Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Programação na Internet
Semestre de Verão de 2016/2017
2ª Época (*14 de Julho de 2017)* – Duração: 2h30

Grupo 1

1. [5] Justifique as respostas às seguintes questões:

   a. [1] Existe alguma relação entre o significado de URI e fragmento na arquitetura da Web?

   b. [1] Admita a existência de uma aplicação express `app` configurada com 2 *middlewares* `mw1` e `mw2`, onde:
      1. são recebidos 2 pedidos HTTP que serão atendidos respectivamente por `mw1` e `mw2`;
      2. `mw1` começa a ser executado antes que `mw2` atenda o 2º pedido HTTP;

      Em que condições é que `mw2` pode atender um 2º pedido, antes que `mw1` conclua toda a sua execução relativa ao 1º pedido?

   c. [1] Qual a diferença entre um elemento HTML como um modo de display de bloco ou *inline*?

   d. [1] Existe diferença entre o significado das propriedades `status` e `readyState` de `XMLHttpRequest`?

   e. [1] Dado: `function A() { } let x = new A(), y = new A(); x.____ = ____; x.foo(); y.foo();` Complete a instrução `x.____ = ____;` de modo a que o resultado da chamada `x.foo();` e `y.foo();` imprima na consola o texto `ola`. NOTA a variável `y` não pode ser utilizada na resolução deste problema.

2. [3]

   a) [1,5] Implemente a função `parallel` com assinatura `function (tasks, cb)` que invoca todas as funções passadas no *array* `tasks`. Estas funções obedecem à convenção de *callback* do node.js e recebem um único parâmetro correspondente ao *callback*. Se algumas das funções devolver um erro então `cb` é chamado de imediato com esse erro. Se não, `cb` será chamado apenas quando todas as funções tiverem concluído. Neste caso, é passado a `cb` um *array* com os resultados de cada função, onde cada resultado ocupa o mesmo índice da sua função em `tasks`. Exemplo:

```
const a1 = [
    cb => setTimeout(() => cb(null, 'ola'), 200),
    cb => setTimeout(() => cb(null, 'super'), 100),
    cb => setTimeout(() => cb(null, 'maria'), 300)
]
```

```
parallel(a1, (err, data) => {
  /* data = ['ola', 'super', 'maria']*/
})
```

   b) [1,5] Implemente a função `retry` com assinatura `function (tasks, cb)`. O parâmetro `tasks` é um *array* de objectos com as propriedades `times` e `task`, onde `task` é uma função que recebe um único parâmetro *callback* e `times` é o número de vezes máxima que a função é chamada se devolver erro. Se `times` tiver o valor `n` e a `n` ézima chamada da função retornar erro então `cb` é chamado de imediato com esse erro.
   Se todas as funções devolverem um resultado sem erro, num número de vezes inferior ao de `times`, então é passado a `cb` um *array* com os resultados de cada função nas mesmas condições da alínea anterior.
   Exemplo:

```
let count = 0
const a2 = [
  {
    times: 7,
    task: cb => {
            if(count++ < 3) setTimeout(() => cb(new Error()), 200)
            else setTimeout(() => cb(null, 'ola'), 300)
          }
  }, {
    times: 1,
    task: cb => setTimeout(() => cb(new Error(), 'super'), 200)
  }
]
```

```
retry(a2, (err, data) => {
  /* data = ['ola', 'super']*/

  a2[0].times = 2

  retry(a2, (err, data) => {
    /* err = Error instance */
  })
})
```

Grupo 2

[12] Considere a *view* `artists.hbs` como parte da implementação de uma aplicação web para gestão dos músicos favoritos de um utilizador e que tem as funcionalidades de: listagem dos músicos favoritos e remoção de um músico.

```
{{#> layout}}
<script language='javascript' type='text/javascript' src='/assets/js/musics.js'></script>
<table class="table">
  <tr><th>Artist</th><th>Top Track</th><th>Play Count</th><th></th></tr>
  {{#each artists}}
    <tr id="row{{mbid}}">
      <td>{{name}}</td><td>{{trackName}}</td><td>{{playcount}}</td>
      <td><button onclick="removeArtist({{mbid}})">Remove</button></td>
    </tr>
  {{/each}}
</table>
{{/layout}}
```

Artist	Top Track	Play Count
Ana Moura	Desfado	40518
Luminosity	Oye Como Va	995430
Muse	Knights of Cydonia	8877323

A aplicação usa dados de uma Web API (http://musicsapi.net) e dados próprios armazenados numa BD CouchDB.
A Web API http://musicsapi.net tem 2 *endpoints*:

- http://musicsapi.net/artists/{mbid}/tracks que retorna um *array* Json com todas as músicas do artista com identificador `mbid` e onde cada elemento tem as propriedades: `trackName` e `trackid`.
- http://musicsapi.net/tracks/{trackid} que retorna um documento Json com todas os detalhes da música identificador por `trackid` contendo as propriedades: `trackName` *e* `playcount`.

A BD está acessível no *path* `localhost:5984/musics` e cada documento tem as propriedades do exemplo seguinte:

```
  "_id": "zemanel",
  "password": 123,
  "artists": [ { "name": "Muse", "mbid": "9c9f1380-2516-4fc9-a3e6-f9f61941d090" },
               { "name": "Ana Moura", "mbid": "b5954bc9-bd18-4064-a17a-1ad719e617a5" },
               { "name": "Luminosity", "mbid": "9a3bf45c-347d-4630-894d-7cf3e8e0b632" } ]
```

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

a. [1] Defina a assinatura da função `request` usada para fazer pedidos HTTP na aplicação node.js explicando o significado de cada parâmetro. Defina também a assinatura da função `ajaxReq` usada para fazer pedidos HTTP no cliente, explicando o significado de cada parâmetro. **Não implemente o corpo destas funções**. Reutilize estas funções nas próximas alíneas sempre que necessário.

b. [1] Implemente o módulo `usersService` que exporta as funções `find` e `authenticate`. Admita que estas são usadas respectivamente nas operações `deserializeUser` e `authenticate` do `passport`.

c. [3] Implemente o módulo `musicsService` que exporta a função `getTopTrack` com a assinatura `(mbid, cb) => void`, que passa a cb um objeto com propriedades `trackName` e `playcount`, correspondente à música mais tocada (com maior número de `playcount`) do artista com o identificador `mbid`.

d. [2] Implemente um *middleware* e adicione a `app` a rota `/artists` que apresenta a listagem das músicas favoritas do utilizador que estiver autenticado, através da *view* `artists.hbs` apresentada. Caso não esteja autenticado deverá redirecionar para `/login`.

e. [2] Implemente no módulo `musicsService` a função `removeArtist`, que remove da CouchDB um determinado artista da lista de favoritos de um utilizador. **Leia a alínea seguinte antes de implementar esta alínea.** A assinatura de `removeArtist` deve ser definida de acordo com os requisitos da alínea seguinte.

f. [1] Implemente um *middleware* e adicione a rota DELETE `/api/artists/{mbid}` em `app`, que remove da CouchDB o artista com o identificador `mbid` da lista de favoritos do utilizador autenticado. Este *endpoint* será acedido através de um 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.

g. [2] Implemente a função de cliente `removeArtist`, que faz um pedido HTTP ao *endpoint* criado na alínea anterior, para remoção do artista da lista de favoritos do utilizador autenticado.
A respectiva linha da tabela será removida em caso de sucesso. Em caso de erro é apresentado um painel de alerta com a respectiva mensagem.

Os Docentes,
Luís Falcão e Miguel Carvalho