Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Programação na Internet
Semestre de Verão de 2016/2017
1ª Época (*27 de Junho de 2017)* – Duração: 2h30

Grupo 1

1. [4] Justifique as seguintes questões:

   a. [1] Indique 3 formas de passagem de parâmetros de um *web agent* (e.g. *browser*) a uma aplicação web e como são acedidos esses parâmetros numa aplicação *express*.

   b. [1] Numa aplicação *express* a escrita de um cookie através do método `cookie()` do objecto `resp` (e.g. `resp.cookie({name}, {value})`) traduz-se em quê na resposta HTTP?

   c. [1] Se o cookie for escrito no *front-end* (Javascript Cliente) qual o resultado no pedido HTTP? Justifique.

   d. [1] Dado: `function foo(nr){let a = nr;  return () => a }`

   Justifique qual das seguintes afirmações está correcta:
   "A expressão `() => a` cria um *closure* sobre a variável `a` declarada no *scope* da função `foo`."
   "A expressão `() => a` cria um *scope* sobre a variável `a` declarada no `closure` da função `foo`."

2. [2] Quais as mensagens que são escritas na consola quando são feitos dois pedidos HTTP pela seguinte ordem: 1) http://localhost:3000/foo, 2) http://localhost:3000/bar a um servidor HTTP que corre a aplicação seguinte.

```
const express = require('express')
const app = express()
let wait = true
app.get('/foo', (req, res) => {
    console.log('foo')
    res.end()
    while(wait) { }
    console.log('finish')
})
app.get('/bar', (req, res) => {
    wait = false;
    console.log('bar')
    resp.end()
})
app.listen(3000)
```

3. [2] Implementes a função `memoize(fn)` que retorna uma nova função que redirecciona as chamadas para a função `fn` e retorna os seus resultados mantendo-os em *cache*.

   A função `memoize(fn)` retorna uma nova função `m` cujos resultados são iguais aos da execução da função `fn` com os mesmos argumentos. Contudo, em futuras chamadas da função `m` com argumentos já usados anteriormente, a função `m` retorna os valores mantidos em cache.

   Implemente a função `unmemoize(m)` que desfaz o efeito da função `memoize`. Ou seja, retorna a função original `fn` que foi passada a `memoize`. Se a função `m` não corresponder a uma função que tenha sido *memoized* então é lançado um erro.

   Nota: não pode usar variáveis globais na implementação deste exercício.

Grupo 2

[12] Considere a *view* `movies.hbs` como parte da implementação de uma aplicação web para gestão de filmes que tem as funcionalidades de: listagem de filmes favoritos e adição de opinião sobre um filme.

```
{{#> layout}}
<script language='javascript' type='text/javascript' src='/assets/js/movies.js'></script>
<table class="table">
  <tr>
    <th>Title</th><th>Duration</th><th>Rating</th><th>Opinion</th>
  </tr>
  {{#each movies}}
    <tr>
      <td>{{title}}</td><td>{{duration}}</td><td>{{rating}}</td>
      <td>
        <div id="div{{movieId}}">
          <span id="span{{movieId}}">{{opinion}}</span>
          <button onclick="editOpinion({{movieId}})"class="btn">EDIT</button>
        </div>
        <div id="edit{{movieId}}" style="display: none">
          <textarea id="txt{{movieId}}"></textarea>
          <button onclick="addOpinion({{movieId}})" class="btn">Update</button>
        </div>
      </td>
    </tr>
  {{/each}}
</table>
{{/layout}}
```

Title	Duration	Rating	Opinion
Pulp Fiction	154	8.2	Filme pouco convencional EDIT
Fight Club	139	8.2	Filosófico pouco Update
Match Point	124	7.2	WoW que brutalidade EDIT

A aplicação usa dados de uma Web API (http://moviesapi.net) e dados próprios armazenados numa BD CouchDB.
A Web API http://moviesapi.net tem um único *endpoint* http://moviesapi.net/{movieId} que retorna um documento Json com a informação do filme com identificador movieId e as propriedades: *title*, *duration* e *rating*.
A BD CouchDB tem o nome `movies` e os *endpoints*:

- `GET localhost:5984/movies/{userid}` – retorna o documento com `_id` igual a `userid`
- `POST localhost:5984/movies` – insere um novo documento passado no corpo do pedido
- `PUT localhost:5984/movies/{userid}` – actualiza o documento com `_id` igual a `userid`

A BD guarda para cada utilizador a sua lista de filmes favoritos. A BD apenas guarda o ID do filme (`movieId`) sendo os restantes dados obtidos de moviesapi.net. Para cada filme a BD pode guardar a opinião do seu utilizador.

Admitindo que `app` refere uma instância de um servidor HTTP *express*, configurado com uma instância de passport referida pela variável `passport`:

a. [1] Defina e proponha uma organização para os dados da aplicação na CouchDB, dando um exemplo de um documento Json para um determinado utilizador que tem como favoritos os filmes com ids: 127, 203, 540.

b. [2] Configure a instância de `passport` e implemente os seus métodos `serializeUser`, `deserializeUser` e autenticação.

c. [4] Implemente um *middleware* e adicione a `app` a rota `/movies` para listagem dos filmes favoritos do utilizador que estiver autenticado. Caso não esteja autenticado deverá redireccionar para `/login`.

d. [2] Implemente um *middleware* e adicione a `app` a rota PUT `/api/movies/{movieId}/opinion` que guarda na CouchDB uma opinião para o filme com *id* `movieId` do utilizador autenticado.
O texto da opinião é recebido no corpo do pedido HTTP num formato JSON.
Este *endpoint* será acedido via pedido AJAX podendo retornar as respostas: 401 se o utilizador não estiver autenticado, ou 500 em caso de erro da CouchDB com a respectiva mensagem de erro, ou 200 em caso de sucesso.

e. [3] Implemente os métodos de *front-end* (JavaScript cliente) `editOpinion` e `addOpinion`, que tornam alternadamente visível uma divisória (`div{{movieId}}` ou `edit{{movieId}}`) escondendo a outra através da propriedade de estilo `style.display = "none"`.
Cada um dos métodos copia o conteúdo de `span{{movieId}}` para `txt{{movieId}}` e vice-versa.
Além disso, o método `addOpinion` faz um pedido HTTP via AJAX ao endpoint da alínea anterior.

Os Docentes,
Luís Falcão e Miguel Carvalho