› 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.

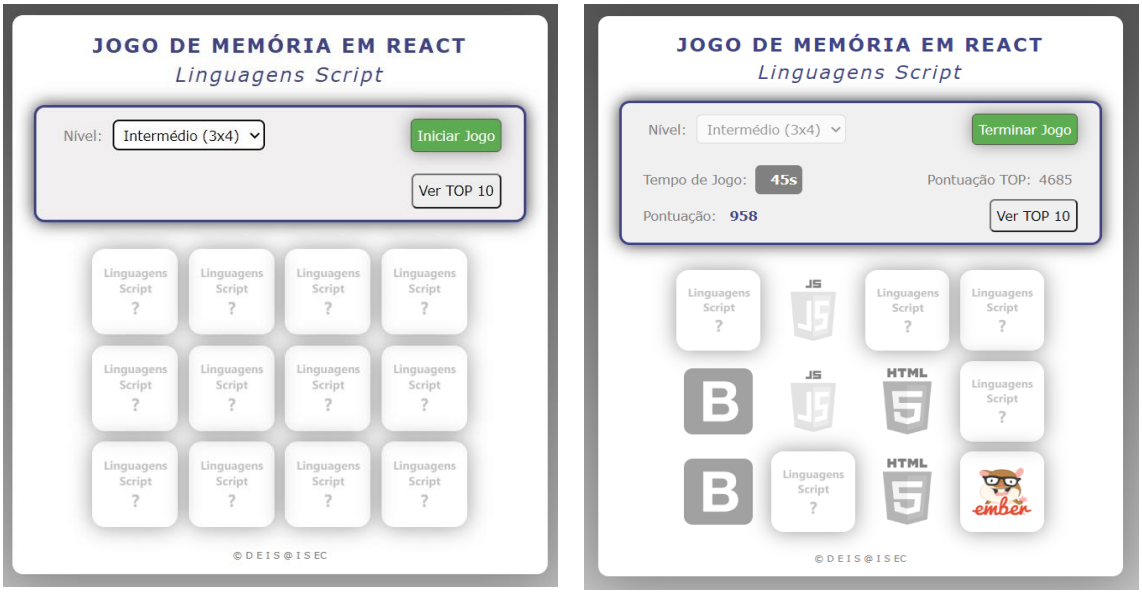


Figura 1 - Jogo de Memória - Versão React

› Preparação do ambiente

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

b. Considera-se que os alunos já tenham o node.js instalado no computador, caso contrário, devem efetuar a instalção (passos disponíveis na ficha prática nº7).

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/ .

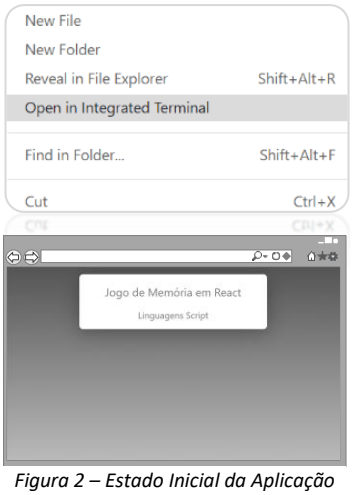


Figura 2 – Estado Inicial da Aplicação

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.

```
package.json
README.md

public
    favicon.ico
    index.html
    manifest.json
    _robots.txt

src
    App.js
    index.js

    assets
        styles
            App.css
            index.css
            w3.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

        header
    constants
        index.js

    helpers
        checkIfIsFlipped.js
        checkIfIsMatched.js
        index.js
        shuffle.js
```

Figura 3 - Estrutura de Directorios

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

- ***assets***: onde estão especificados os estilos globais;
- ***componentes***: 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

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 Memoria. Assim, destacam-se os seguintes componentes principais, como se pode ver na página seguinte, Figura 4:

→ **Header:** corresponde ao componente que contém o cabeçalho como os títulos;

→ **ControlPanel:** componente que conterá 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.

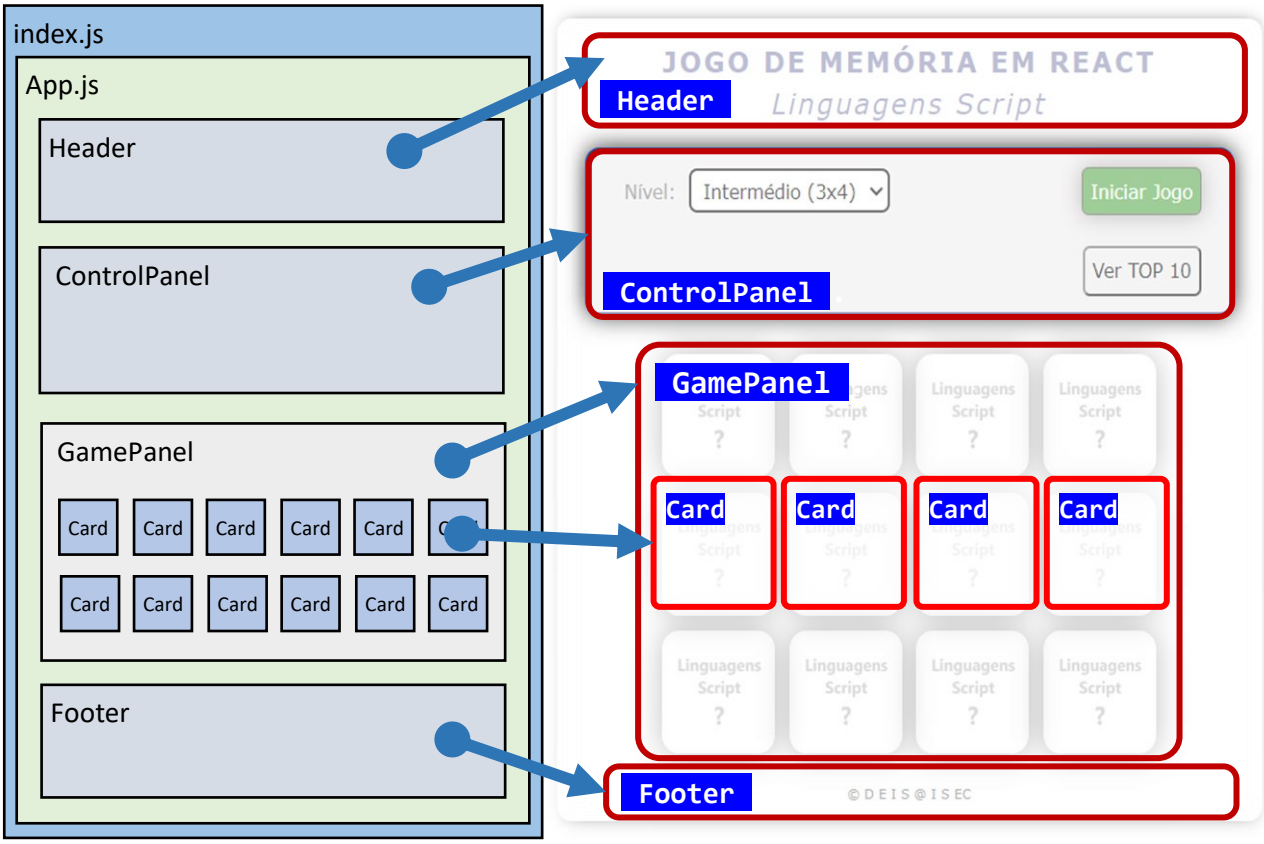


Figura 4 - Divisão aplicação em Componentes

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 **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>
);
```

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 adefinição da funsão ser substituida
// export default function App() {
```

Código 2 - export default

- Na secção seguinte, sugere-se ainda a criação de um ficheiro index.js na pasta components, de forma a concentrar todos os *export*, 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.

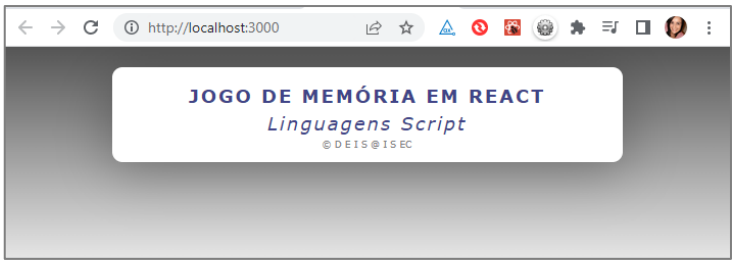
a. Tendo em consideração o código HTML que se apresenta abaixo, que especifica a estrutura do cabeçalho e rodapé da aplicação, implemente o componente `Header` e o componente `Footer`, cujos nome dos ficheiros deverão ser **`header.component.jsx`** e **`footer.component.jsx`**, respectivamente.

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 EC</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.



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 4.
- 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 (página 6).
  - 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.

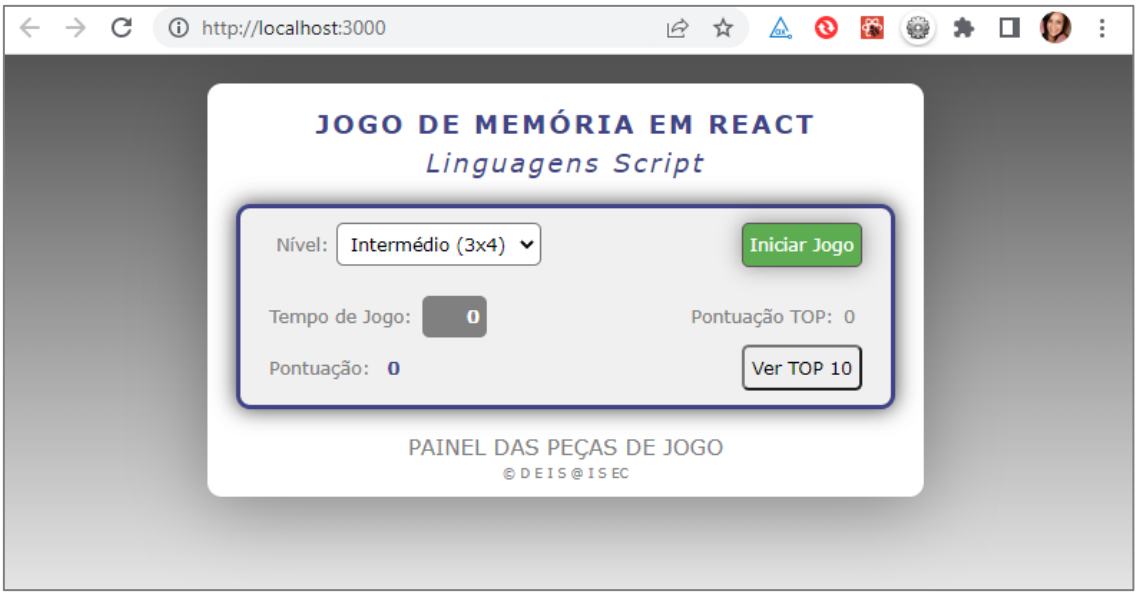


Figura 5 – ControlPanel

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, para importar um determinado componente, por exemplo no **GamePanel**, que será necessário importar o card, o caminho ficará simplificado desta forma

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

```
</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="col1 form-group">
              <label for="btLevel">Nível:</label>
              <select id="btLevel">
                <option selected 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" class="col2">Iniciar Jogo</button>
          </form>
          <div class="form-metadata">
            <p id="message" role="alert" class="hide">
              Clique em Iniciar o Jogo!
            </p>
            <dl class="col1 gameStarted list-item">
              <dt>Tempo de Jogo:</dt>
              <dd id="gameTime">0</dd>
            </dl>
            <dl class="col2 gameStarted list-item">
              <dt>Pontuação TOP:</dt>
              <dd id="pointsTop">0</dd>
            </dl>
            <dl class="col1 gameStarted list-item">
              <dt>Pontuação:</dt>
              <dd id="points">0</dd>
            </dl>
            <div id="top10" class="col2">
              <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> …
```

3> Por fim, implemente o componente **Card**, com nome de ficheiro **`card.component.jsx`**. Para implementação deste componente especifique apenas a propriedade **name.**

→ Tendo em consideração um conjunto de cartas implementadas em HTML, a estrutura seria, por exemplo, a que se apresenta de seguida.

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

→ Repare que no ficheiro **index.js** da pasta **constants**, contém duas contantes 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 logotipo como do ls.png, respectivamente.

→ Invoque o componente **Card** no componente **GamePanel**, tantas vezes quantas necessárias para apresentar as seis cartas com o os logotipos acima apresentados.

→ Visualize a aplicação no browser. Para que os logotipos fiquem visíveis, adicione a classe `flipped` à Card por forma a que as cartas fiquem viradas como se mostra na figura seguinte.