ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

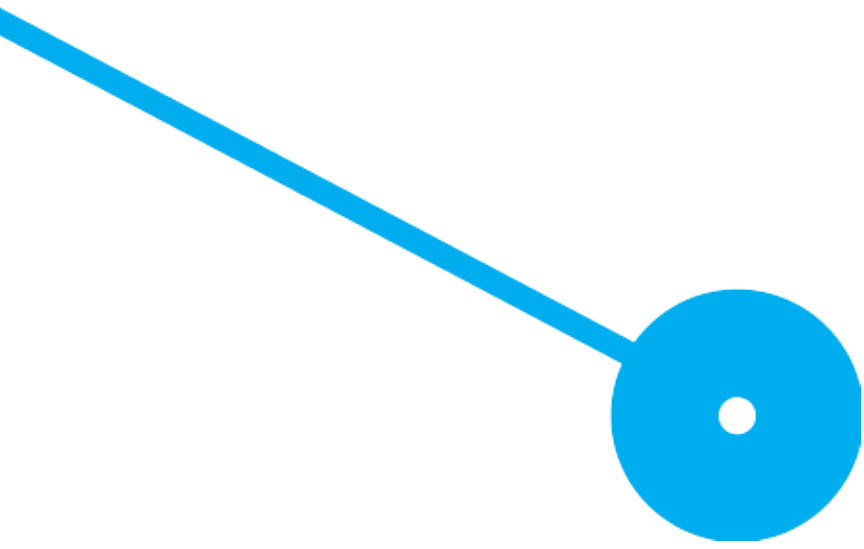
P.PORTO

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 Excertos de Código

# Abreviaturas

**LEI** Licenciatura em Egenharia Informática

**ESTG** Escola Superior de Tecnologia e Gestão

**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*

**PWA** *Progressive Web App*

**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*** São nada mais nada menos do que módulos/dependências que podem ser adicionados e utilizados ao longo de um determinado projeto.

***Scripts*** TODO

***Template*** TODO

***Build*** TODO

# Resumo

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

# Apresentação do Autor

# Apresentação da Entidade de Acolhimento

# 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.

**Nota**

Informação da nota

Para além das notas e, recorrendo ao mesmo esquema, sempre que seja necessário apresentar informações relativas algum erro será utilizado o exemplo que se segue.

⚠ Erro Apresentado

Mensagem ou informações sobre o erro.

Já para apresentar excertos de código ao longo deste relatório, optou-se por utilizar o seguinte esquema:

```
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

1.1 Introdução

1.2 Objetivos

1.3 Organização

# Capítulo 2

# Tecnologias

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 2.1:*** ***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."*

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; }'`.

### 2.1.1 Instalação

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

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

A maneira mais comum é a instalação por projeto, visto que desta forma sempre que existir um *clone* do projeto e sejam instaladas as dependências[1], o **TypeScript** será também instalado e pronto a ser utilizado.

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

**Nota**

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

### 2.1.2 Configuração

O **TypeScript** permite que o programador realizar determinadas configurações no seu projeto recorrendo a um ficheiro, no caso o ficheiro `tsconfig.json`,[2] neste ficheiro, tal como é possível encontrar em anexo, é possível definir desde configurações relacionadas com a estrutura de pastas do projeto, bem como definir *paths* para ajudar a manter as importações realizadas mais "enxutas".

No exemplo em anexo 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] Recorrendo a `yarn install` ou `npm install`.
[2] Documentação Oficial

## 2.2 React

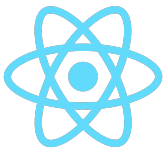


*Figura 2.2: React — logo*

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.

### 2.2.1 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 3:*** *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**

Estes ficheiros é possível encontrar em anexo, mais precisamente em **ficheiros React**.
É ainda importante referir que estes ficheiros são apenas a base para colocar um projeto **React** a funcionar.

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"

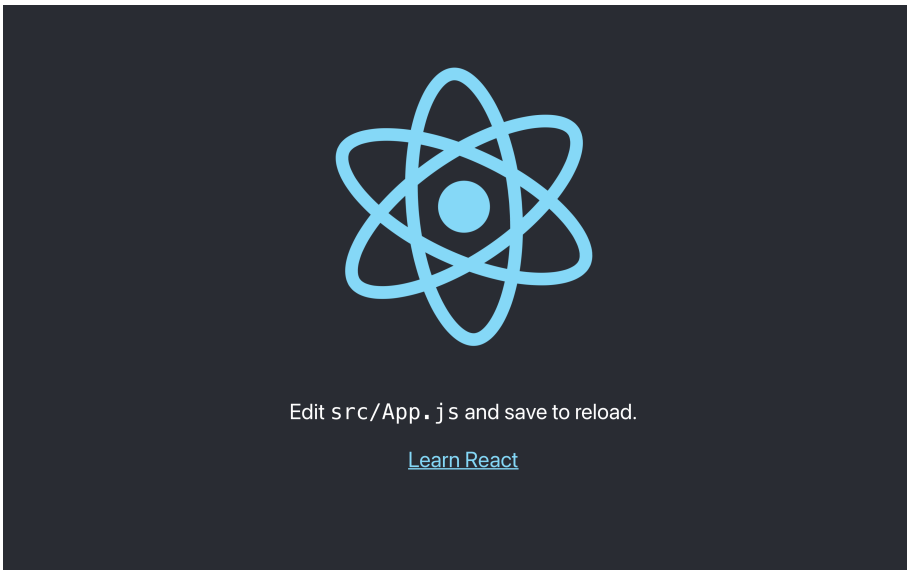



***Figura 2.3:*** *Página inicial do* ***React*** *após a execução do projeto*

#### 2.2.1.1 Opções Adicionais [4]

Na criação de um projeto **React** através do `create-react-app` é possível especificar o *template* a usar , não sendo de uso obrigatório. Neste *template* é possível especificar, por exemplo, para ao gerar o projeto **React** gerar com **TypeScript** e não **JavaScript**.

Além do *template* é ainda possível especificar o *package manager* utilizado, recorrendo à opção `--use-npm`, isto para usar o **NPM** como *package manager*[5].

### 2.2.2 Estrutura de Pastas

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

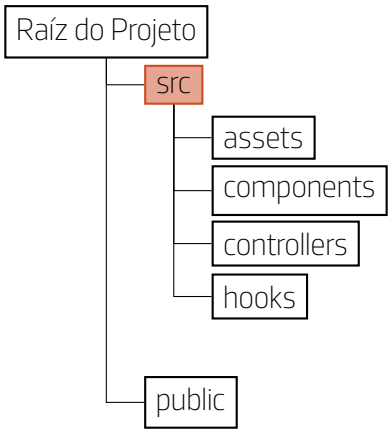



***Figura 2.4:*** ***React*** *– possível estrutura de pastas*

[4] **Nota:** as opções referidas são apresentadas acima como `[<options>]`.
[5] No caso de possuir o **Yarn** instalado.

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

**Nota**

É importante relembrar que consoante o uso de **TypeScript** ou **JavaScript** será possível encontrar ficheiros `.tsx`, `.ts`, `.jsx` ou `js`.
Além da extensão dos ficheiros será possível encontrar na raíz do projeto um ficheiro de configuração, podendo ser:

- `tsconfig.json` – a quando a utilização de **TypeScript**;
- `jsconfig.json` – a quando a utilização de **JavaScript**.

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:

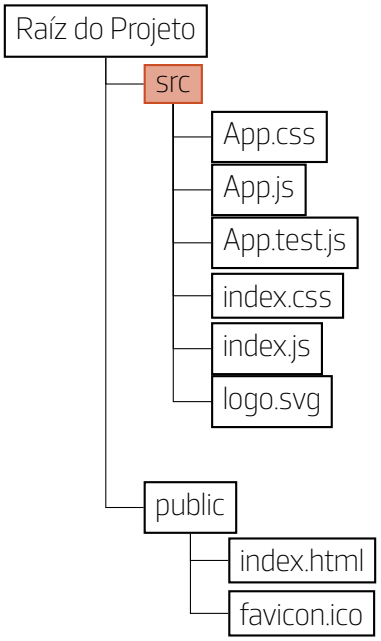



***Figura 2.5: React** – estrutura de pastas e ficheiros gerados pelo `create-react-app`*

### 2.2.3 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`, para isso basta utilizar 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 2.6:*** *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`.

# Capítulo 3

# Ambiente de Desenvolvimento

# Capítulo 4

# Controlo de Versões

## 4.1 *Issues*

## 4.2 *Merge Requests*

## 4.3 *Board*

# Referências Bibliográficas

# Anexos

## TypeScript — Configurações

---

```
{
  "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": [
    "src"
  ]
}
```

*Excerto de Código 4: TypeScript* — *Ficheiro* `tsconfig.json`

**Nota**

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.

# Ficheiros React

```
<!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 5:** *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 6:** *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 7:** *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.
```

```
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 8:** *Ficheiro* `app.jsx` *de um projeto* **React**

# 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",
```

```
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 9:*** *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 inicias (`{}`) no código apresentado e colocar as restantes configurações.