Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Programação na Internet
Semestre de Inverno de 2016/2017 - Época Especial (*4 de Março de 2017)* – Duração: 2h (+30 min tolerância)

Grupo 1

1. Justifique as respostas às seguintes questões:

   a. [1] Na implementação de um servidor HTTP com o módulo `express` o envio de um objecto `ctx` numa resposta através do método `render()` de `res`, i.e. `res.render(view_name, ctx)` é equivalente à execução de quais acções
   b. [1] Justifique porque é que a resposta a um POST deve ter sempre um código de estado HTTP de *redirect*?
   c. [1] Justifique se alguma das abordagens é **incorrecta**: A) Um URI para um recurso com duas representações diferentes; B) Dois URIs diferentes para o mesmo recurso com a mesma representação.
   d. [1] Justifique como deve proceder para submeter um formulário HTML através do método HTTP PUT?

2. [3] **Implemente** o módulo `autorouter` que permite adicionar a uma aplicação `express` (e.g. `app`) rotas paras as funções de um objecto (e.g. `service`) recebidos por parâmetro.
   Cada função tem um prefixo (e.g. `post_`) que indica o método HTTP dessa rota.
   Aos parâmetros de cada função são passados argumentos obtidos das propriedades do objecto `req`, em conformidade com o significado desse parâmetro na sua função, e.g. um parâmetro de uma função `get_` deve ser obtido da *query-string* (não considere parâmetros de rota). O último parâmetro é sempre o res.
   Admita a existência de um módulo que dada uma função retorna um *array* com os nomes dos parâmetros.

```
const express = require(express)
const app = express()

//...adicão de middlewares, incluindo autenticação via passport.

const service = require('./footApi')
const autorouter = require('./autorouter')
autorouter.bind(app, service)
```

```
// footApi.js

module.exports = {
   get_favorites = function(user, res) {
        ...
        res.render(...)
   }
   post_favorites = function(teamId, user, res) {
        ...
        res.redirect(...)
   }
}
```

1. [4] Implemente o módulo `async` com as seguintes funções.

a. [1,5] Implemente a função `compose(outer, inner)` que retorna uma nova função que é a composição das funções assíncronas recebidas por parâmetro. A função `outer` consome o valor produzido pela função `inner`.

b. [1,5] Implemente a função `waterfall(arg, funcs, cb)` que executa sequencialmente as funções do array `funcs`, onde cada função passa o seu resultado à função seguinte. Primeira função do array `funcs` é executada com o argumento `arg` passado a `waterfall`.

USE obrigatoriamente a função compose da alínea a) na resolução desta alínea.

c. [1] Caso tenha usado uma abordagem iterativa (e.g. `while`, `for`, `forEach`, etc) na alínea b), implemente uma nova versão de `waterfall` **recursiva e NÃO iterativa**.
   Dica: pode usar a função de array `slice(begin, end)` que retorna um novo array com os elementos desde o índice `begin` inclusive e `end` exclusive.

```
function inc(arg, cb) {
    //...
    console.log('inc(' + arg + ') --> ' + ++arg)
    cb(null, arg)
}
function square(arg, cb) {
    //...
    const res = arg * arg
    console.log('square(' + arg + ') --> ' + res)
    cb(null, res)
}
function dup(arg, cb) {
    //...
    const res = arg * 2
    console.log('dup(' + arg + ') --> ' + res)
    cb(null, res)
}
```

```
const async = require('./async')

const squareInc =
  async.compose(square, inc)

squareInc(2, (err, data) => {
  console.log(
    'square(inc(2)) --> ' + data)
})

async.waterfall(
  2,
  [inc, square, dup],
  (err, data) => {
    console.log(
     'dup(square(inc(2)))  =  '  +
data)
})
```

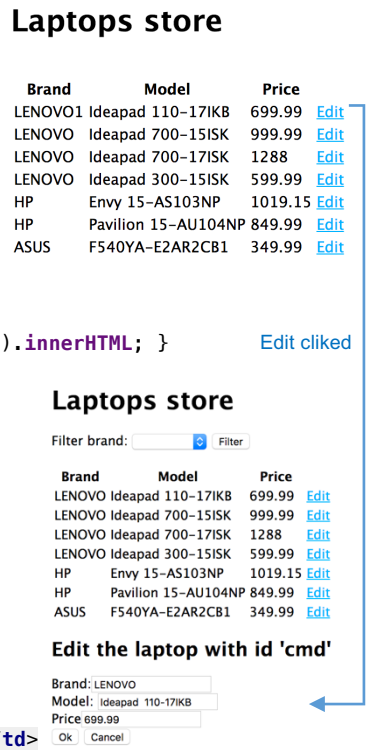
```
output:


inc(2) --> 3
square(3) --> 9
square(inc(2)) --> 9


inc(2) --> 3
square(3) --> 9
dup(9) --> 18
dup(square(inc(2))) = 18
```

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

```
1  <style>.edit { display: block } .view { display: none }</style>
2  <script>
3  window.onload = function () {
4     Array.prototype.forEach.call(document.getElementsByClassName("edit-cmd"), a => a.onclick = edit);
5     getElem("cancel_cmd").onclick = cancel; getElem("ok_cmd").onclick = ok;
6
7     const properties = ["brand","model", "price"]
8     function edit() {
9        let id = getId(this.id);
10       getElem("laptop_id").innerHTML = getElem("edit_id").value = id;
11       properties.forEach(p => setValue("edit_" + p, "view_" + p + "_" + id));
12       getElem("edit").className = "edit";
13       return false;
14    }
15    function cancel() { // TODO – Alínea c) }
16    function ok() {
17       let id = getElem("edit_id").value;
18       const laptop = { id: id};
19       properties.forEach(p => laptop[p] = getElem("edit_" + p).value);
20       update(laptop).then(updated);
21       // TODO – Alínea d)
22    }
23    function setValue(idInput, idElem) { getElem(idInput).value = getElem(idElem).innerHTML; }
24    function getId(str) { return str.substr(str.lastIndexOf("_")+1); }
25    function getElem(id) { return document.getElementById(id); }
26 }
27 </script>
28 <h1>{{title}}</h1>
29 <div class='panel'>
30     <table class='table'>
31         <thead><tr><th>Brand</th><th>Model</th><th>Price</th><th> </th>
32         </tr>
33         </thead>
34         <tbody id='laptops'>
35            {{#each laptops}}
36             <tr class=view" id="view_{{id}}">
37                 <td id="view_brand_{{id}}">{{brand}}</td>
38                 <td id="view_model_{{id}}">{{model}}</td>
39                 <td id="view_price_{{id}}">{{price}}</td>
40                 <td><a id="edit_cmd_{{id}}" class="edit-cmd" href="">Edit</a></td>
41             </tr>
42             {{/each}}
43         </tbody>
44     </table>
45     <!-- Este tr apenas é necessário para as questões -->
46     <div id="edit" class="view">
47         <h2>Edit the laptop with id '<span id="laptop_id" ></span>'</h2>
48         <input id="edit_id" type="hidden"/>
49         Brand:<input id="edit_brand" type="text" value="{{brand}}"/></br>
50         Model: <input id="edit_model" type="text" value="{{model}}"/></br>
51         Price<input id="edit_price" type="text" value="{{price}}"/></br>
52         <button id="ok_cmd" class="ok-cmd">Ok</button>
53         <button id="cancel_cmd" class="cancel-cmd">Cancel</button>
54   </div>
55 </div>
56
```



   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 e realize uma implementação em memória desse módulo.
   b. [1,5] 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. [2] Implemente a função cancel(), de modo que a página tenha o comportamento visual apresentado nas Figura 1. Quando se clica em `Edit`, aparece um formulário preenchido para a edição do laptop selecionado. Em modo de edição, o utilizador pode cancelar, ou gravar (ok). Em ambos os casos O formulário desaparece.
   d. [3] Implemente o que falta no método ok de Ok, de modo a realizar um pedido ao servidor para atualizar os dados do laptop editado, e caso a resposta indique sucesso, esconder o formulário e atualizar a linha editada da tabela.
   e. [2] Descreva o que fazem as linhas 4 e 5 da listagem.

Os Docentes,
Luís Falcão e Miguel Carvalho