> *Ficha Prática Nº8 (Jogo de Memória em React)*

Pretende-se, durante as próximas aulas práticas, implementar o jogo de memória, recorrendo à tecnologia **React.** O jogo será composto por vários níveis, que especificam o número de cartas a serem distribuídas num tabuleiro, tempo limite para um jogo e cálculo da pontuação. Os estilos (CSS) necessários à implementação do jogo já são disponibilizados, de forma a que o aluno foque a sua atenção na criação dos vários componentes React. O aspeto do jogo encontra-se apresentado nas imagens seguintes.

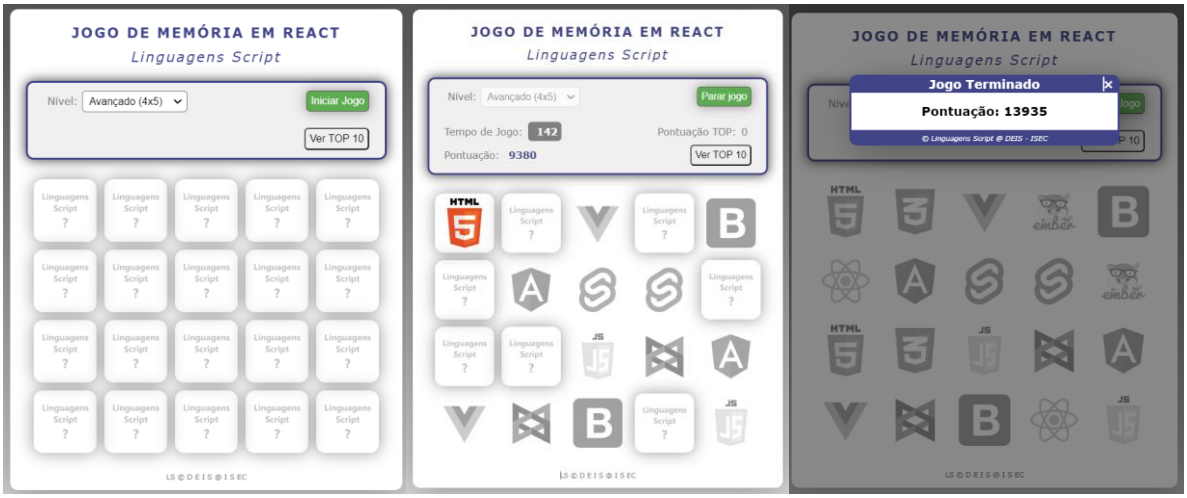


Figura 1 - Jogo de Memória em React

> *Preparação do ambiente*

Como referido na aula introdutória ao React, existem diferentes abordagens para criar e configurar uma aplicação React. Nesta ficha, será adotada uma abordagem usada em aplicações reais, em que é necessário configurar um ambiente de desenvolvimento com o Node, que juntamente com ferramentas de desenvolvimento, simplificará a gestão de dependências. Para isso, efetue os seguintes passos:

a. Confirme a instalação e versão do Node. Para isso:

- → 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 18, e caso seja uma versão inferior, desinstale e execute o passo seguinte.

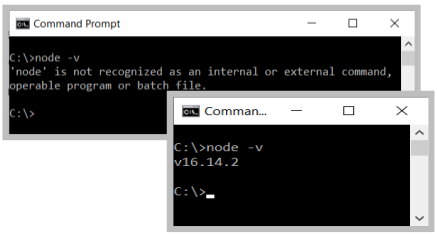


Figura 2 - Confirmação da versão do Node

- → Instale o node.js (https://nodejs.dev/download), selecionando a opção adequada ao SO. **Nota:** Apesar de não ser utilizado o node de forma direta, 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. Inicie o *Visual Studio Code* e abra a **pasta da ficha 8 no workspace**.

c. Com o botão direito do rato num dos ficheiros/sub-pasta, selecione a opção "***Open Integrated Terminal***" , ou então, diretamente na linha de comando posicione-se na pasta correspondente, e execute os seguintes comandos no terminal:

- → **`npm install`** (para instalar as dependências em falta)
- → **`npm start`** (para iniciar a aplicação)

d. Confirme a aplicação no browsers, no dentreço http://localhost:3000/

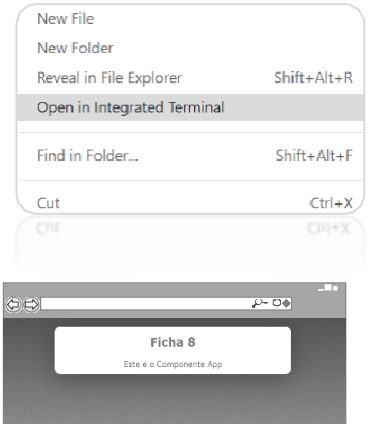


Figura 3 - Estado inicial da aplicação

Parte I – Estrutura da Aplicação e Organização do Código

O "`create-react-app`" é uma ferramenta que permite criar a base de uma aplicação react, reduzindo assim a complexidade envolvida na configuração de um projeto a implementar com esta tecnologia (ver anexo). Assim, o projeto fornecido na ficha, foi criado com recurso a esta ferramenta, existindo, no entanto, algumas pastas e ficheiros que não fazem parte da estrutura base do "`create-react-app`", tendo estas sido criadas para uma melhor organização e estruturação do código a implementar. Para além disso, já se encontram especificados todos os ficheiros de estilos necessários à implementação da aplicação.

```
public
    favicon.ico
    index.html
    manifest.json
    _robots.txt
assets
    images
        angular.png
        backbone.png
        bootstrap.png
        css.png
        ember.png
        html.png
        javascript.png
        ls.png
        react.png
        svelte.png
        vue.png
src
    App.js
    index.js
assets
    styles
        App.css
        index.css
components
    card
        card.css
    control-panel
        control-panel.css
    footer
    game-over-modal
        game-over-modal.component.jsx
        game-over-modal.css
    game-panel
        game-panel.css
    top-10-modal
        top-10-modal.component.jsx
        top-10-modal.css
    header
constants
    index.js
helpers
    checkIfIsFlipped.js
    checkIfIsMatched.js
    index.js
    shuffle.js
```

› *Diretórios*

Segue uma breve descrição dos ficheiros, pastas e subpastas principais.

- → A pasta **src** contém várias subpastas:
  - ***assets***: onde estão especificados os estilos globais;
  - ***components***: local onde devem serão especificados todos os componentes *react*. Repare que existe uma pasta para cada componente com o respetivo ficheiro de estilos CSS;
  - ***constants****:* local onde se encontram concentradas várias variáveis contantes a serem utilizadas na aplicação;
  - ***helpers****:* inclui várias funções *javascript*, independentes, podendo ser reutilizadas em diferentes contextos.
- → A pasta **public** é o local onde se encontram ficheiros estáticos, tais como, imagens, index.html e outros ativos, etc.. Apenas ficheiros dentro da pasta "*public*" podem ser referenciados no ficheiro HTML.

Organização dos Componentes

As aplicações React baseiam-se na implementação de diversos componentes reutilizáveis e independentes. Este tipo de arquitetura permite a divisão da aplicação em diversas partes (componentes), que juntas e interligadas permitem a criação de uma UI complexa. A Figura 4 apresenta **uma possível abordagem** para divisão da aplicação, onde se destacam os diversos componentes a serem implementados ao longo das aulas práticas, de forma a completar o Jogo de Memória. Assim, destacam-se os seguintes componentes principais, como se pode ver na figura:

- **Header:** corresponde ao componente que contém o cabeçalho como os títulos;
- **ControlPanel:** componente que conterá a caixa de seleção do nível, os botões e outros elementos;
- **GamePanel:** componente que engloba o conjunto total das cartas;
- **Card**: componente que define uma carta;
- **Footer:** componente que define a área do em rodapé, mais propriamente o simples texto.

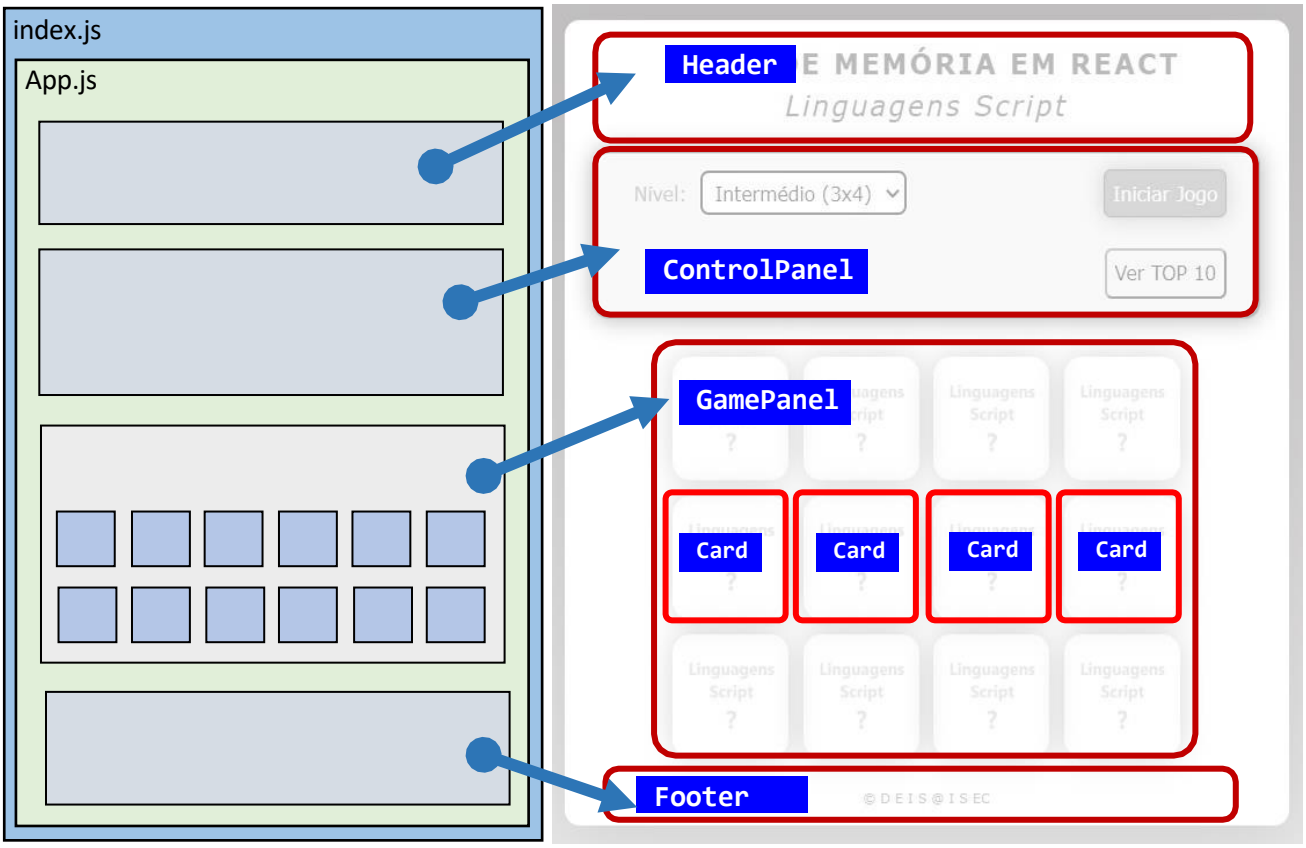


Figura 4 - Divisão aplicação em Componentes

Para além dos componente anteriormente apresentados, existem, ainda, dois componentes, já implementados, que se referem às janelas modais para fim do jogo, referente ao componente **GameOverModal**, e para apresentação do top, o componente **TopTenModal**.

> Organização por Módulos

De forma a organizar o código da aplicação e o mesmo não estar todo concentrado num só ficheiro, a aplicação deve ser dividida em vários ficheiros (como apresentado já na secção estrutura dos diretórios), tornando assim a aplicação **modular** e de mais fácil manutenção. Este conceito não é novo, mas apenas foi introduzido no *JavaScript* na versão ES6, com a introdução de **módulos em JS**. Assim, para modularizar a aplicação React é recomendável especificar um componente por ficheiro e especificar os estilos de cada componente num ficheiro CSS independente. Tenha em atenção os seguintes pontos:

- Quando se recorre a módulos para organização do código, os objetos definidos num **modulo** são privados, por omissão. Logo, o componente especificado dentro de um ficheiro, não é visível em outro ficheiro ou aplicação. Portanto, para o tornar público, isto é, visível fora desse local, é necessário fazer o **export** do componente. Nos ficheiros onde esses componentes serão usados, é necessário efetuar o **import** especificando o nome do ficheiro.

- **Resumindo**, a importação permite usar o conteúdo de outro ficheiro, enquanto que a exportação torna o conteúdo do ficheiro elegível para importação. Dessa forma, é possível trocar código entre vários ficheiros.

- É possível adicionar a palavra *default* num **export**, especificando o elemento default na exportação, num determinado módulo. Esta palavra-chave permite simplificar a forma como se especifica o import, e portanto, evita o especificar {}, no qual são usados quando se pretende especificar *Named Exports.*

- Abaixo apresenta-se o ficheiro index.js com um conjunto de imports, e o ficheiro App.js com export. Explore a aplicação.

```
import React from 'react';
import ReactDOM from 'react-dom/client';

import App from './App';
import "./assets/styles/index.css";
import "./assets/styles/w3.css";

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Figura 5 - Import no ficheiro index.js

```jsx
import "./assets/styles/App.css";

function App() { return (
    <div id="container">
      <h2>Jogo de Memória em React</h2>
      <h3>Linguagens Script</h3>
    </div>
  );
}

export default App;
// Esta linha também poderia ser eliminada e a definição da função ser
// export default function App() {
```

Figura 6 -export default

Durante a resolução da ficha, será ainda apresentada uma forma de concentrar todos os *export* num ficheiro, facilitando assim a quantidade de *imports* dos componentes.

Parte II – Implementação de Componentes

Pretende-se que o aluno implemente a estrutura básica da aplicação, mais propriamente a UI da aplicação, como apresentada na figura 7, devendo para isso implementar vários componentes, nomeadamente o `Header`, `Footer`, `ControlPanel`, `GamePanel` e `Card`.

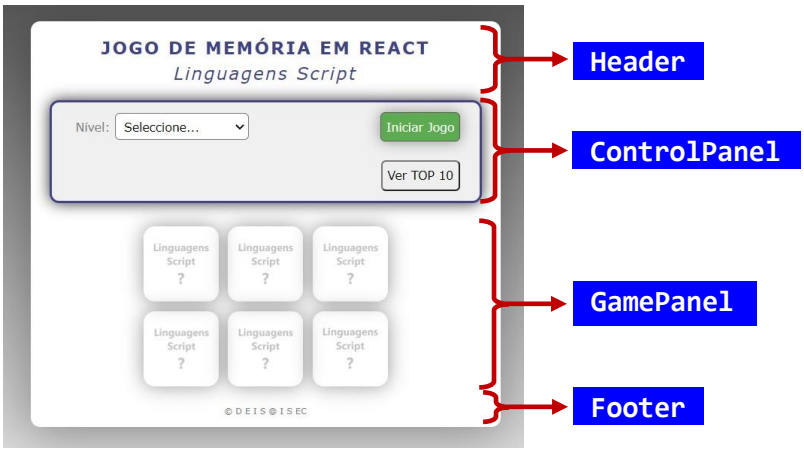


Figura 7 - Ficha 8

1> Implemente o componente `Header`. Considere o seguinte:

- → O ficheiro para o componente deve ser **`header.component.jsx`** e deve ser colocado **dentro da pasta correspondente - header**.
- → Especifique *import* do React, com a linha de código `import React from "react";`
- → Efetue o *export defoult* do componente de forma a ser acessível no ficheiro do componente App.

→ O código HTML que especifica o **header pretendido** encontra-se no trecho de código abaixo. Copie e efetue as alterações necessárias para que o componente renderize corretamente o pretendido. Não esqueça que o atributo **class** deverá ser substituído por **className**

```
<header>
      <h1 class="title">Jogo de Memória em React</h1>
      <h2 class="subtitle">Linguagens Script</h2>
</header>
```

Figura 8 – HTML do Header e Footer

→ No **componente App**, remova o código `<h2>Ficha 8</h2>` , e invoque o Header, **não esquecendo de efetuar o import.**

→ Visualize, no browser, os componentes implementados que devem ter ficado com o aspeto da figura seguinte. Verifique ainda a inexistência de erros no terminal / browser ou qualquer erro que possa estar a ser apresentado na consola.

Figura 9 – Componente Header e Footer

2> Implemente o componente Footer. Considere o seguinte:

→ O ficheiro para o componente deve ser **`Footer.component.jsx`** e deve ser colocado **dentro da pasta correspondente - footer**.

→ Especifique *import* do React, com a linha de código `import React from "react";`

→ Efetue o *export defoult* do componente de forma a ser acessível no ficheiro do componente App.

→ O código HTML do **footer** encontra-se no seguinte trecho de código. Não se esqueça que o atributo **class** deverá ser substituído por **className**

```
<footer>
   <p>LS © D E I S @ I S EC</p>
</footer>
```

Figura 10 – HTML do Header e Footer

→ No **componente App**, remova o texto "Este é o Componente App" pela invocação do Footer, **não esquecendo de efetuar o *import*.**

→ Visualize, no browser, os componentes Header e Footer.

3> Implemente o componente `ControlPanel`. O componente não ficará funcional, apenas a interface deverá ficar com o resultado apresentado na Figura 11.

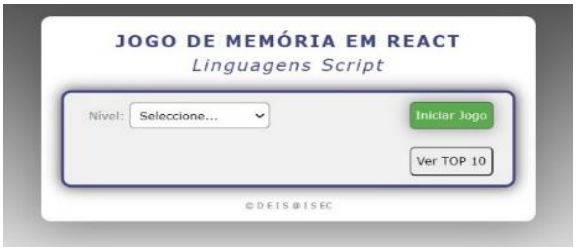


Figura 11 – Componentes Header, ControlPanel, Footer

Tenha em consideração:

- Nome de ficheiro deverá ser **`control-panel.component.jsx`**
- Especifique o *import* do React e do ficheiro de estilos: `import "./ control -panel.css";`
- Especifique o *export* do componente.
- A estrutura HTML para o componente encontra-se de seguida. Não se esqueça de converter os atributos HTML, para JSX, tais como **class** → **className** e **for** → **htmlFor**.

```
<section id="panel-control">
  <h3 class="sr-only">Escolha do Nível</h3>
  <form class="form">
    <fieldset class="form-group">
      <label for="btLevel">Nível:</label>
      <select id="btLevel">
        <option value="0">Seleccione...</option>
        <option value="1">Básico (2x3)</option>
        <option value="2">Intermédio (3x4)</option>
        <option value="3">Avançado (4x5)</option>
      </select>
    </fieldset>
    <button type="button" id="btPlay">Iniciar Jogo</button>
  </form>
  <div class="form-metadata">
     <p id="message" role="alert" class="hide" >
       Clique em Iniciar o Jogo!
     </p>
     <dl class="list-item left">
        <dt>Tempo de Jogo:</dt><dd id="gameTime">0s</dd>
     </dl>
     <dl class="list-item right">
        <dt>Pontuação TOP:</dt><dd id="pointsTop">0</dd>
     </dl>
     <dl class="list-item left">
        <dt>Pontuação:</dt><dd id="points">0</dd>
     </dl>
     <div id="top10" class="right">
        <button id="btTop">Ver TOP 10</button>
     </div>
  </div>
</section>
```

Figura 12 - HTML para componente ControlPanel

- No **componente App**, invoque o componente `PanelControl` como se apresenta no código seguinte. Não se esqueça de efetuar o *import*.

```
function App() {
  return (
    <div id="container">
      <Header />
      <main>
           <ControlPanel />
           { /* <GamePanel /> */}
      </main>
      <Footer />
    </div>
  );
}
```

- Confirme o resultado da Figura 11 no browser

4> Implemente o componente `GamePanel`, sendo que nesta fase, não apresenta as peças que compõem o tabuleiro de jogo.

Tenha em consideração:

- Nome de ficheiro deverá ser **`game-panel.component.jsx`**
- Especifique o *import* do React e do ficheiro de estilos: `import "./game-panel.css";`
- Especifique o *export* do componente.
- A estrutura HTML para o componente encontra-se no trecho abaixo. Efetue as alterações necessárias aos atributos.

```
<section id="panel-game">
      <h3 class="sr-only">Peças do Jogo</h3>
      <div id="game">
           Tabuleiro com Cartas
     </div>
</section>
```

- No **componente App**, invoque o componente, removendo o comentário e não esquecendo de efetuar o *import*.
- Confirme o resultado no browser

Figura 13 – GamePanel sem cartas

5> Implemente o componente `Card`, devendo o nome do ficheiro ser **`card.component.jsx`**.

a. Para implementação deste componente especifique a propriedade **name,** de forma a que a invocação seja efetuada da seguinte forma.

```
<Card name="angular" />
<Card name="html" />
<Card name="javascript" />
...
```

→ Tendo em consideração o conjunto de cartas implementadas no HTML, a estrutura seria, por exemplo, a que se apresenta na página seguinte.

```
<div class="card" data-logo="angular">
       <img src="images/ls.png" class="card-back" alt="card placeholder" />
       <img src="images/angular.png" class="card-front" alt="card front" />
</div>
<div class="card" data-logo="bootstrap">
       <img src="images/ls.png" class="card-back" alt="card placeholder"/>
       <img src="images/bootstrap.png" class="card-front" alt="card front" />
</div>
<div class="card" data-logo="html">
       <img src="images/ls.png" class="card-back" alt="card placeholder"/>
       <img src="images/html.png" class="card-front" alt="card front" />
</div>
<div class="card" data-logo="javascript">
       <img src="images/ls.png" class="card-back" alt="card placeholder"/>
       <img src="images/javascript.png" class="card-front" alt="card front" />
</div>
<div class="card" data-logo="vue">
       <img src="images/ls.png" class="card-back" alt="card placeholder" />
       <img src="images/vue.png" class="card-front" alt="card front" />
</div>
...
```

Figura 14 - HTML para Cartas

→ Além disso, repare que, no ficheiro index.js da pasta **constants**, existem duas contantes que deverão ser usadas na criação deste componente, a `PLACEHOLDER_CARD_PATH` e `PLACEHOLDER_CARDBACK_PATH` que especificam o caminho onde se encontram os ficheiros das imagens, seja dos logótipos como do ficheiro ls.png, respetivamente.

- Invoque o componente `Card` no componente `GamePanel`, tantas vezes quantas necessárias para apresentar seis cartas com o os logotipos acima apresentados.
- Visualize a aplicação no browser, que deverá ter o aspeto apresentado na figura 15.
- Por fim, para que os logótipos fiquem visíveis, aplique a classe `flipped` à **Card** por forma a que as cartas fiquem viradas como se mostra na figura 16.

Figura 15 - Ficha 8

Figura 16 - Ficha 8 Concluída

b. Analise o código do componente `GamePanel`. Como pode verificar, o componente `Card` é invocado várias vezes, variando apenas, em cada invocação, o valor do atributo *name*. Em vez de especificar várias invocações deste modo, reduza o código, implementando o pretendido através de um método ou estrutura de controlo, considerando que o valor do atributo *name* se encontra num array de strings. Desse modo:

- Considere o array de string **CARD_LOGOS** que se encontra declarado no ficheiro existente na **pasta constants**, correspondente aos logotipos;
- Substitua a invocação das várias instancias do componente card, pelo uso de uma estrutura de controlo ou um método, considerando que o atributo name deve ser preenchido com os valores existentes no array.
- Altere o código de forma a apresentar apenas 6 cartas (*slice*)

Parte III – Simplificação dos imports

6> De forma a simplificar o *import* entre os vários componentes, implemente os seguintes passos:

a. Crie o ficheiro **index.js** na pasta **components**.

b. Copie a linha de código abaixo e, além disso, adapte este export para os restantes componentes implementados.

```
export { default as GameOverModal } from "./game-over-modal/game-over-modal.component";
```

c. No ficheiro **App.js** altere o modo como efetua os imports, devendo agora ser do seguinte modo:

```
import { ControlPanel, Footer, Header,GamePanel} from "./components";
```

d. Por fim, para importar um determinado componente no **GamePanel**, o qual será necessário importar o componente Card, o caminho ficará simplificado da seguinte forma:

```
import { Card } from "../index";
```

› Anexo: Criação de Projecto em React com 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.

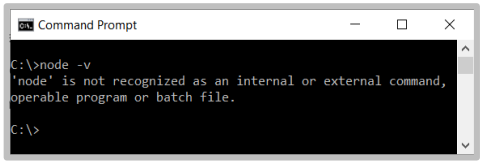


Figura 17 - Node não se encontra instalado

- 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.

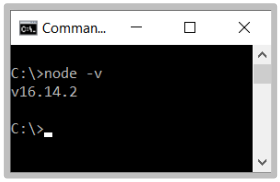


Figura 18 - node instalado

- **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*:

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

Figura 19 - create-react-app

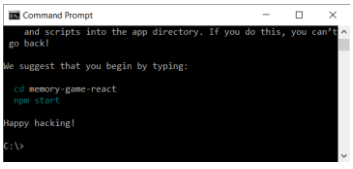


Figura 20 - Projeto criado

- 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 alguma autorização devido a *firewalls*, esta deve ser aceite de forma a poder visualizar a página no browser.

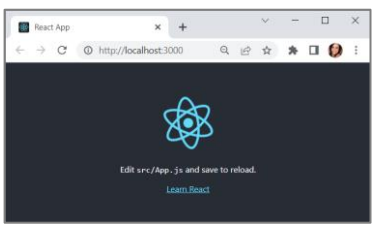


Figura 21 - Pagina create-react-app

→ **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.

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**

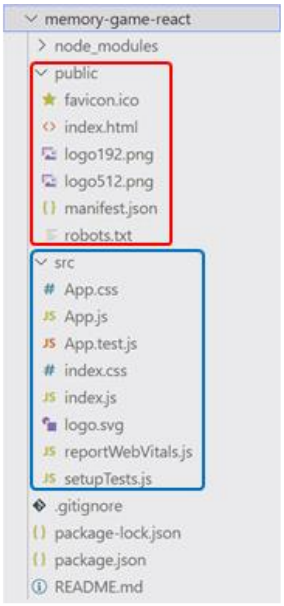
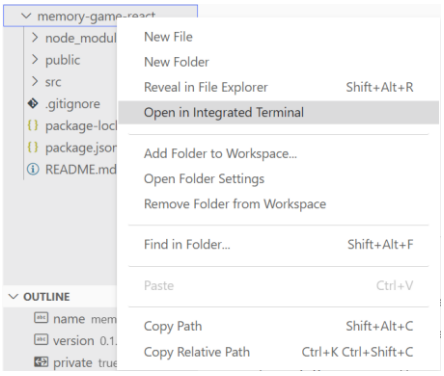
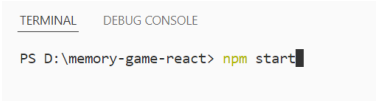


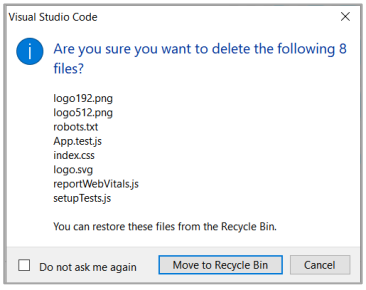
Figura 22 - Estrutura de ficheiros/directorios - Aplicação React

→ 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.**

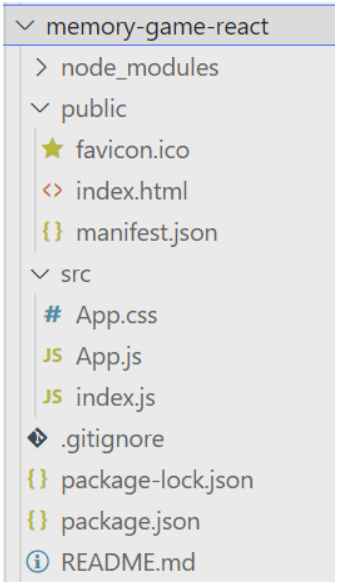
→ Como existem referencias a esses ficheiros no código implementado, remova-as também, nomeadamente:

- 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.

```
TERMINAL    DEBUG CONSOLE

Compiled successfully!

You can now view memory-game-react in the browser.

  Local:            http://localhost:3001
  On Your Network:  http://192.168.68.105:3001

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

webpack compiled successfully
```

→ 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.