

> **Ficha Prática Nº7 (Introdução ao React – Componentes e Props)**

O **React** é uma biblioteca JavaScript, muito utilizada na indústria, a qual permite o desenvolvimento de interfaces (UI) rápidas e interativas. A compreensão do funcionamento base do React exige a clarificação dos seguintes conceitos:

- componente;
- árvore de componentes;
- método *render()*;
- **JSX** (*JavaScript Syntax Extension*);
- Criar/Configurar conversão JSX -> JS;
- *class component / functional component*.

As aplicações React baseiam-se na implementação de diversos **componentes reutilizáveis e independentes**, que podem ser vistos como “peças” que irão compor a interface pretendida. Assim, este tipo de arquitetura baseada em **componentes** permite a divisão da aplicação em diversas partes (componentes), que uma vez juntas e interligadas possibilitam a criação de uma UI complexa.

Qualquer aplicação React é, essencialmente, uma **árvore de componentes**, o que implica que:

- Uma aplicação em React é composta, pelo menos, por um componente, que representa a aplicação total.
- Este componente pode conter filhos (situação mais frequente), os quais, são na realidade, outros componentes React.

A implementação de um componente React consiste na:

- a) implementação de uma **classe ou função** em JavaScript, dependendo da abordagem utilizada, podendo conter algum estado. Os estados são os dados que se pretende apresentar quando o componente é renderizado;
- b) implementação do método **render**, o qual é responsável por descrever como a UI deve ser apresentada. O resultado do método render é um objeto JavaScript mapeado com um elemento DOM, de forma simplificada, sem ser necessário recorrer às API DOM existentes nos browsers. Assim, quando se altera o estado do componente, o React irá automaticamente alterar o DOM para coincidir com esse estado.

JSX (*JavaScript Syntax Extension = JavaScript + XML*) é a sintaxe utilizada pelo React. Esta extensão de sintaxe de JavaScript disponibiliza funcionalidades adicionais que permitem simplificar a leitura e escrita de código HTML dentro do JavaScript. Embora o uso do JSX não seja obrigatório, é fundamental na criação de aplicações React. No entanto, como os browsers desconhecem esta sintaxe, é necessário converter o JSX em JavaScript para que dessa forma o browser o consiga interpretar, sendo, portanto, necessário utilizar um compilador, como exemplo, o compilador/transpilador **Babel JavaScript**, que será adotado nas aulas práticas.

Existem essencialmente duas abordagens a criar e configurar uma aplicação React para converter o código JSX em Javascript:

- O **browser converte o JSX para o JS**, de forma automática, em tempo de execução. Para isso é necessário referenciar um ficheiro de script no código, o qual será responsável por transformar o JSX em JavaScript aquando do carregamento da página. **Esta será a abordagem utilizada nesta ficha prática**, como forma de iniciação, no entanto, este método não é usado em implementação de aplicações reais uma vez que o desempenho da aplicação diminui sempre que o browser traduz JSX em JS. Esta opção, também dificulta a manutenção do código.
- Configurar um **ambiente de desenvolvimento com o Node**, juntamente com um conjunto de ferramentas de desenvolvimento, simplificando a gestão das dependências. Esta é o tipo de abordagem utilizado no desenvolvimento de aplicações reais, pelo será adotada nas fichas práticas seguintes.

Quando se recorre a JSX na implementação de uma aplicação em React é necessário ter em consideração algumas características desta linguagem, nomeadamente:

- As *tags* do HTML e dos componentes devem ser sempre fechadas `</>`;
- Alguns atributos HTML mudam de nome, como por exemplo, o atributo **“class”** passa a ser **“className”** (pois a palavra *class* faz referência à criação de classes em JavaScript). Além disso, o JSX usa a notação *camelcase* para nomear atributos HTML.
- No método `render()`, não é possível retornar mais de um elemento HTML de uma só vez. Esta limitação pode ser contornada envolvendo todos os elementos num elemento pai (exemplo: um elemento `div`), ou então como alternativa, entre *tags* vazias (designados *React Fragments* `<> </>`);
- JSX permite escrever expressões entre `{}`, podendo ser qualquer expressão JS ou uma variável React.
- Para adicionar código JavaScript no meio do JSX, é necessário escrever entre `{}`. São permitidas apenas expressões que avaliem algum valor, como string, número, array um método `map`, entre outros. (expressões estas também conhecidas como *JSX Expression Syntax*).

- Quando um tipo de elemento se inicia com letra minúscula, refere-se a um *built-in component*, por exemplo o `<div>` ou ``. Tipos de elementos que iniciem com letra maiúscula, correspondem a um componente definido, por exemplo: `<Linguagens />`

Como referido anteriormente, um **componente** é um bloco de código reutilizável e independente, que gera (via JSX) elementos HTML, e que permite dividir a interface do utilizador em partes menores. Existem duas formas de criar os componentes em React, por recurso a uma:

- **classe** (*class component*);
- **função** (*functional component*).

A diferença clara entre estes dois tipos de componentes (*class* ou *functional*), reside na sua sintaxe. Um componente funcional é uma função JavaScript simples que retorna um elemento do React(JSX). Um componente de classe é uma classe JavaScript que estende `React.Component` e possui o método de renderização.

Atualmente, os componentes funcionais são os mais populares e mais utilizados na implementação de aplicações React, uma vez que a introdução de *Hooks*, a partir da versão 16.8 do React, permitiram adicionar estados e outras características do React, sem recorrer à implementação de classes. Como tal, no contexto das aulas práticas serão, essencialmente, usados componentes funcionais.

Os nomes dos **componentes** em React **devem iniciar sempre com letra maiúscula**, como por exemplo `<LinguagensScript />`. Esta é a forma utilizada pelo React para distinguir entre um componente, ou uma tag html, que devem iniciar por minúscula. Seguindo também a convenção, é habitual que o principal componente seja o designado `<App />`.

Os exemplos seguintes exemplificam a criação de um componente muito simples. A figura 1 refere-se à criação de um componente funcional, o `FunctionalComponent`, que retorna um elemento h1 com o texto “Linguagens Script”.

```
function FunctionalComponent() {
  return <h1>Linguagens Script</h1>;
}
```

Figura 1 - Componente funcional sem estado

A Figura 2 implementa um componente funcional com o mesmo objetivo, mas recorrendo à sintaxe de uma *arrow function*.

```
const FunctionalComponent = () => <h1>Linguagens Script</h1>;
```

Figura 2 - Componente Funcional anterior recorrendo a uma arrow function

A figura 3, contém um componente de classe, o **ClassComponent**, que também retorna um elemento h1, com o texto “Linguagens Script”.

```
class ClassComponent extends React.Component {
  render() {
    return <h1>Linguagens Script</h1>;
  }
}
```

Figura 3 - Componente de Classe

Um aspeto muito importante no funcionamento dos componentes é a forma como estes comunicam entre si. Na realidade, o React tem um objeto especial, chamado de **props** (que significa propriedades), que permite transportar dados de um componente para o outro.

As **props** transportam dados apenas numa direção (do elemento pai para os elementos filhos; árvore de componentes) e, portanto, não é possível com base nas props passar dados do elemento filho para o pai, nem para componentes ao mesmo nível.

Adicionalmente existe uma regra em relação às props: **são apenas de leitura**. O exemplo seguinte apresenta um exemplo do uso das props de forma a que os componentes se tornem mais flexíveis.

```
const Exemplo = props => <h1>LS-{props.capitulo}</h1>;
const Exemplo2 = ({capitulo}) => <h2>LS-{capitulo}</h2>;
const Exemplo3 = p => <h2>LS - {p.children}</h2>;

ReactDOM.render(
  <React.StrictMode>
    <Exemplo capitulo="JavaScript" />
    <Exemplo2 capitulo="React" />
    <Exemplo3>HTML e CSS</Exemplo3>
  </React.StrictMode>,
  document.getElementById("root")
);
```



Figura 4 – Exemplo com Props em React

Algumas ligações úteis para esta parte introdutória ao React:

- <https://react.dev/learn/your-first-component>
- <https://react.dev/learn/passing-props-to-a-component>
- <https://react.dev/learn/add-react-to-an-existing-project>
- <https://react.dev/learn/writing-markup-with-jsx>
- <https://react.dev/learn/javascript-in-jsx-with-curly-braces>

> **Preparação do ambiente (Parte Introdutória)**

No **browser**, instale a extensão “**React Developer Tools**”, que permite inspecionar e analisar os componentes React existentes na página, entre outras funcionalidades:

- “[React Developer Tools](#)” no *chrome*
- “[React Developer Tools](#)” no *Firefox*

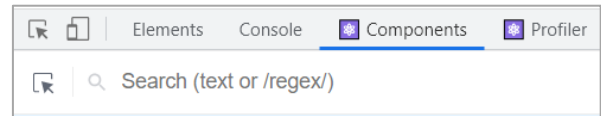


Figura 5 – React Developer Tools

Inicie o **Visual Studio Code** e instale as seguintes extensões úteis (caso ainda não estejam instaladas):

- **Live Server**
- **Simple React Snippets**
- **Prettier – Code Formatter**
 - > Depois de instalar, ative este Code formatter, nas preferências.
 - > File > Preferences > Settings (CTRL + ,)
 - > Procure por Format, e na opção “Default Formatter” selecione o Prettier–Code Formatter

Descompacte o ficheiro **ficha7.zip**.

- Abra a pasta obtida no **workspace no VSCode** e visualize a página **index.html**, recorrendo ao “Live Server” (botão direito do rato -> “**Open with Live Server**” ou então clicando no “**Go Live**”, existente na barra do rodapé).
- No browser deverá ser apresentado o título “Linguagens Script” com o aspeto da figura seguinte.

Linguagens Script!

Figura 6 – Página Principal

Parte I – Análise do código

1> Nesta primeira parte, pretende-se demonstrar uma página simples em React, adicionando as bibliotecas necessárias diretamente na página HTML.

a. Analise o código do ficheiro **index.html** e efetue a leitura do texto seguinte.

Para que a página HTML possa executar React e além disso seja possível manipular os elementos DOM de uma página, foi necessário adicionar ao **index.html**, os *scripts* React e ReactDOM. Estes fazem a ligação à API do React, bem como aos vários elementos que o React necessita para funcionar com o DOM.

Para além desses scripts, é necessário também adicionar uma referência ao compilador **Babel JavaScript**, que irá adicionar a capacidade de transformar o código JSX em JavaScript. O código seguinte apresenta a ligação aos scripts referidos.

```
<script src="https://unpkg.com/react@18.2.0/umd/react.development.js" crossorigin></script>
<script src="https://unpkg.com/react-dom@18.2.0/umd/react-dom.development.js" crossorigin></script>
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
```

Como é possível verificar, o corpo do documento index.html contém apenas uma única div, , sendo esta a abordagem geralmente seguida numa aplicação React. Todo o restante conteúdo HTML, interface, e outros elementos será construído pelo JavaScript, que irá depois colocar todos esses elementos nessa div principal. Este elemento HTML é, por convenção, geralmente identificado como *root*. O exemplo seguinte segue esta abordagem, embora pudesse ser usado outro nome para o id. Repare ainda que o tipo “**text/babel**” é obrigatório para usar Babel.

```
<body>
  <div id="root"></div>
  <script type="text/babel" src="js/index.js"></script>
</body>
```

Figura 7 – Ligação ao Babel

Por fim, é adicionada uma ligação ao ficheiro **index.js**, que contém o código a ser injetado na div **root**, assim como, o método **render()**, que irá renderizar o primeiro componente react.

b. Análise do ficheiro **index.js**

O código existente no ficheiro index.js, inclui duas partes essenciais: o componente **LinguagensScript** e a definição do **método render** que especifica o HTML que será gerado, e o local DOM onde o mesmo será renderizado, que no exemplo apresentado, será no elemento com id **root**.

```
const containerRoot=document.getElementById("root");
ReactDOM.render( <React.StrictMode>
  <LinguagensScript />
</React.StrictMode>, containerRoot);
```

Figura 8 – Método Render

O componente **LinguagensScript** é um componente funcional o qual apenas define um título de nível 1, com o texto Linguagens Script, como apresentado de seguida:

```
function LinguagensScript() {
  return <h1>Linguagens Script!</h1>
}
```

Figura 9 - Componente LinguagensScript

Parte II – Alterações e Estilização do Componente

2> Após analisar o código anterior (Parte I), implemente as seguintes alterações:

- Transforme a função *LinguagensScript* numa *arrow function*;
- Sem alterar a função, especifique código para que a aplicação fique com o aspeto da figura seguinte.

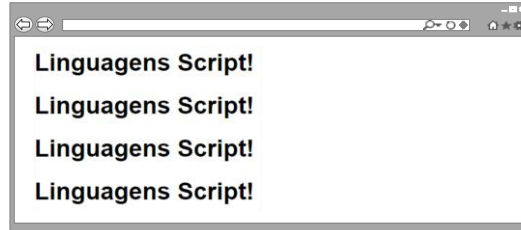


Figura 10 - Exercício

- Altere a função, por forma a que o componente receba um **nome** de forma a apresentar **boas vindas**. Consulte o apoio apresentado nas páginas iniciais da ficha. O componente deverá ser invocado, da seguinte forma, e ficar com o aspeto da figura 11.

```
<LinguagensScript nome="José Antunes" />
```

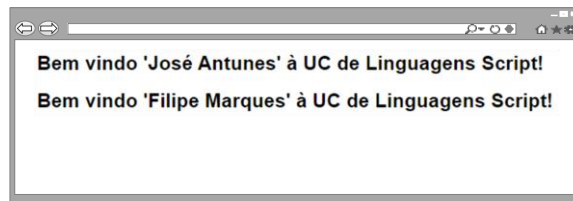


Figura 11 - Exercício

3> Existem várias formas para estilizar um componente. É possível aplicar estilos *inline*, através de folhas de estilo, pré-processadores CSS, *styled components*, CSS modules, entre outros.

Para especificação de um estilo *inline*, com recurso ao atributo *style*, pode efetuar da seguinte forma:

```
const divStyle = {
  margin: '40px',
  border: '5px solid red'
};
const pStyle = {
  textAlign: 'center'
};
const Exemplo4 = ()=>( <div style={divStyle}>
  <p style={pStyle}> Aplicação de CSS Inline - LS</p>
</div>
);

ReactDOM.render(
  <React.StrictMode>
    <Exemplo4 />
    <Exemplo4 />
  </React.StrictMode>,
  document.getElementById("root"));
```



Figura 12 – Estilos em React

Repare que algumas propriedades definidas se diferenciam, de forma residual, das propriedades CSS. Por exemplo, a propriedade **text-align**, como é **constituída por duas palavras, necessita de uma transformação** para poder ser usada em React.

Assim, a **regra estabelece que** as propriedades CSS constituídas por palavras separadas por um hífen, tais como como *background-color*, *border-radius* ou *font-family*, entre outras, são transformadas numa única palavra, no formato *camelCase* (o hífen é removido e a primeira letra da segunda palavra passa a maiúscula). Por exemplo, em React as propriedades **background-color** e **border-radius** passam respetivamente a `backgroundColor` e `borderRadius`. ,.

Apesar da abordagem CSS *inline* apresentar várias desvantagens, isto é, usando diretamente o atributo `style` para estilizar os elementos, e esta forma não ser recomendada, seja por questões de desempenho, como também na limitação das regras que se pode definir, será usada nas próximas secções por conveniência e como forma introdutória. Na maioria dos casos, deve ser usado o atributo **className**, para referenciar classes definidas num ficheiro CSS externo.

O atributo **style** é muito utilizado em aplicações React para adicionar estilos que são calculados dinamicamente, no momento da renderização.

O atributo `style` aceita um **objeto** JavaScript com propriedades *camelCase* em vez de uma string CSS. Isso é consistente com a propriedade JavaScript do estilo DOM, é mais eficiente e evita falhas de segurança XSS. Em certas propriedades numéricas, o React anexa de forma automática o sufixo “px”. Se se pretender usar unidades diferentes, o valor deve ser especificado como uma *string* com a unidade desejada.

- a. Para aplicar estilos no `h1` com recurso ao atributo `style` directamente no `h1`, declare, no ficheiro `index.js`, o objeto **styleH1**. Este objeto deverá conter as seguintes propriedades:

```
fontFamily: 'sans-serif',
textDecoration: 'underline',
color: 'brown'
```

Aplique este estilo ao `h1` existente, com recurso ao atributo `style` de forma a obter a página com o aspeto seguinte.



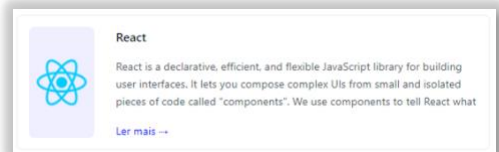
Figura 13 – Exercício

Parte II – Criação de componente em React

- 4> De forma a exercitar a criação de componentes, pretende-se criar um componente em React como o que se apresenta nas figuras ao lado.

O componente é composto por um logotipo, um título, uma descrição e um url. Além disso, o componente é responsivo, apresentando um layout diferente de acordo o *viewport* disponível.

Todo o CSS necessário para a criação do componente já se encontra no ficheiro `componenteReact.css`, portanto, não é necessário especificar regras CSS para estilizar o componente.




a. Apresenta-se, de seguida, o código html que especifica um elemento. Considere as seguintes características:

- O elemento com **class** *container* é o bloco que envolve todos os componentes e o *wrapper* corresponde ao bloco de um componente. Se se pretender criar 2 componentes, serão dois *wrappers* dentro do container.
- As imagens encontram-se dentro da pasta *imagens*.
- O logotipo encontra-se enquadrado num div com determinadas propriedades
- O texto é composto pelo **título**, a **descrição** e pelo **link** referente ao texto “Ler mais”.

```
<div class="container">
  <div class="wrapper">
    <div class="logo">
      
    </div>
    <div class="text">
      <h2>React</h2>
      <p>Descrição do React</p>
      <a href="https://reactjs.dev/" target="_blank">Ler mais</a>
    </div>
  </div>
</div>
```


O mesmo componente, invocado com dados diferentes, e apresentado em *viewports* diferentes. As figuras seguintes apresentam o comportamento do elemento em diferentes layouts.



React

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called "components". We use components to tell React what we want to


[Ler mais →](#)



JavaScript

JavaScript (JS) is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also


[Ler mais →](#)



Ember

Ember.js is a productive, battle-tested JavaScript framework for building modern web applications. It includes everything you need to build rich UIs that work on any device.

[Ler mais →](#)




Vue

Vue is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS and JavaScript, and provides a declarative and component-based programming model that helps you efficiently develop user interfaces. be

[Ler mais →](#)


Figura 6 a) – Layout para Viewport inferior a 640px



React

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called "components". We use components to tell React what


[Ler mais →](#)



JavaScript

JavaScript (JS) is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser


[Ler mais →](#)



Ember

Ember.js is a productive, battle-tested JavaScript framework for building modern web applications. It includes everything you need to build rich UIs that work on any device.

[Ler mais →](#)




Vue

Vue is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS and JavaScript, and provides a declarative and component-based programming model that helps you efficiently develop

[Ler mais →](#)

Figura 6 b) – Layout a partir de Viewport com largura de 640px



React

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small


[Ler mais →](#)



JavaScript

JavaScript (JS) is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions. While it is most well-


[Ler mais →](#)



Ember

Ember.js is a productive, battle-tested JavaScript framework for building modern web applications. It includes everything you

[Ler mais →](#)



Vue

Vue is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS and JavaScript, and provides a

[Ler mais →](#)

Figura 7 – Layout a partir de Viewport com largura de 768px

5> Exercício: Tendo em consideração o código HTML, apresentado na página anterior, implemente os seguintes passos:

- a. Não altere o ficheiro index.html
- b. No ficheiro index.js, elimine ou coloque em comentários as funções e os componentes já existentes, criados anteriormente. Deixe apenas o método render e a const containerRoot.
- c. Tendo em consideração a estrutura HTML apresentada na página 8, implemente um componente funcional, de nome **InfoComponent**, o qual deverá ser invocado como se apresenta no trecho de código seguinte. Alerta-se ainda que, as imagens encontram-se dentro da pasta images, e todos os componentes devem estar envolvidos com um div com classe **container**.

```
<InfoComponent title="React" src="react.png" url="https://react.dev/"> React is a
declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you
compose complex UIs from small and isolated pieces of code called "components". We use components
to tell React what we want to see on the screen. When our data changes, React will efficiently
update and re-render our components. A component takes in parameters, called props (short for
"properties"), and returns a hierarchy of views to display via the render method.
</InfoComponent>
<InfoComponent title="Javascript" src="javascript.png"
url="https://developer.mozilla.org/en-US/docs/Web/JavaScript">
JavaScript (JS) is a lightweight, interpreted, or just-in-time compiled programming language with
first-class functions. While it is most well-known as the scripting language for Web pages,
many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat.
JavaScript is a prototype-based, multi-paradigm, single-threaded, dynamic language, supporting
object-oriented, imperative, and declarative (e.g. functional programming) styles. Read more about
JavaScript.
</InfoComponent>
```

d. Verifique no browser a página implementada com o “live browser” e adicione mais elementos com a seguinte informação:

- Ember
 - ember.png
 - <https://emberjs.com/>
 - Ember.js is a productive, battle-tested JavaScript framework for building modern web applications. It includes everything you need to build rich UIs that work on any device.
- Vue
 - vue.png
 - <https://vuejs.org/>
 - Vue is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS and JavaScript, and provides a declarative and component-based programming model that helps you efficiently develop user interfaces, be it simple or complex.