TDW - Módulo B - Miniprojeto Javascript
Blackjack by Rato

André da Silva Rato (115030) - PROJETO

1 de novembro de 2022

1 Introdução

O presente relatório pretende clarificar e descrever os tópicos mais importantes da implementação do projeto desenvolvido no módulo 1B da unidade curricular **Tecnologias e Desenvolvimento Web**, no âmbito do **Mestrado em Comunicações e Tecnologias Web** da Universidade de Aveiro.

De modo a demonstrar o domínio dos conceitos lecionados durante as aulas da UC, decidi implementar uma versão simplificada do jogo **Blackjack**, criando assim uma SPA (*Single Page App*) que permite que um utilizador, o *Player*, jogue contra um oponente fictício, o *Dealer*.

É de notar que, no código fonte do projeto, o ficheiro `scripts.js` apenas é carregado no final do ficheiro `index.html`. Isto acontece pois era necessário que a página estivesse totalmente carregada para que todos os blocos de scripting funcionassem na íntegra.

2 Estratégia de implementação geral

Para a implementação do jogo Blackjack em Javascript, foram utilizadas duas bibliotecas externas, *jQuery* e *Bootstrap*. A página implementada é composta por um *header*, situado no topo, um "*gameboard*", a área de jogo, e um *footer*, situado no final da página.

2.1 API

De modo a fazer a gestão do baralho que está a ser utilizado para o jogo, e de modo a obter uma carta a cada "*draw*", foram utilizados os seguintes *endpoints* da API "*Deck of Card*":

- `https://deckofcardsapi.com/api/deck/new/` - cria um *deck* novo e retorna o id do mesmo;
- `https://deckofcardsapi.com/api/deck/<<deck_id>>/draw/?count=1` - obtém uma carta do baralho gerado anteriormente, passando o id do mesmo no url.

2.2 Conexão à API

Para realizar pedidos à API foi necessário criar uma função genérica que recebesse, não só o url do *endpoint* a ser acedido, mas também o método associado ao pedido e a função que seria executada após o término do mesmo. A função `makeRequest(url, method, callback)` utiliza um objeto `XMLHttpRequest` para realizar a conexão entre o *browser* do utilizador e a API. É de notar que o método utilizado em todos os pedidos feitos pela página é o métogo *GET*.

2.3 Funcionalidades extra

Para além da funcionalidade de jogo que o projeto possui, é ainda possível:

- listar os jogos anteriores;
- acender ou apagar as luzes (alternar entre *dark and light mode*);
- apostar, ou não, de modo a obter *RATS* (moeda do jogo);
- fazer *reset* na carteira, clicando no logo da página (apenas funciona se a carteira estiver vazia);

- aceder à lista de regras e instruções do jogo;
- transferir o relatório do projeto.

3 Elementos dinâmicos implementados

A implementação de elementos dinâmicos permitiu que o mesmo excerto de código fosse reaproveitado para a renderização do mesmo conteúdo em vários momentos do projeto. Neste caso, e para o desenvolvimento do jogo, em si, foi criado um *web component*, o *playing-card*, permitindo assim ter uma estrutura fixa para a representação visual das cartas de jogo.

Este componente é responsável por gerar o código necessário para a visualização de uma carta de jogo. Recebe os seguintes atributos:

- ***cardStyle*** - estilos para serem aplicados à carta no geral;
- ***player*** - determina qual o jogador que obteve a carta ('*true*' caso seja o utilizador, ou '*false*', caso seja o computador);
- ***suit*** - determina qual o símbolo do naipe que é renderizado;
- ***suitStyle*** - estilos para serem aplicados ao símbolo do naipe que é renderizado;
- ***value*** - valor a ser mostrado na carta.

3.1 Exemplo de utilização

```
<playing-card card-style="border-color: purple" player="true" suit="SPADES" value="A"
   suit-style="filter: brightness(0)" />
```

4 Desafios técnicos

4.1 Armazenamento de dados

Visto que não foi requerido a implementação de uma API, tive de pensar numa estratégia alternativa para realizar a gestão do histórico de jogos e da carteira do utilizador. A utilização de *browser Cookies* foi a solução que considerei mais adequada, tornando-se assim possível fazer a gestão dos dados do utilizador.

4.1.1 Histórico de jogos

O histórico de jogos é guardado na *cookie* '`history`', sendo atualizada sempre que uma partida termina. Visto que não é possível guardar *arrays* ou listas em *cookies*, foi necessário encontrar uma forma de guardar as informações da partida. Foi então pensada a estratégia de possuir um caractere que separasse cada elemento do *array*, sendo que neste caso foi utilizado o '`a`'. No caso '`JOGO1@JOGO2@JOGO3`', possuímos informação relativa aos jogos 1, 2 e 3.

Para além da utilização de um caractere de separação, foi necessário também pensar numa estrutura, em *string*, que permitisse guardar os pontos do utilizador, os pontos do computador e o resultado final da partida. A solução a que recorri foi à criação de uma *string* com a estrutura `pScore cScore bValue rFinal`, sendo `pScore` e `cScore` a quantidade de pontos obtidos na partida por parte do utilizador e do computador, respetivamente, `bValue` o valor ganho ou perdido, e `rFinal` o resultado final, tendo em vista, sempre, a perspetiva do utilizador. No entanto, não é possível guardar o caractere espaço em *cookies*, convertendo então o caractere espaço na sequência de caracteres '`%20`. Foi então possível definir que cada jogo teria a estrutura '`pScore%20cScore%20bValue%20rFinal`'.

4.1.2 Carteira RATS

À semelhança do histórico de jogos, a quantidade de *RATS* que o utilizador possui é guardada nas *cookies*, sendo, neste caso, utilizada a *cookie* '`EDRrMakaRAT5`' (que significa "*Euro, Dolar or Real Are Money AKA RATS*"). O valor desta *cookie* é alterado sempre que há uma alteração na carteira do *Player*.

4.2 O *Dealer*, uma entidade *No-AI*

De modo a tornar o jogo interativo, e a não ser apenas um jogo entediante em que o utilizador apenas tem de tentar obter, exatamente, 21 pontos, foi adicionado um adversário, o *Dealer*. Embora pareça, que o *Dealer* executa vários cálculos para determinar se deve obter mais uma carta ou não, isso não acontece. A implementação de uma entidade de inteligência virtual seria facilitada se outras tecnologias fossem utilizadas, tais como linguagens declarativas, como é o caso do *prolog*. No entanto, e mesmo não tendo sido implementada nenhuma inteligência artificial no projeto, o *Dealer* continua a ser capaz de executar jogadas "inteligentes"durante o decorrer de uma partida. A função que determina se o *Dealer* deve continuar a obter cartas ou se deve parar possui a seguinte lógica:

1. o *Dealer* obtém uma carta:
   (a) se a quantidade de pontos obtidos após o "*draw*"da carta for menor que 18, ou então, se a quantidade de pontos do *Dealer* for menor que a quantidade de pontos do *Player*, o *Dealer* obtém mais uma carta;
   (b) se não, o jogo termina;
2. o turno do *Dealer* continua enquanto for possível.

Cada jogada do Dealer demora 3 segundos (3 milissegundos) a ser iniciada, de modo a que o Player consiga ter melhor noção das decisões que o Dealer está a fazer.

5 Conclusões

Com a realização deste trabalho foi possível:

- melhorar o meu nível de programação relacionado à utilização "*raw*"de Javascript para a manipulação de páginas *web*;
- adquirir novos conhecimentos relacionados com as boas práticas da programação na *web*;
- proporcionar um aumento significativo ao nível de processos de criação de ideias;
- interpretar um problema e identificar possíveis soluções;
- utilizar soluções "*out of the box*"para a implementação de funcionalidades extra;
- colocar em prática os conceitos que adquiri na licenciatura relativos a servidores que utilizem FTP (*File Transfer Protocol*).

Embora tenha conseguido desenvolver este projeto sem muitas dificuldades, tornou-se óbvio que senti alguma necessidade de utilizar ferramentas (bibliotecas e/ou *frameworks*) que facilitam a implementação de páginas e aplicações *web*, como é o caso do *React.js* e do *Vue.js*. No entanto, foi uma "batalha"saudável que me permitiu valorizar ainda mais quem desenvolve soluções *web* apenas com "*raw*"Javascript.