

> Ficha Prática Nº8 (React – Jogo de Memória – Parte I)

Pretende-se implementar o jogo de memória em **React**, no qual é composto por vários níveis, limite de tempo para cada jogada e pontuação. A implementação do jogo será realizada ao longo das próximas aulas práticas. Todos os estilos (CSS) necessários à implementação, serão fornecidos, de forma a que o aluno foque a sua atenção na implementação dos vários componentes React. O aspeto do jogo encontra-se apresentado nas imagens seguintes.



Figura 1 - Jogo de Memória - Versão React

> Preparação do ambiente

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.

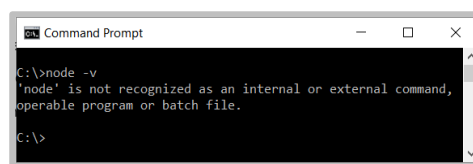


Figura 2 – Verificação instalação do node

- 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.

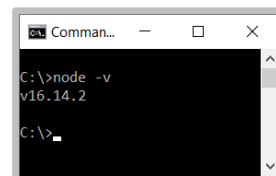


Figura 3 - Versão do node

- **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. Efetue o download e descompacte o ficheiro **ficha8.zip** disponível no *inforestudante*.

c. Inicie o *Visual Studio Code* e abra a **pasta da ficha 8 no workspace**.

d. Com o botão direito do rato um dos ficheiros/sub-pasta, selecione a opção "*Open Integrated Terminal*", ou então, na linha de comando do Windows posicione-se na pasta correspondente;

e. No terminal, digite os seguintes comandos:

→ **npm install** (para instalar as dependências em falta)

→ **npm start** (para iniciar a aplicação)

f. Confirme a aplicação no browser, endereço <http://localhost:3000/>.

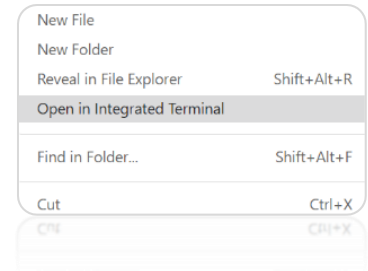


Figura 4 – Estado Inicial da Aplicação

Parte II – Estruturação da Aplicação

> Estrutura de Diretórios

Como pode verificar no projeto fornecido, a estrutura de diretórios já se encontra especificada, existindo algumas pastas e ficheiros que não fazem parte da estrutura base do “create-react-app”, tendo estas sido criadas para organização e estruturação do código. Para além disso, já se encontram especificados todos os ficheiros de estilos necessários à implementação da aplicação. Segue uma breve descrição dos ficheiros, pastas e subpastas principais.

→ A pasta **src** contém várias subpastas:

- **assets**: onde estão especificados os estilos globais;
- **componentes**: local onde devem ser 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 constantes 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.

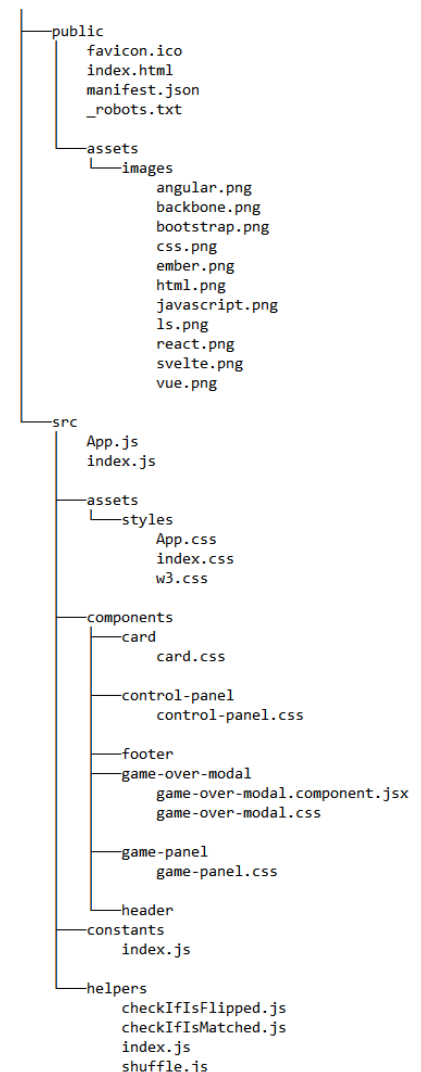


Figura 5 - Estrutura de Directórios

> Organização dos Componentes

Como referido na aula introdutória ao React, 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 mais complexa. A figura seguinte 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 seguinte:

- **Header:** corresponde ao componente que contém o cabeçalho como os títulos;
- **ControlPanel:** componente que conterà a caixa de selecção 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.

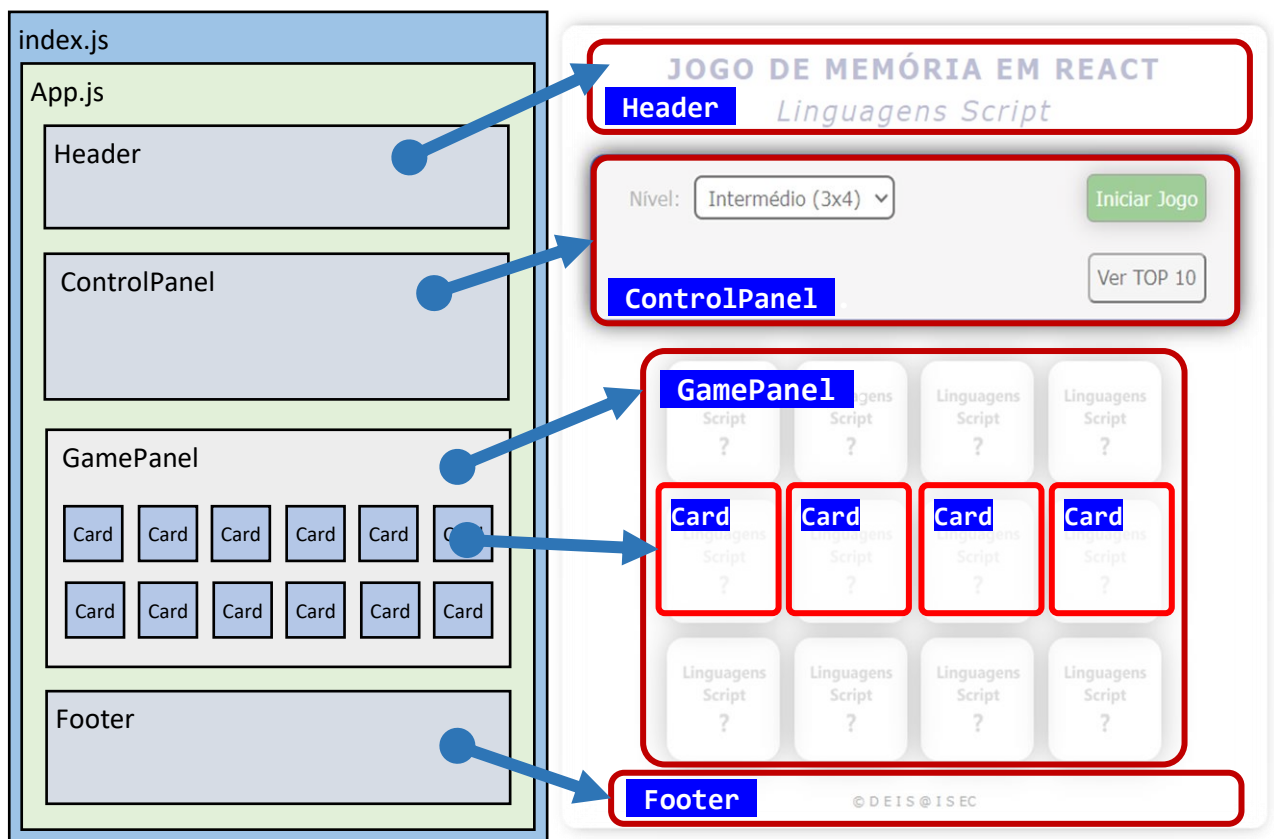


Figura 6 - Divisão aplicação em Componentes

> 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 deve especificar um componente por ficheiro e especificar os estilos de cada componente num ficheiro CSS independente. Tenha em atenção os seguintes pontos:

- Quando se trabalha com módulos, os objetos definidos num **módulo** 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>
);
```

Código 1 - Import no ficheiro index.js

```
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 substituída
// export default function App() {
```

Código 2 - export default

- Na secção seguinte, será depois apresentada uma outra forma de concentrar todos os *export* num ficheiro, facilitando assim o *import* de todos os componentes.

Parte II – Exercício: Implementação de Componentes

- 1>** Pretende-se que o aluno implemente a estrutura básica da aplicação, mais propriamente a UI da aplicação, devendo para isso criar os vários componentes.
- a.** Tendo em consideração o código HTML que se apresenta abaixo, que especifica a estrutura do cabeçalho e rodapé da aplicação, implemente o componente **Header** e o componente **Footer**, cujos nome dos ficheiros deverão ser **header.component.jsx** e **footer.component.jsx**, respetivamente.

Notas:

- Os componentes devem ser implementados dentro das pastas e sub-pastas correspondentes. Assim, crie o ficheiro com o nome dos componentes referidos anteriormente, no local correcto.
- Em cada componente, não se esqueça de especificar `import React from "react";` bem como efetuar o *export* do componente para poder ser acessível na App.
- O HTML do **head** e **footer** encontra-se no seguinte trecho de código.

```

<div id="container">
  <header>
    <h1 class="title">Jogo de Memória em React</h1>
    <h2 class="subtitle">Linguagens Script</h2>
  </header>
  ...
  <footer>
    <p>@ D E I S @ I S E C</p>
  </footer>
</div>

```

- Não se esqueça que o atributo **class** deverá ser substituído por **className**
- Tendo em consideração o HTML acima, invoque os componentes anteriormente implementados **no componente App**, não esquecendo de efetuar o import.
- Visualize se existem erros no terminal / no browser e verifique qualquer erro que possa ser apresentado na consola.

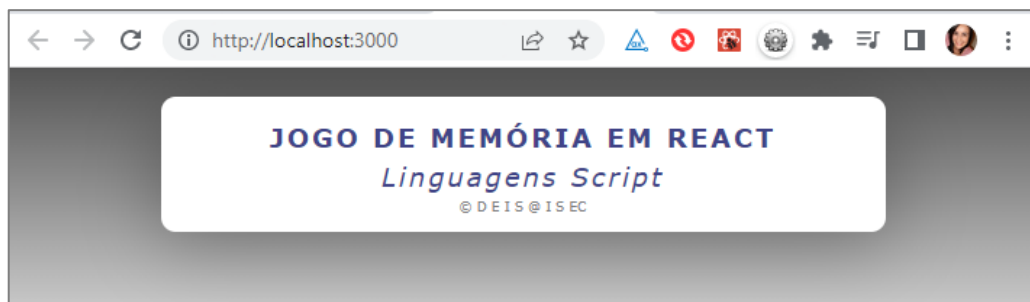


Figura 7 - Header e Footer

b. Implemente o componente `ControlPanel`, com nome de ficheiro `control-panel.component.jsx`, e o componente `GamePanel`, com `game-panel.component.jsx`, sendo que este último, nesta fase, não deverá conter as peças que compõem o jogo. Tenha em consideração:

- Estes componentes não deverão ficar funcionais, apenas a interface deverá ficar definida como o resultado apresentado na figura seguinte.
- Não se esqueça de especificar o *import* do ficheiro de estilos no elemento `ControlPanel`:

```
import "./control-panel.css";
```

- A estrutura HTML para estes componentes encontra-se na página seguinte.
 - Não se esqueça de converter os atributos HTML, para JSX, sempre que necessário (**class**, **for** para **htmlFor**,...).
- Visualize a aplicação no browser, e corrija algum erro que possa existir na consola.

```

</header>
<main class="main-content">
  <!-- Painel de Controlo-->
  <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 selected value="0">Selecione...</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="left hide">
        Clique em Iniciar o Jogo!
      </p>
      <dl class="list-item left">
        <dt>Tempo de Jogo:</dt>
        <dd id="gameTime">0</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>
  <section class="game-panel">
    <h3 class="sr-only">Peças do Jogo</h3>
    <div id="game">
      PAINEL DAS PEÇAS DE JOGO
    </div>
  </section>
</main>
<footer> ...

```

→ Visualize a aplicação no browser que deverá ter o aspeto seguinte.

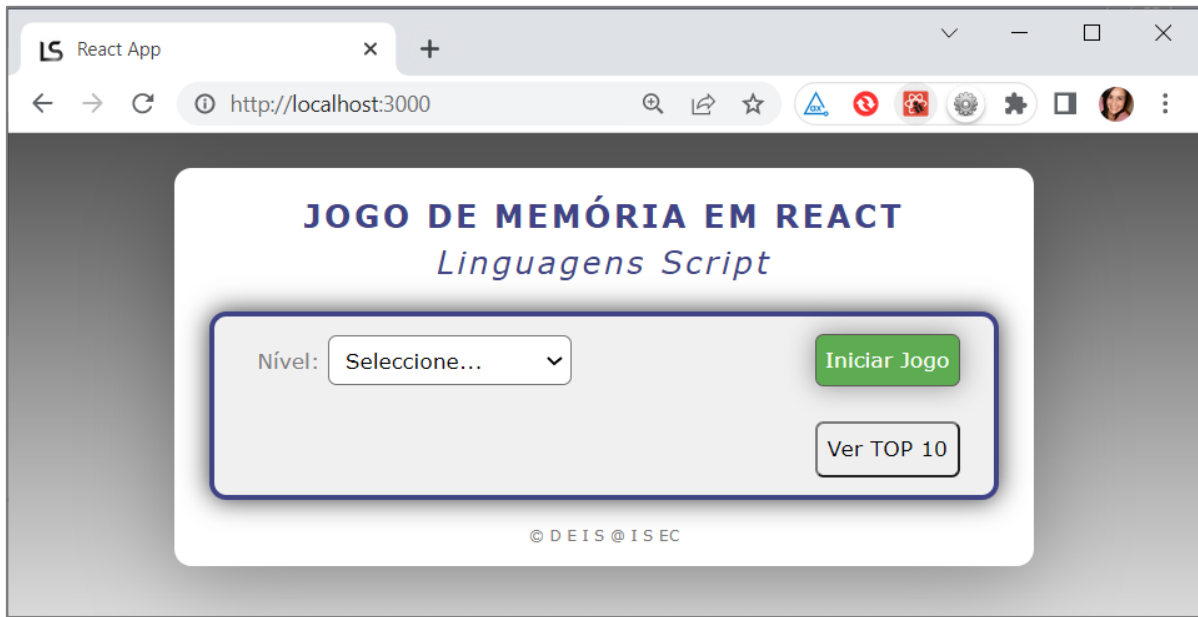


Figura 8 – ControlPanel

2> De forma a simplificar os *import* entre os vários componentes, efetue os seguintes passos:

- a. Crie o ficheiro **index.js** na pasta **components**.
- b. Para cada um dos componentes criados, copie a linha de código abaixo, devendo efetuar o mesmo export, adaptando, para os restantes componentes implementados.

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

- c. Agora, no ficheiro App.js altere o modo como efetua os imports, da seguinte forma:

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

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

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


- 3> Implemente o componente **Card**, com nome de ficheiro **card.component.jsx**. Para implementação deste componente especifique apenas uma propriedade, 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 um conjunto de cartas implementadas em HTML, a estrutura seria, por exemplo, a que se apresenta na página seguinte.
- Repare que no ficheiro **index.js** da pasta **constants**, contém duas constantes que deverão ser usadas na criação deste componente, nomeadamente **PLACEHOLDER_CARD_PATH** e **PLACEHOLDER_CARDBACK_PATH** que especificam o caminho onde se encontram os ficheiros das imagens, seja do logótipo como do **ls.png**, respetivamente.

```
<div class="card" data-logo="angular">
  
  
</div>
<div class="card" data-logo="bootstrap">
  
  
</div>
<div class="card" data-logo="html">
  
  
</div>
<div class="card" data-logo="javascript">
  
  
</div>
<div class="card" data-logo="vue">
  
  
</div>
```

- Invoque o componente **Card** no componente **GamePanel**, tantas vezes quantas necessárias para apresentar as seis cartas com os logótipos acima apresentados.
- Visualize a aplicação no browser que deverá ter o aspeto apresentado na figura 9.



Figura 9 - Ficha 8

- Para que os logotipos fiquem visíveis, aplique a classe `flipped` à **Card** por forma a que as cartas fiquem viradas como se mostra na figura seguinte.



Figura 10 - Ficha 8 Concluída

- Visualize o código do `GamePanel1` e, como pode verificar, o componente `Card` é invocado tantas vezes quanto o número de cartas, alterando apenas o seu *name*. Desse modo, simplifique o código, recorrendo a um array com os diferentes nomes e recorra a um método para invocar várias vezes o componente `Card` reduzindo assim o código do `GamePanel1`.