

Grupo 1

1. [4] Justifique as respostas às seguintes questões:

- [1] Na implementação de um servidor HTTP com o módulo `express`, o envio de um objeto `ctx` numa resposta através do método `render()` de `res`, i.e. `res.render(view_name, ctx)` é equivalente à execução de quais ações?
- [1] Justifique porque é que a resposta a um POST deve ter sempre um código de estado HTTP de *redirect*, sempre que a submissão de um formulário HTML tem sucesso?
- [1] Justifique se alguma das abordagens é **incorreta**: A) Um URI para um recurso com duas representações diferentes; B) Dois URIs diferentes para o mesmo recurso com a mesma representação.
- [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 para as funções de um objeto (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 objeto `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.

<pre>const express = require('express') const app = express() //...adição de middlewares, incluindo autenticação via passport. const service = require('./footApi') const autorouter = require('./autorouter') autorouter.bind(app, service)</pre>	<pre>// footApi.js module.exports = { get_favorites = function(user, res) { ... res.render(...) } post_favorites = function(teamId, user, res) { ... res.redirect(...) } }</pre>
--	---

3. [4] Implemente o módulo `async` com as seguintes funções.

- [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`.
- [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. A 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.

- [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`, exclusivo.

<pre>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) }</pre>	<pre>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) })</pre>	<p>output:</p> <p><code>inc(2) --> 3</code> <code>square(3) --> 9</code> <code>square(inc(2)) --> 9</code></p> <p><code>inc(2) --> 3</code> <code>square(3) --> 9</code> <code>dup(9) --> 18</code> <code>dup(square(inc(2))) = 18</code></p>
--	--	---

Grupo 2

4. [9] 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
3 <script>
4 window.onload = function () {
5   Array.prototype.forEach.call(document.getElementsByClassName("edit-cmd"), a => a.onclick = edit);
6   getElem("cancel_cmd").onclick = cancel; getElem("ok_cmd").onclick = ok;
7
8   const properties = ["brand", "model", "price"]
9
10  function edit() {
11    let id = getId(this.id);
12    getElem("laptop_id").innerHTML = getElem("edit_id").value = id;
13    properties.forEach(p => setValue("edit_" + p, "view_" + p + "_" + id));
14    getElem("edit").className = "edit";
15    return false;
16  }
17
18  function cancel() { // TODO - Alínea c) }
19
20  function ok() {
21    let id = getElem("edit_id").value;
22    const laptop = { id: id };
23    properties.forEach(p => laptop[p] = getElem("edit_" + p).value);
24    update(laptop).then(updated);
25    // TODO - Alínea d)
26  }
27
28  function setValue(idInput, idElem) { getElem(idInput).value = getElem(idElem).innerHTML; }
29
30  function getId(str) { return str.substr(str.lastIndexOf("_")+1); }
31
32  function getElem(id) { return document.getElementById(id); }
33
34 }
35
36 </script>
37
38 <h1>{{title}}</h1>
39
40 <div class="panel">
41   <table class="table">
42     <thead><tr><th>Brand</th><th>Model</th><th>Price</th><th>&nbsp;</th>
43     </tr>
44     </thead>
45     <tbody id="laptops">
46       {{#each laptops}}
47       <tr class="view" id="view_{{id}}">
48         <td id="view_brand_{{id}}">{{brand}}</td>
49         <td id="view_model_{{id}}">{{model}}</td>
50         <td id="view_price_{{id}}">{{price}}</td>
51         <td><a id="edit_cmd_{{id}}" class="edit-cmd" href="">Edit</a></td>
52       </tr>
53       {{/each}}
54     </tbody>
55   </table>
56
57   <!-- Este tr apenas é necessário para as questões -->
58
59   <div id="edit" class="view">
60     <h2>Edit the laptop with id '<span id="laptop_id"></span>'</h2>
61     <input id="edit_id" type="hidden"/>
```

Laptops store

Brand	Model	Price	
LENOVO	Ideapad 110-17IKB	699.99	Edit
LENOVO	Ideapad 700-15ISK	999.99	Edit
LENOVO	Ideapad 700-17ISK	1288	Edit
LENOVO	Ideapad 300-15ISK	599.99	Edit
HP	Envy 15-AS103NP	1019.15	Edit
HP	Pavilion 15-AU104NP	849.99	Edit
ASUS	F540YA-E2AR2CB1	349.99	Edit

Edit clicked

Laptops store

Filter brand: [Filter](#)

Brand	Model	Price	
LENOVO	Ideapad 110-17IKB	699.99	Edit
LENOVO	Ideapad 700-15ISK	999.99	Edit
LENOVO	Ideapad 700-17ISK	1288	Edit
LENOVO	Ideapad 300-15ISK	599.99	Edit
HP	Envy 15-AS103NP	1019.15	Edit
HP	Pavilion 15-AU104NP	849.99	Edit
ASUS	F540YA-E2AR2CB1	349.99	Edit

Edit the laptop with id 'cmd'

Brand: LENOVO
Model: Ideapad 110-17IKB
Price 699.99

```

Brand:<input id="edit_brand" type="text" value="{{brand}}"/></br>
Model: <input id="edit_model" type="text" value="{{model}}"/></br>
Price<input id="edit_price" type="text" value="{{price}}"/></br>
<button id="ok_cmd" class="ok-cmd">Ok</button>
<button id="cancel_cmd" class="cancel-cmd">Cancel</button>
</div>
</div>

```

- [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.
- [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 laptops.hbs` fornecida.
- [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.
- [2] 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.
- [2] Descreva o que fazem as linhas 4 e 5 da listagem.

Os Docentes,
Luís Falcão, Paulo Pereira e Miguel Carvalho