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

O **React** é uma biblioteca JavaScript, fortemente utilizada na indústria, que permite o desenvolvimento de interfaces (UI) rápidas e interativas. 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 juntas e interligadas permitem a criação de uma UI complexa.

O desenvolvimento de qualquer aplicação em React é composta, pelo menos, por um componente, que irá representar a aplicação total, podendo conter filhos, que, na realidade, se tratam de outros componentes React. Assim, qualquer aplicação React é, essencialmente, uma árvore de componentes. A implementação de um componente React consiste na implementação de uma classe ou função em JavaScript, dependendo da abordagem utilizada, podendo conter algum estado, e na implementação do método render. Os estados são os dados que se pretende apresentar quando o componente é renderizado e o método render é responsável por descrever como a UI deve ser apresentada. O resultado deste método render é um elemento react, tratando-se de um simples 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 é a sintaxe utilizada pelo React, que significa *JavaScript Syntax Extension (JavaScript + XML)*. Embora não seja obrigatória a sua utilização na criação de aplicações React, esta extensão de sintaxe de JavaScript disponibiliza funcionalidades adicionais que permitem simplificar a leitura e escrita de código HTML dentro do JavaScript. 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 várias formas de criar e configurar uma aplicação React para converter o código JSX em Javascript, entre elas:

- → Deixar o **browser converter 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 JavaSript aquando 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, mas também torna o código difícil de manter.

- Configurar um **ambiente de desenvolvimento com o Node**, juntamente com um conjunto de ferramentas de desenvolvimento, simplificando a gestão das dependências, sendo esta uma das forma utilizadas no desenvolvimento de aplicações reais. Esta será a abordagem 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 de HTML e dos componentes devem ser sempre fechadas < />;
- Alguns atributos HTML mudam de nome, como por exemplo o atributo "**class**", que passa a ser "className" (pois class faz referência às classes em JavaScript). Além disso, o JSX usa a notação de camelcase para nomear atributos HTML.
- Não é possível, no método render, retornar mais de um elemento HTML de uma só vez, então, é necessário envolver todos os elementos em um elemento pai, como por exemplo, recorrendo a um elemento div, ou então, como alternativa entre *tags* vazias (designados *React Fragments* <> </>);
- JSX permite escrever expressões em {}, podendo esta ser qualquer expressão JS ou variável React.
- Para adicionar código JavaScript no meio do JSX, é necessário escrever entre {} , o qual apenas pode escrever uma expressão que avalia algum valor, como string, numero, array, um método map, entre outros. (expressões estas também conhecidas como *JSX Expression Syntax).*
- Quando um tipo de elemento inicia com letra minúscula, refere-se a um built-in componentes, por exemplo o 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. Podem ser implementados com recurso a uma **classe ou função**, e, portanto, designados como ***class components*** e ***functional components***, respetivamente. 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 mas sim de funções. A diferença clara entre estes dois tipos de componentes (*class* ou *functional*), é a sintaxe. Um componente funcional é apenas 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. No contexto das aulas práticas, serão essencialmente usados componentes funcionais.

Os nomes dos **componentes** em React **devem iniciar sempre com letra maiúscula,** como 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, com recurso a um componente funcional, o `FunctionalComponent`, e a um componente de classe, o `ClassComponent`. Qualquer um destes componentes, retorna um elemento h1, com o texto Linguagens Script.

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

Código 1 - Componente funcional sem estado

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

Código 2 - Componente Funcional anterior recorrendo a uma arrow function

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

Código 3 - Componente de Classe

Um conceito importante dos componentes é a forma como estes se comunicam. O React tem um objeto especial, chamado de **props** (que significa propriedade), que permite transportar dados de um componente para o outro. As props transportam dados apenas numa direção (do elemento pai para os elementos filhos) e, portanto, não é possível com as props passar dados do elemento filho para o pai, nem para componentes ao mesmo nível. 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")
);
```

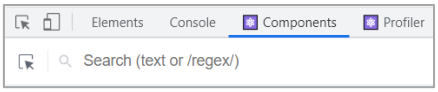
Algumas ligações úteis para esta parte introdutória ao React:

- 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)*

a. No **browser**, instale a extensão "***React Developer Tools***", que permite inspecionar e analisar os componentes React existentes na ágina, entre outras funcionalidades:

- "React Developer Tools" no *chrome*
- "React Developer Tools" no *firefox*



b. 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`

c. Efetue o download e descompacte o ficheiro **ficha7.zip** disponível no *inforestudante*.

d. 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 1 - Pagina Inicial

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. Análise do ficheiro **`index.html`**

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 *scritps* `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 em transformar o código JSX em JavaScript. O código abaixo 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 pode verificar, o corpo do documento index.html contém apenas uma div, única, sendo isso o que geralmente acontece 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, geralmente identificada como *root*, por convenção, como acontece neste exemplo, 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>
```

Por fim, é adicionada uma ligação ao ficheiro **`index.js`** que contém o código que será injetado na div **root** e o método **`render`** que irá renderizar o primeiro componente react.

Para além do código anteriormente descrito, encontram-se ainda alguns estilos CSS embebidos, de forma a explicar como estes podem ser utilizados numa simples aplicação 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 na página seguinte, será no elemento com id root.

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

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>
}
```

Parte II – Alterações e Estilização do Componente

2> Após analisar o código anterior, implemente as seguintes alterações ao código:

a. Transforme a função *LinguagensScript* numa *arrow function;*

b. Sem alterar a função, especifique código para que a aplicação fique com o aspeto da figura 2.

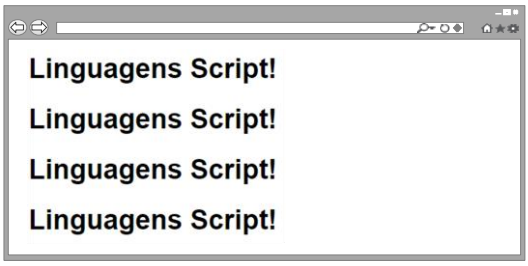


Figura 2 - Exercicio

c. Altere a função, por forma a que o componente receba um nome de forma a especificar boas vindas. Veja as dicas apresentadas na página 3. O componente deverá ser invocado, da seguinte forma, e ficar com o aspeto da figura 3.

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

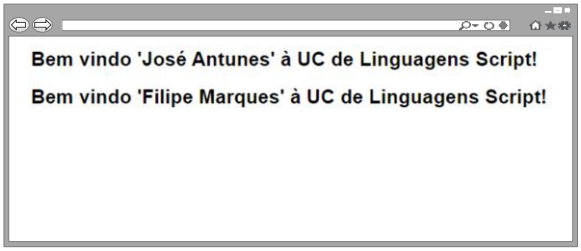


Figura 3 - Exercicio

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 se efetuado da forma como se apresenta nos seguintes exemplos.

Repare que algumas propriedades definidas diferenciam, de forma residual, das propriedades CSS, pois a propriedade text-align, como é constituída por duas palavras, necessita de uma certa transformação para serem usadas em React. Portanto, as propriedades CSS constituídas por palavras separadas por um hífen, tais como como *background-color*, *border-radius* ou o *font-family*, entre outras, são transformadas numa única palavra, no formato camelCase, isto é, o traço é removido e a primeira letra da segunda palavra passa a maiúscula. Por exemplo, propriedades como o background-color e border-radius passam a backgroundColor e borderRadius, respetivamente, em React.

```
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")
);
```

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 *camelCased* 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 automatica o sufixo "px" a certas propriedades numéricas de estilo embutido. Se se pretender usar unidades diferentes, o valor deve ser especificado como uma *string* com a unidade desejada.

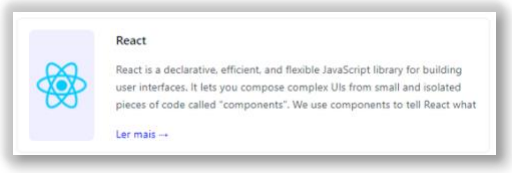
a. Para aplicar estilos no h1 com recurso ao atributo style directamente no h1, declare, no ficheiro index.js, o objeto **styleH1** . Este objeto deverá conter as seguintes propriedades:

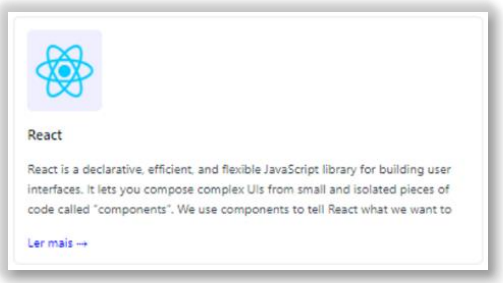
```
fontFamily: 'sans-serif',
textDecoration: 'underline',
color: 'brown'
```

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

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 da página seguinte. O componente é composto por um logotipo, um titulo, 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` e portanto, não será necessário especificar qualquer regra CSS.





a. Apresenta-se de seguida o código html que especifica um elemento.

- 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 logo encontra-se enquadrado num div com determinadas propriedades
- O texto é composto pelo título, a descrição e pelo link.

```
<div class="container">
    <div class="wrapper">
       <div class="logo">
            <img src="images/react.png" alt="React" />
        </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.

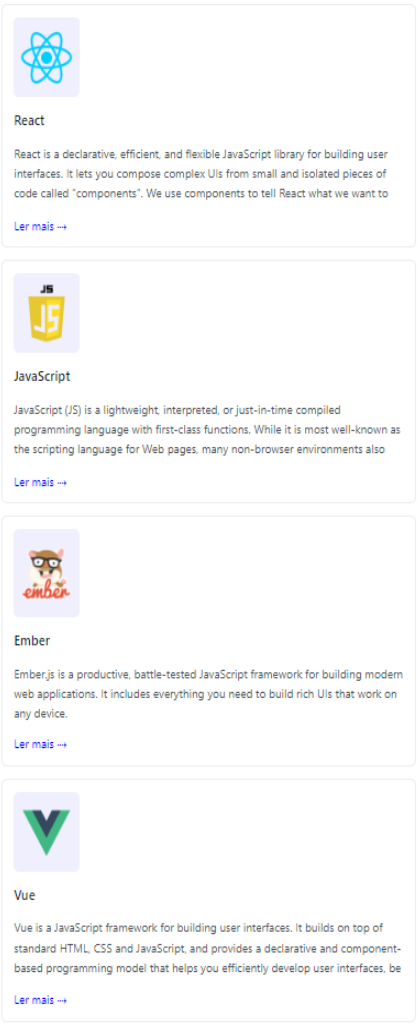


Figura 5 – Layout para Viewport inferior a 640px

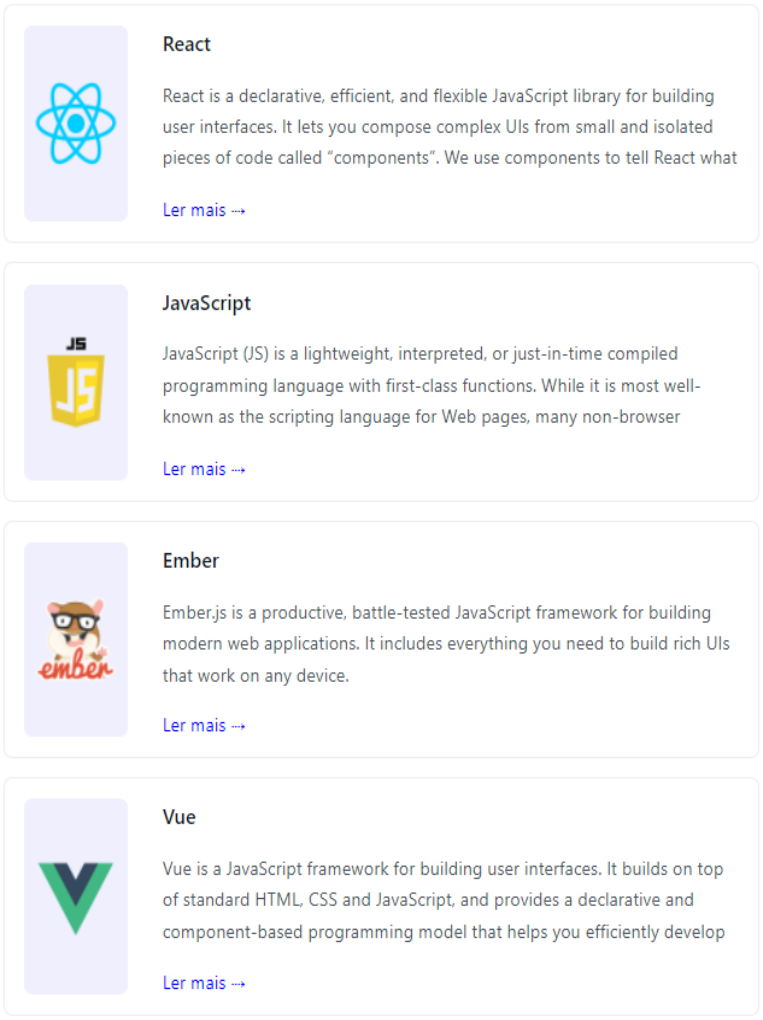


Figura 5 – Layout a partir de Viewport com largura de 640px

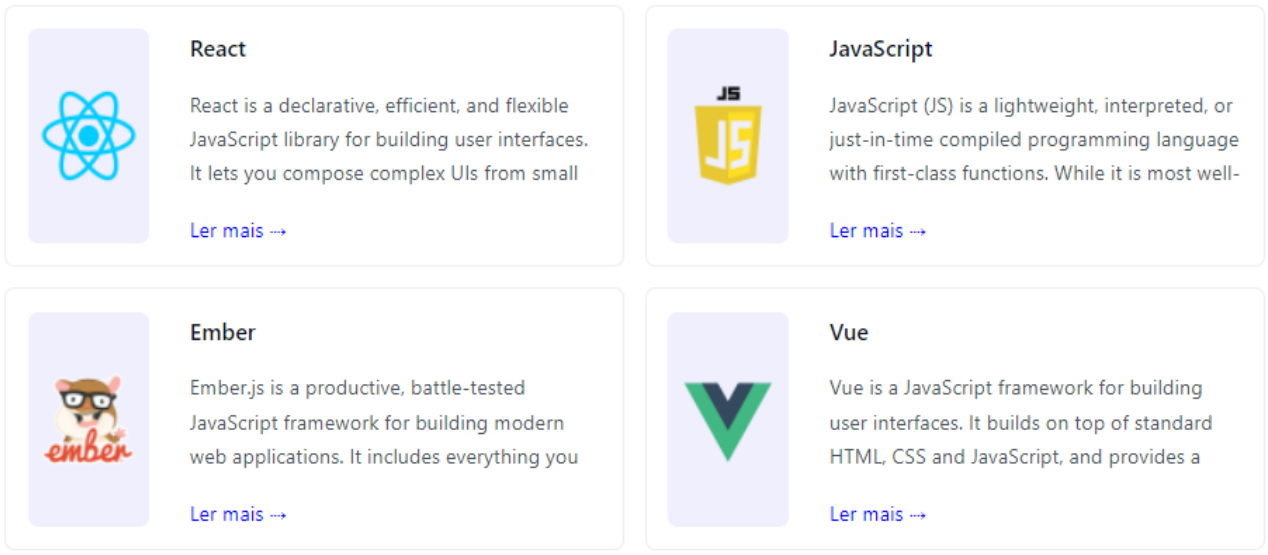


Figura 6 – Layout a partir de Viewport com largura de 768px

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 contant 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.

› *Criação de Projecto em React (Para explorar em casa)*

Criação de um projeto com o Create React App.

Para facilitar a criação de aplicações React, o Facebook lançou uma ferramenta que permite reduzir a complexidade envolvida na configuração de um projeto React, para quem esteja a dar os primeiros passos com esta tecnologia. A ferramenta é designada como "Create React App" (https://create-react-app.dev/) e recorre-se apenas a um comando para a criação do projeto. Está pré-configurada, permitindo gerar facilmente a estrutura básica de diretórios e toda a configuração de *build* e dependências necessárias, permitindo a abstração de um conjunto de comandos complexos do Babel para compilação do código e do *Webpack* que permite empacotar os ficheiros, elementos usados na maioria dos projetos, facilitando, desse modo, a ecriação da estrutura inicial e implementação de uma aplicação React.

a. O primeiro passo é confirmar se o **Node** já se encontra instalado na máquina:

- Na linha de comando, ou então no terminal do *visual studio code*, escreva o comando "**node -v**". Caso seja apresentada a versão instalada, verifique se tem, pelo menos, a versão 16, e caso não tenha, desinstale e execute o passo seguinte.

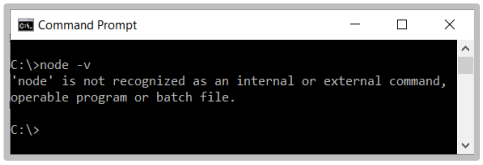


Figura 7 - Node não se encontra instalado

- Caso não esteja instalado, instale o node.js (https://nodejs.dev/download), selecionando a opção adequada ao SO. Depois de instalar, volte a confirmar se o mesmo se encontra instalado, executando o comando anteriormente referido.

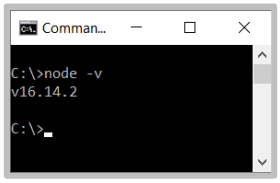


Figura 8 - node instalado

- **Nota:** Apesar de não ser utilizado o node, a sua instalação é necessária pois será usada uma das suas ferramentas "built-in", como o Node Package Manager (NPM) que permite a instalação de bibliotecas e outros elementos de terceiros.

b. Para criar a estrutura da aplicação, execute os seguintes comandos na linha de comando ou no terminal do *VSCode*:

Figura 9 - create-react-app

- **`npx create-react-app memory-game-react`**
  Neste momento, o projeto com toda a estrutura base de pastas e ficheiros foram criados.

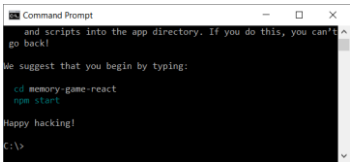


Figura 10 - Projeto criado

- Execute a aplicação, através dos comandos:
  - **`cd memory-game-react`**
  - **`npm start`**
    Este ultimo irá abrir a aplicação no browser no endereço http://localhost:3000/. Caso seja solicitada

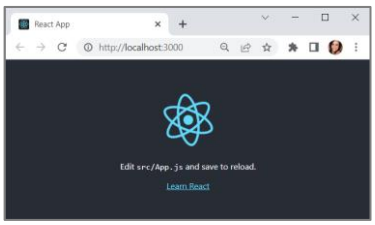


Figura 11 - Pagina create-react-app

alguma autorização devido a *firewalls*, esta deve ser aceite de forma a poder visualizar a página no browser. A página terá o aspeto da figura 6.

→ **Nota:** Os passos anteriores poderiam ser substituídos recorrendo ao site https://createapp.dev/ que permite escolher o tipo de aplicação pretendida (react neste caso) e criar uma aplicação vazia mas a estrutura de pastas e ficheiros necessários.

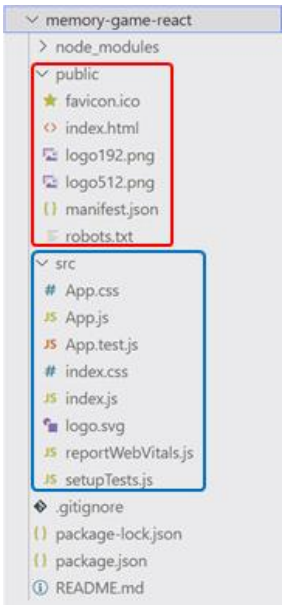


Figura 12 - Estrutura de ficheiros/directorios - Aplicação React

c. Abra o projeto `memory-game-react` no *VSCode* de forma analisar a estrutura de diretórios e ficheiros existentes.

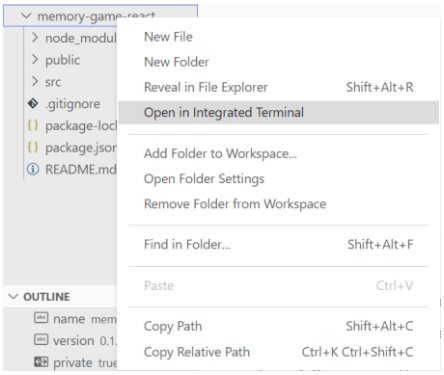
→ A estrutura obtida deverá ser a que se apresenta na Figura 7.

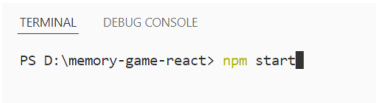
→ No ficheiro `package.json`, verifique que existe uma dependência chamada react-scripts e três scripts:

- **`start:`** `react-scripts start`
- **`build:`** `react-scripts build`
- **`eject:`** `react-scripts eject`

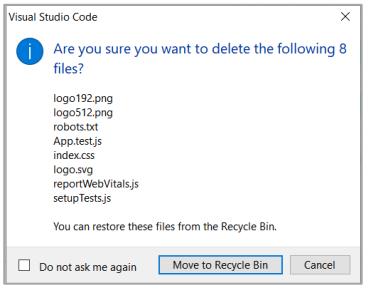
Logo, o comando anteriormente especificado **npm start** executa o script *start* que permite iniciar a aplicação tendo como base os componentes que estão na pasta src/, nomeadamente o ficheiro **index.html**

→ Esse comando, pode ser especificado diretamente no *visual studio code*. Assim, abra o terminal nessa pasta, clicando com o botão direito na pasta do projecto, e selecionado "*Open Integrated Terminal*" e inicie a aplicação diretamente pelo terminal.





d. Como existem alguns ficheiros que **não serão utilizados** no contexto do projeto a implementar, execute os seguintes passos:



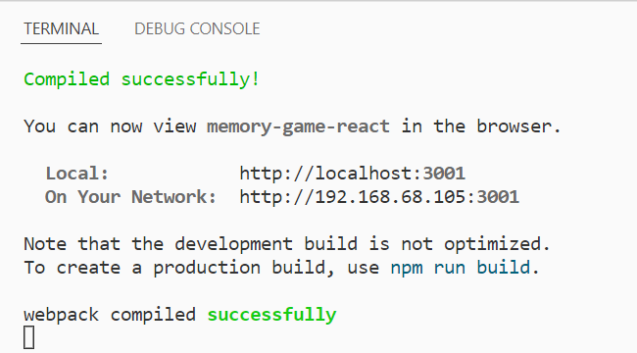
→ **Selecione os 8 ficheiros apresentados na figura e remova-os.**

→ Como existem referencias a esses ficheiros no código implementado, remova-as também, nomeadamente:

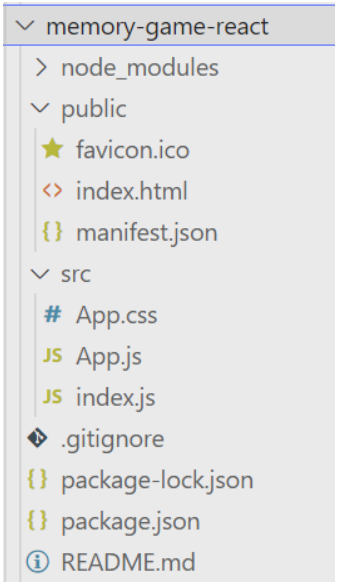
- No ficheiro `/src/App.js` remova as seguintes linhas de código
  ```
  import logo from './logo.svg';
  <img src={logo} className="App-logo" alt="logo" />
  ```
- No ficheiro `/src/index.js` remova o seguinte código:
  ```
  import './index.css';
  ```

```
import reportWebVitals from './reportWebVitals';
reportWebVitals();  // Encontra-se no fim do ficheiro
```

→ De forma a verificar se ficou tudo bem, execute a aplicação, através do comando **npm start** e verifique se a compilação teve sucesso, como apresentado abaixo. Poderá demorar alguns segundos, aguarde.



→ Depois dos passos anteriores, a estrutura final deverá ser a seguinte:



→ De forma a eliminar umas dependências desnecessárias execute o seguinte comando:
npm uninstall @testing-library/jest-dom @testing-library/react @testing-library/user-event web-vitals

- Este passo pode ser substituído, eliminando a pasta "node_modules" e o ficheiro *package-lock.json* e posteriormente executar o comando **npm install**

→ Os ficheiros que deve dar mais atenção na fase inicial, é o index.html (dentro da pasta public), index.js e App.js.

- A partir do momento que a aplicação estiver a correr no browser, **sempre que for efetuada qualquer alteração de um dos ficheiros, eles serão recompilados automaticamente e o browser também será atualizado.** Se existir algum erro durante a compilação, estes são apresentados na consola.

- Abaixo encontra-se uma explicação genérica da estrutura e ficheiros que compõem o projeto:
  - **node_modules** esta pasta é onde o npm armazena localmente todos os pacotes instalados. Não se precisa efetuar alterações a esta pasta.
  - **index.html** ficheiro principal o qual é apresentado no browser. Inclui todos os *metadados* que descrevem o conteúdo da página.
  - **manifest.json** Descreve toda a aplicação incluindo a informação necessária para os icons, cores entre outros.
  - **.gitignore** ficheiro que descreve quais diretorias serão ignoradas, no contexto do controlo de versões *git*, como por exemplo, o diretório *node_modules* será ignorado. Não é obrigatório usar o *git* no contexto das aulas, mas não será removido para quem desejar explorar o seu uso.
  - **package.json** inclui todos os pacotes que o projeto depende, e qual a versão que o projeto pode usar. Inclui ainda um conjunto de script do *create_react_app*, que permitem, por exemplo, executar a aplicação com npm start.
  - **package-lock.json** inclui a versão exata os pacotes utilizados no projeto.
  - **README.md** inclui informação geral do projeto, a descrição, como instalar e executar.
  - **Pasta src** inclui o código fonte. Aqui pode-se incluir todos os componentes em react, módulos, ficheiros de estilização, entre outros.
  - **index.js** ponto de entrada para a aplicação React. O método render é invocado para renderizar um elemento React no DOM, e retorna a referencia do componente. Neste caso, o elemento React é o componente App.
  - **App.js** é o componente principal (root) da aplicação, o qual irá importar outros componentes, filhos.
  - **App.css** contém estilos CSS que estão a ser usados no ficheiro App.js.

e. **Nota final:** Embora não seja necessário para aulas práticas de Linguagens Script, é possível, em qualquer momento, remover permanentemente a configuração padrão do **Create React App**, por exemplo, para criar uma versão para produção. Para isso é necessário executar o comando *eject* e, nessa sequência, será criado um diretório config/ com todas as configurações utilizadas por padrão. Portanto, o ficheiro package.json será modificado para obter as dependências do Babel, Webpack e afins.

f. Comandos uteis:

- → **npx depcheck** verifica as dependências instaladas que não estão a ser utilizadas
- → **npm uninstall package** permite remover uma dependência especifica.