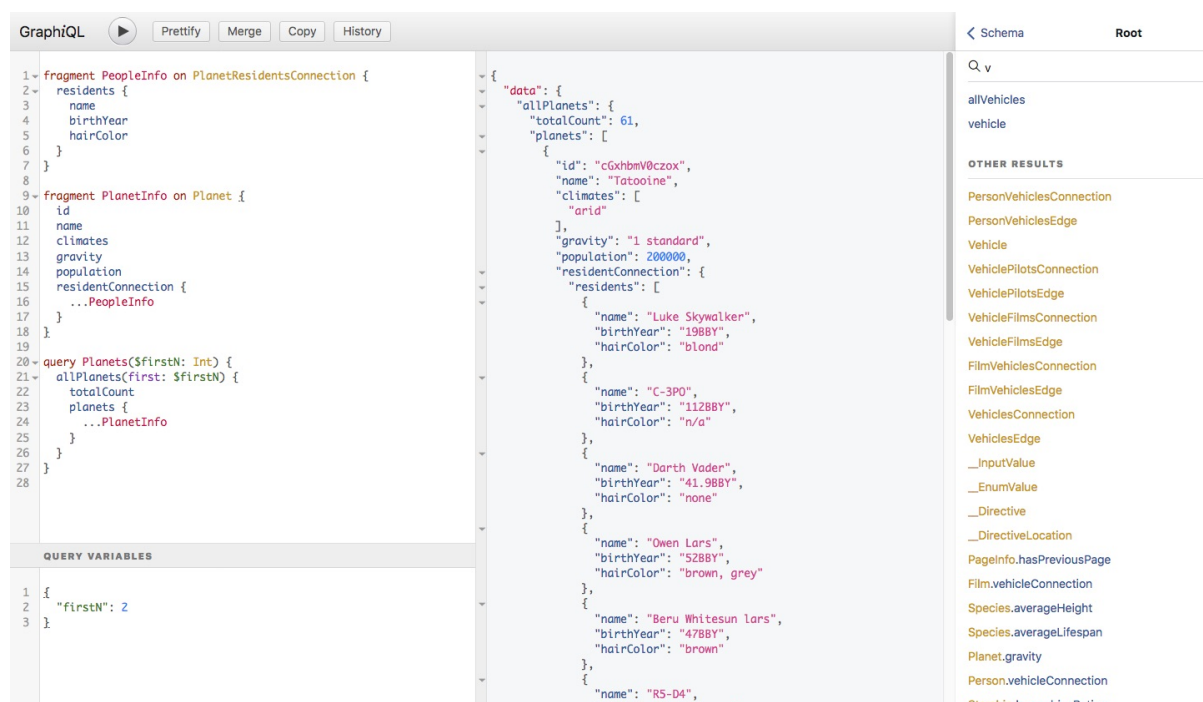


Figura 9: Demonstração do **GraphiQL**

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**).

```
1 {
2   hero {
3     name
4     # Queries can have comments!
5     friends {
6       name
7     }
8   }
9 }
```

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**, contendo a propriedade **data**, possuindo todos os resultados obtidos.

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

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

Em [anexo](#) é possível encontrar como realizar a instalação do **GraphQL**.

2.6 PostgreSQL

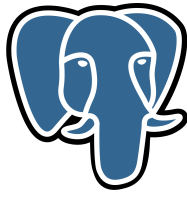


Figura 10:
Post-
greSQL
— logo

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

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

Capítulo 3

Ambiente de Desenvolvimento

3.1 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 otimizar o seu *workflow*.

3.1.1 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**: [15, 23, 26, 28]

3.1.2 WebStorm



Figura 12:
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.

3.2 Controlo de Versões

Durante o desenvolvimento de todo o projeto foi utilizado o **GitLab** para controlo de versões, usufruindo de todas as funcionalidades que este oferece. Nos pontos que se seguem será possível analisar toda a parte relativa ao **GitLab**, desde da criação e atribuição de *issues*, bem como a sua resolução implicando para tal a criação de uma nova *branch* e de um *merge request*.

3.2.1 Board

A *board* do **GitLab** é bastante versátil, permitindo criar *labels* de forma a organizar todas as tarefas, bem como indicar o seu estado atual. Na imagem que se segue é possível analisar a *board* existente para este projeto, bem como as respetivas *labels*.

Na *board* utilizada no decorrer do projeto é possível encontrar as seguintes colunas:

- **Open** — coluna por defeito do **GitLab** para todas as *issues* pendentes;
- **Closed** — coluna por defeito do **GitLab** para todas as *issues* encerradas/terminadas;
- **To Do** — coluna com todas as *issues* pendentes;
- **In Progress** — coluna com todas as *issues* em progresso;
- **Review** — coluna com todas as *issues* que aguardam análise e futura aprovação.

A imagem que se segue apresenta a *board* utilizada no decorrer do projeto.

3.2.2 Issues

3.2.3 Merge Requests

Referências Bibliográficas

- [1] Inglês. AltexSoft. Fev. de 2020. URL: <https://www.altexsoft.com/blog/typescript-pros-and-cons/>.
- [2] Cleber Campomori. *Precisamos falar sobre o TypeScript*. Português do Brasil. Abr. de 2017. URL: <https://www.treinaweb.com.br/blog/precisamos-falar-sobre-o-typescript/> (acedido em 17/03/2021).
- [3] Tomislav Capan. *Why The Hell Would I Use Node.js? A Case-by-Case Tutorial*. Inglês. TopTal. URL: <https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js> (acedido em 19/03/2021).
- [4] Alex Banks e Eve Porcello. *Learning React. Functional Web Development with React and Redux*. Inglês. O'Reilly, mai. de 2017. ISBN: 978-1-491-95462-1.
- [5] Diego Fernandes. *PWA: O que é? Vale a pena? Quando utilizar?* Português do Brasil. Rocketseat. 2018. URL: <https://blog.rocketseat.com.br/pwa-o-que-e-quando-utilizar/> (acedido em 20/03/2021).
- [6] *Getting started with PostgreSQL on Linux*. Inglês. RedHat. Mar. de 2019. URL: <https://www.redhat.com/sysadmin/postgresql-setup-use-cases> (acedido em 20/03/2021).
- [7] *Introduction to Node.js*. Inglês. NodeJS. URL: <https://nodejs.dev/learn> (acedido em 19/03/2021).
- [8] Pavels Jelisejevs. *Create React App: Get React Projects Ready Fast*. Inglês. SitePoint. Nov. de 2020. URL: <https://www.sitepoint.com/create-react-app/> (acedido em 15/03/2021).
- [9] Glaucia Lemos. *Conhecendo TypeScript!* Português do Brasil. Jan. de 2017. URL: <https://medium.com/@glaucia86/conhecendo-typescript-8fd74e8d13fb> (acedido em 17/03/2021).
- [10] Lenon. *Node.js – O que é, como funciona e quais as vantagens*. Português do Brasil. Set. de 2018. URL: <https://www.opus-software.com.br/node-js/>.
- [11] Samuel Martins. *Por que usar TypeScript como linguagem de programação*. Português do Brasil. Nov. de 2018. URL: <https://take.net/blog/devs/por-que-usar-typescript> (acedido em 17/03/2021).
- [12] *Node.js Introduction*. Inglês. W3Schools. URL: https://www.w3schools.com/nodejs/nodejs_intro.asp (acedido em 19/03/2021).

- [13] O que é GraphQL? Português do Brasil. RedHat. URL: <https://www.redhat.com/pt-br/topics/api/what-is-graphql> (acedido em 19/03/2021).
- [14] O que é ReactJS e porque devemos usá-lo? Português. EDIT. Dez. de 2018. URL: <https://edit.com.pt/blog/o-que-e-reactjs-e-porque-devemos-usa-lo/> (acedido em 15/03/2021).
- [15] Elad Ossadon. *The Ultimate VSCode Setup for Front End/JS/React*. Inglês. Nov. de 2018. URL: <https://medium.com/productivity-freak/the-ultimate-vscode-setup-for-js-react-6a4f7bd51a2> (acedido em 20/03/2021).
- [16] Joe Previte. *How to Migrate a React App to TypeScript*. Inglês. SitePoint. Mar. de 2020. URL: <https://www.sitepoint.com/how-to-migrate-a-react-app-to-typescript/> (acedido em 15/03/2021).
- [17] Joe Previte. *React with TypeScript: Best Practices*. Inglês. SitePoint. Set. de 2020. URL: <https://www.sitepoint.com/react-with-typescript-best-practices/> (acedido em 15/03/2021).
- [18] PWA (Progressive Web App): O Que É e Como Criar o Seu (+3 Exemplos). Português do Brasil. NeilPatel. URL: <https://neilpatel.com/br/blog/pwa-o-que-e/> (acedido em 20/03/2021).
- [19] *ReactJS. Notes for Professionals*. Inglês. GoalKicker.com, mai. de 2018. URL: <https://goalkicker.com/ReactJSBook/>.
- [20] Camilo Reyes. *20 Essential React Tools*. Inglês. SitePoint. Out. de 2020. (Acedido em 15/03/2020).
- [21] Filipe Portela e Ricardo Queirós. *Introdução ao Desenvolvimento Moderno para Web. Do front-end ao back-end: uma visão global!* Português. FCA, 2018. ISBN: 978-972-722-897-3.
- [22] *Setting up a Node development environment*. Inglês. Mozilla. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/development_environment (acedido em 19/03/2021).
- [23] Chris Sev. *React Tools for VS Code*. Inglês. Scotch. URL: <https://scotch.io/starters/react/editor-tools> (acedido em 20/03/2021).
- [24] Daniel de Souza Neris. *A importância do TypeScript*. Português do Brasil. Jul. de 2020. URL: <https://www.linkedin.com/pulse/import%C3%A2ncia-do-typescript-daniel-de-souza-neris/?articleId=6684939787595513856> (acedido em 17/03/2021).
- [25] *The Good and the Bad of Node.js Web App Development*. Inglês. AltexSoft. Out. de 2019. URL: <https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-node-js-web-app-development/> (acedido em 19/03/2021).
- [26] Michael Wanyoike. *10 Must-have VS Code Extensions for JavaScript Developers*. Inglês. Site-Point. Abr. de 2020. URL: <https://www.sitepoint.com/vs-code-extensions-javascript-developers/> (acedido em 20/03/2021).

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

Anexos

TypeScript

Instalação

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

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

A maneira mais comum é a instalação por projeto, visto que desta forma sempre que existir um *clone* do projeto e sejam instaladas as dependências¹, 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**.

¹Recorrendo 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.

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

²[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 raiz do projeto para garantir o seu correto funcionamento.

React

Criação do Projeto

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

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

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

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

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**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 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*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.

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

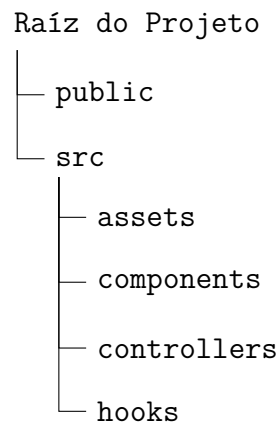


Figura 14: React – possível estrutura de pastas

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:

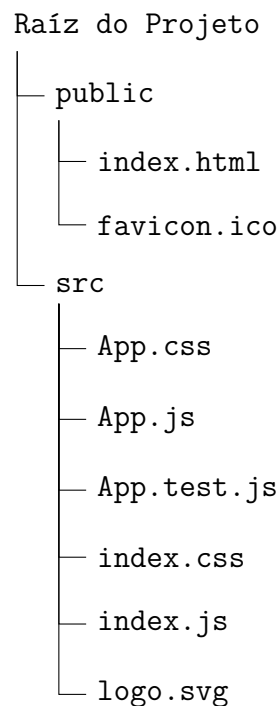


Figura 15: React – estrutura de pastas e ficheiros gerados pelo `create-react-app`

Nota

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

Ficheiros Iniciais

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="utf-8" />
6   <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
7   <meta name="viewport" content="width=device-width, initial-scale=1" />
8   <meta name="theme-color" content="#000000" />
9   <meta name="description" content="Web site created using create-react-app" />
10  <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
11  <!--
12    manifest.json provides metadata used when your web app is installed on a
13    user's mobile device or desktop. See
14    ↪ https://developers.google.com/web/fundamentals/web-app-manifest/
15    -->
16  <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
17  <!--
18    Notice the use of %PUBLIC_URL% in the tags above.
19    It will be replaced with the URL of the `public` folder during the build.
20    Only files inside the `public` folder can be referenced from the HTML.
21
22    Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
23    work correctly both with client-side routing and a non-root public URL.
24    Learn how to configure a non-root public URL by running `npm run build`.
25    -->
26  <title>React App</title>
27</head>
28<body>
29  <noscript>You need to enable JavaScript to run this app.</noscript>
30  <div id="root"></div>
31  <!--
32    This HTML file is a template.
33    If you open it directly in the browser, you will see an empty page.
34
35    You can add webfonts, meta tags, or analytics to this file.
36    The build step will place the bundled scripts into the <body> tag.
37
38    To begin the development, run `npm start` or `yarn start`.
39    To create a production bundle, use `npm run build` or `yarn build`.
40    -->
41</body>
42
43</html>
```

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

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

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

```
1 body {
2   margin: 0;
3   font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
4     'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
5     sans-serif;
6   -webkit-font-smoothing: antialiased;
7   -moz-osx-font-smoothing: grayscale;
8 }
9
10 code {
11   font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
12     monospace;
13 }
```

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

```
1 import React from 'react';
2 import './App.css';
3
4 function App() {
5   return (
6     <div className="App">
7       <header className="App-header">
8         <p>
9           Edit <code>src/App.jsx</code> and save to reload.
10        </p>
```

```

11     <a
12         className="App-link"
13         href="https://reactjs.org"
14         target="_blank"
15         rel="noopener noreferrer"
16     >
17         Learn React
18     </a>
19 </header>
20 </div>
21 );
22 }
23
24 export default App;

```

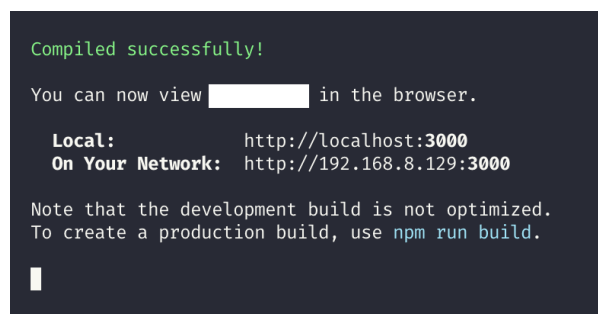
Excerto de Código 18: Ficheiro *app.jsx* de um projeto **React**

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:



```

Compiled successfully!

You can now view [redacted] in the browser.

Local:      http://localhost:3000
On Your Network:  http://192.168.8.129:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

```

Figura 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 do **GraphQL** em **TypeScript***

Visual Studio Code

Configurações

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

```

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

```

```

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

```

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](#)

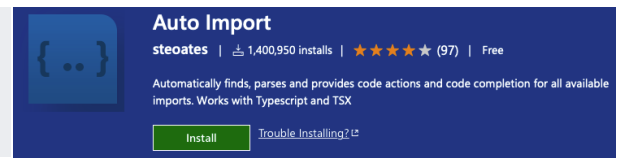
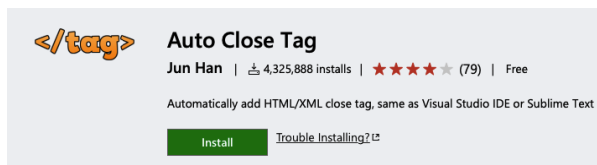


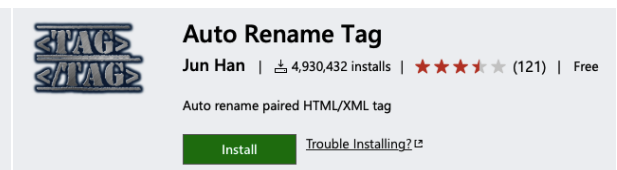
Figura 18: Extensão Auto Import

[Link](#)



**Figura 19: Extensão Auto Close
Tag**

[Link](#)



**Figura 20: Extensão Auto
Rename Tag**

[Link](#)

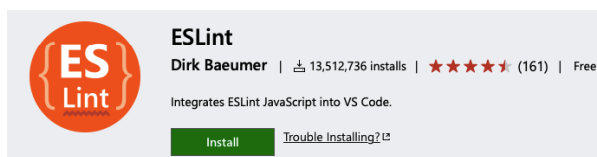


Figura 21: Extensão ESLint

[Link](#)

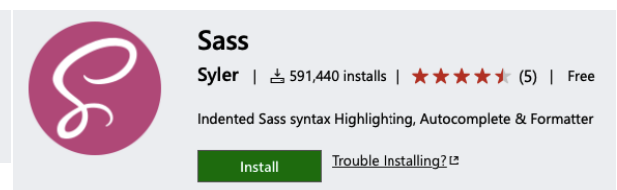


Figura 22: Extensão Sass

[Link](#)

Nota

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