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 Janeiro de 2017*) – Duração: 2h (+30 min tolerância)

Grupo 1

1. [3] Classifique as seguintes afirmações de verdadeiras ou falsas e justifique essa classificação.
   a. [1] Todos os elementos HTML têm o seu Box Model.
   b. [1] Em JavaScript, as funções são objetos e como tal, podem ser usadas em todos os contextos onde é válida a sua utilização.
   c. [1] O protocolo HTTP tem um modo de interação pedido-resposta. Este modo, não impõe qualquer restrição às aplicações que usam este protocolo.
2. [2] Considere o seguinte pedido e respetiva resposta HTTP:

Pedido	**Resposta**
POST /tasks HTTP/1.1 Host: localhost:3500 Accept: */* Content-Type: application/x-www-form-urlencoded Content-Length: 45 title=Note Title&description=Note Description	HTTP/1.1 302 Found X-Powered-By: Express Location: /tasks/28 Date: Fri, 27 Jan 2017 16:05:46 GMT Connection: keep-alive

Indique e justifique toda a informação que pode ser obtida desta resposta, nomeadamente o que é suposto o pedido realizar na componente servidora da aplicação, se a operação foi concluída com sucesso do ponto de vista da aplicação e o porquê desta resposta.

3. [3] Os objetos que num browser representam elementos HTML, têm propriedades com o nome `on<Event>`, que permitem registar apenas uma função como *handler* do evento <Event> (exº `onclick`). A função `addHandler()` permite suprimir esta lacuna. Em seguida, apresenta-se excerto de código que utiliza esta função. Assuma que na página onde está este código, existe um botão com o id `btn`. Quando esse botão for clicado, aparecem as mensagens `handler1` e `handler2` na consola.

```
1  let btn = document.getElementById("btn");
2  addHandler(btn, "onclick", h1).then(h2).then(h3);
3  function h1(e) { console.log("handler1"); }
4  function h2(e) { console.log("handler2"); return false; }
5  function h3(e) { console.log("handler3"); }
```

Implemente a função `addHandler()` e de modo a ter o comportamento descrito e apresentado no exemplo.

4. [3] Considere o módulo Node `ensure-login.js`. Este módulo retorna um *middleware* compatível com o `express`, que verifica se quem fez um pedido HTTP é um utilizador autenticado ou não. Se for, deixa seguir o pedido para o próximo *middleware*. Caso contrário retorna uma resposta 302 ou 403, consoante o método `ensureLoggedIn()` é chamado previamente ou não.
   O código seguinte apresenta um exemplo de utilização deste modulo numa aplicação Express.

```
1  'use strict';
2  const PORT = 3500;
3  const express = require('express');
4  const ensureLoggedIn = require('./ensure-login');
5  //const ensureLoggedIn1 = require('./ensure-login').ensureLoggedIn('/login');
6  const app = express();
7  var passport = require('passport'); // O módulo passport e a respetiva estratégia são registados
8  também na aplicação. Esse código não está no exemplo
9
10 app.get('/public', function (req, rsp) {
11     rsp.end("Public");
12 });
13
14 app.get('/private', ensureLoggedIn, function (req, rsp) {
15     rsp.end("private");
16 });

   app.listen(PORT, () => console.log("Listening on port " + PORT));
```

No exemplo anterior, um acesso à *path* `/public` por qualquer utilizador retorna o texto `Public`. Um acesso à *path* `/private` por um utilizador não autenticado, retorna uma resposta com código 403, mas se comentarmos a linha 4 e removermos o comentário da linha 5, retorna uma resposta 302 para a localização `/login`.

   a. [2] Implemente o módulo `ensure-login`, de modo a apresentar a funcionalidade descrita.
   b. [1] Caso se registasse um *middleware* de autenticação numa aplicação (exº `passport`) que usa o módulo `ensure-login`, para os *endpoints* que se pretende apenas dar acesso a utilizadores autenticados, qual dos *midlewares* deveria ser registado em primeiro lugar? Justifique.

Grupo 2

5. [9] Considere a *view* `tasks.hbs` como parte da implementação de uma aplicação web para gestão de tarefas que tem as funcionalidades de: listagem, adição e remoção de tarefas.

```
<div class='panel'>
 <table class='table'>
   <thead>
     <tr>
       <th>Name</th>
       <th>Description</th>
       <th> </th>
     </tr>
   </thead>
   <tbody id='tasksList'>
     {{#each tasks}}
     <tr id='task{{_id}}'>
       <td>{{name}}</td>
       <td>{{description}}</td>
       <td>
         <button>delete</button>
       </td>
     </tr>
     {{/each}}
   </tbody>
 </table>
</div>
```

```
<div>
 <form action='/tasks' method='post'>
   <div class='form-group'>
     <label>Name:</label>
     <input type='text' name='name'/>
     <label>Description:</label>
     <input type='text' name='description'/>
     <input type='submit' value='Add Task'/>
   </div>
 </form>
</div>
<div id='panelAlerts' class='alert'>
</div>
```

Os dados são armazenados numa BD CouchDB, que tem os *endpoints*:

- `GET localhost:5984/tasks/:id` – retorna o documento com `_id` igual a `:id`
- `GET localhost:5984/tasks/_all_docs?include_docs=true` – retorna todos os documentos na forma: `{rows: [{doc: {_id: …, _rev: …}}, {doc: {…}}, …]}`
- `POST localhost:5984/tasks` – insere o documento passado no corpo do pedido
- `DELETE localhost:5984/tasks/:id?rev=:rev` – remove o documento com `_id` e `_rev` iguais a `:id` e `:rev`.

a. [2] Implemente o módulo `tasks-service` com os métodos `getAll(…)` e `add(…)` que implementam as operações de listagem de tarefas e adição de nova tarefa sobre a CouchDB.
Use um módulo para realização dos pedidos HTTP sobre a CouchDB.

b. [2] Admitindo que `app` refere uma instância de um servidor HTTP *express*, implemente os *middlewares* e adicione a `app` as rotas que disponibilizam as operações de listagem de tarefas e adição de uma nova tarefa.
Use como base o módulo desenvolvido na alínea anterior e a *view* `tasks.hbs` fornecida.

c. [2] Adicione ao módulo `tasks-service` um método `remove(id, …)` que remove da CouchDB a tarefa com id passado por parâmetro.

d. [3] Implemente um *middleware* e adicione a `app` a rota para remoção de uma tarefa, que retorna 200 em caso de sucesso ou 503 em caso de erro com uma resposta em `json` no formato: `{ message: '…' }`
Adicione ao botão delete de `tasks.hbs` um *handler* (método Javascript) que realiza um pedido HTTP via AJAX para remoção de uma tarefa.
Em caso de sucesso remove a respetiva linha da tabela.
Em caso de erro apresenta o código de resposta e a respetiva mensagem de erro no painel `'panelAlerts'`.
Se for um erro de servidor deve apresentar a mensagem recebida na resposta json.

Os Docentes,
Luís Falcão e Miguel Carvalho