Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Programação na Internet
Semestre de Inverno de 2016/2017 - 2ª Época (*13 de Fevereiro de 2017)* – Duração: 2h (+30 min tolerância)

Grupo 1

1. [3] Justifique as respostas às seguintes questões:

   a. [1] Classifique de verdadeira ou falsa a seguinte afirmação: "*Um parâmetro de rota (route parameter) ou um parâmetro em query-string permitem definir a mesma informação num URI*".

   b. [1] Na implementação de um servidor HTTP com o módulo `express` o envio de um objeto `ctx` numa resposta através do método `json()` de `res`, i.e. `res.json(ctx)` é equivalente à execução de quais ações?

   c. [1] No contexto do módulo `passport`, após a autenticação com sucesso, como é implementado internamente pelo `passport` a obtenção do objeto `user` nos pedidos HTTP subsequentes?

2. [3] **Implemente** o módulo `cargo` cuja função *default* retorna um objeto (`pool`) que tem um limite máximo de funções (tarefas) que são chamadas (postas em execução).
   Quando é chamada uma tarefa é dada a mensagem "*callling <nome da função>*".
   No exemplo abaixo é criado um objecto `pool` com o limite de 2, onde são submetidas 3 tarefas (`task1`, `task2` e `task3`) e onde `task3` só é chamada quando uma das tarefas em execução tiver terminado.

```
function task1(done) {
 setTimeout(
   () => { debug('task1 end'); done() }, 10)
}

function task2(done) {
 setTimeout(
  () => { debug('task2 end'); done() }, 10)
}

function task3(done) {
 setTimeout(
  () => { debug('task3 end'); done() }, 10)
}
```

```
Const cargo = require('./cargo.js')

const pool = cargo(2)

debug('pushing task 1')
pool.push(task1)
debug('pushing task 2')
pool.push(task2)
debug('pushing task 3')
pool.push(task3)
```

```
Output:

pushing task 1
pushing task 2
pushing task 3
calling task1
calling task2
task1 end
calling task3
task2 end
task3 end
```

3. [2] **Implemente** o módulo `static` cuja função *default* retorna um novo *middleware* que responde com o conteúdo do ficheiro localizado num *path* igual ao do pedido HTTP. A função recebe por parâmetro:
   - `folder` – raz da directoria onde se localizam os ficheiros.
   - `extension` – extensão dos ficheiros admitidos.
   - `contentType` – contente type corresponde a `extension`.

   Se o *path* do pedido não tiver um sufixo igual à extensão configurada então passa ao próximo *middleware.*
   E.g. `const static = require('./static')('c:\\tmp', '.txt', 'text/plain')`

4. [2] **Implemente** o módulo `authz` cuja função *default* recebe uma função de autorização por parâmetro com a descrição: `(user, role, cb) => void`, tal como no exemplo seguinte `authz` é configurado com `authorize`.
   Neste exemplo a rota `'/foo'` só será atendida por `fooAction` se o utilizador autenticado tiver o role `'admin'`.

```
function authorize(user, role, cb) {
   // Em caso de erro chama cb com o respetivo objeto de erro.
   // Caso contrário, passa a cb o valor true ou false consoante user tenha ou não o role especificado.
}

const express = require(express)
const app = express()
// ... adicão de middlewares, incluindo autenticação via passport.

const authz = require('./authz')(authorize)
app.get('/foo', authz('admin'), fooAction) // só passa a fooAction se o utilizador tiver o role 'admin'.
app.get('/bar', authz('user'), barAction) // só passa a barAction se o utilizador tiver o role user'.
```

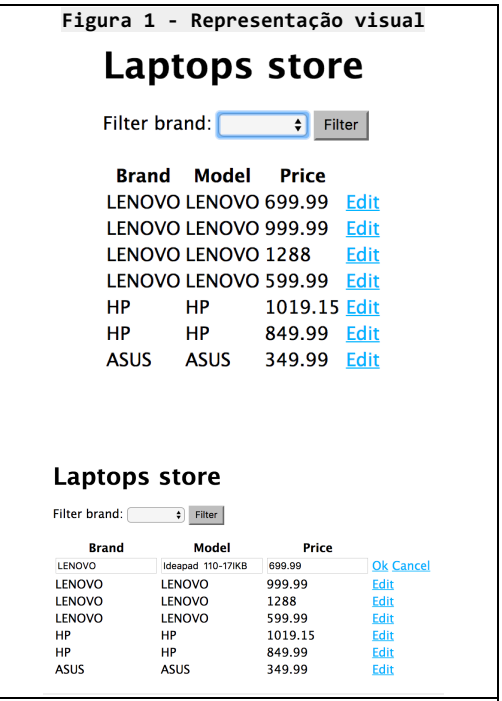
Grupo 2

1. [10] Considere a *view* na Listagem 1, pertencente a uma aplicação web de uma loja para venda de *laptops*. Esta *view* apresenta todos modelos disponíveis e permite também filtrar, pelos modelos de uma marca. É apresentada quando se acede à *path* `/laptops` da aplicação.

Listagem 1 - laptops.hbs

```
<style>.edit { display: none }</style>
<h1>{{title}}</h1>
<form method="GET" action=".">
  Filter brand: <select name="brand">
    <option value=""></option>
    {{#each brands}}
      <option>{{this}}</option>
    {{/each}}
  </select>
  <button type="submit">Filter</button>
</form>
<div class='panel'>
  <table class='table'>
    <thead>
      <tr><th>Brand</th><th>Model</th><th>Price</th><th> </th></tr>
    </thead>
    <tbody id='laptops'>
      {{#each laptops}}
      <tr class=view" id="view_{{id}}">
        <td id="view_brand_{{id}}">{{brand}}</td>
        <td id="view_model_{{id}}">{{brand}}</td>
        <td id="view_price_{{id}}">{{price}}</td>
        <td><a id="edit_cmd_{{id}}" class="edit-cmd"
href="">Edit</a></td>
      </tr>
     <!-- O tr seguinte apenas é necessário para as questões c e d -->
      <tr class="edit" id="edit_{{id}}">
        <td><input id="edit_brand_{{id}}" type="text"
value="{{brand}}"/></td>
        <td id="edit_model_{{id}}"><input type="text"
value="{{model}}"/></td>
        <td id="edit_price_{{id}}"><input type="text"
value="{{price}}"/></td>
        <td>
          <a id="ok_cmd_{{id}}" class="ok-cmd" href="">Ok</a>
          <a id="cancel_cmd_{{id}}" class="cancel-cmd"
href="">Cancel</a>
        </td>
      </tr>
      {{/each}}
    </tbody>
  </table>
</div>
```

Figura 1 - Representação visual



Listagem 2 - Funções para as questões c e d

```
function toglleView(hideId, showId) {
    let id = getId(this.id);
    document.getElementById(hideId)
          .style.display = "none";
    document.getElementById(showId)
         .style.display="table-row";
    return false;
}

function getId(str) {
 return str.substr(str.lastIndexOf("_")+1);
}
```

   a. [1,5] A aplicação suporta a listagem (com filtragem por marca) e edição de um *laptop*. Defina a interface do módulo `laptops-service`, que disponibiliza as funcionalidades necessárias para suportar estas operações.
   NOTA: Não tem que implementar o módulo. Apenas descrever a sintaxe e semântica do que o módulo exporta.
   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, com filtragem, e edição de um laptop. Use como base o módulo desenvolvido na alínea anterior e a *view* `tasks.hbs` fornecida.
   c. [3] Implemente o necessário em script de cliente, de modo que a página tenha o comportamento visual apresentado nas Figura 1. Quando se clica em `Edit`, a linha correspondente passa para modo de edição. Em modo de edição, o utilizador pode cancelar, ou gravar (ok). Em ambos os casos a linha volta para o modo de visualização. Não implemente o comportamento da ação de ok, pois tal será feito em d).
   NOTAS: Na implementação pode utilizar as funções da Listagem 2. Tem ainda que fazer o registo nos eventos necessários.
   d. [3,5] Implemente o comportamento necessário para a ação de Ok, de modo a realizar um pedido ao servidor para atualizar os dados do laptop editado e, caso a resposta indique sucesso, voltar a linha ao modo de visualização com os dados alterados. Tudo isto deve suceder sem um refrescamento total da página.

Os Docentes,
Luís Falcão e Miguel Carvalho