Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Programação na Internet
Inverno de 2017/2018
1ª Época (*19 de Fevereiro de 2018)* – Duração: 2h30

Número: _____ Nome:_____

Grupo 1 [6 valores]

Uma resposta assinalada corretamente conta 0,5 valores, incorrectamente desconta 0,25 valores ao total do grupo, sem resposta conta 0 valores.

1. [2] No processo de autenticação de uma dada aplicação web o envio das credenciais do utilizador é realizada através da submissão de um formulário HTML num pedido HTTP POST. O projectista da componente que atende este pedido pode optar pelas seguintes respostas:

   - ❑ `200 OK` para credenciais válidas e `403 Forbidden` para inválidas.
   - ❑ `200 OK` em ambos os casos
   - ❑ `303 See Other` para credenciais válidas e `403 Forbidden` para inválidas.
   - ❑ `303 See Other` para ambos os casos

2. [2] Para a seguinte definição do módulo `wacky` assinale com V as utilizações que executam sem erros e com F as utilizações que dão erro de execução.

```
const wacky = nr => console.log(nr/3)
wacky.dup =  nr => console.log(nr*2)
module.exports = wacky
```

   - ❑ `require('./wacky')(7).dup(3)`
   - ❑ `require('./wacky')(7)`
   - ❑ `require('./wacky')(7)(3)`
   - ❑ `require('./wacky').dup(3)`

2. [2] Considere a seguintes regras de estilo e classifique as afirmações que se seguem de verdadeiras ou falsas.

```
.table { border-collapse: collapse; }
th, td { border-bottom: 1px solid #ddd; }
tr.even { background-color: #f2f2f2;}
tr td.first, tr th.first { width: 60%; }
```

   - ❑ A 1ª regra aplica-se a todas as tabelas.
   - ❑ A 2ª regra aplica-se a elementos th e td.
   - ❑ A 3ª regra aplica-se a elementos tr em posição par.
   - ❑ A 4ª regra aplica-se a todos elementos tr, td e th.

Grupo 2 [7 valores]

3. [2] Considere o pedido HTTP e a correspondente resposta, na Listagem 1. O código que processou o pedido encontra-se na Listagem 2. Pretende-se que a resposta contenha o valor da chave 'a' presente no pedido. Identifique os problemas que encontra, quer provoquem erros funcionais ou não, e proponha as respetivas correções.

Listagem 1

```
POST 3000/tasks HTTP/1.1
Accept: */*
Host: localhost:3000
Content-Length: 7

a=value
-------------------------------------------------
-----------
HTTP/1.1 200
Date: Wed, 17 Jan 2018 23:14:41 GMT
Content-Type: text/html

a = undefined
```

Listagem 2

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
app.use(bodyParser.urlencoded());
app.post('/tasks', function (req, rsp) {
   rsp.end(`a = ${req.body.a}`);
});
```

4. [3] Implemente o módulo restMw que exporta uma função com a assinatura: function(method, fn), que constrói um *middleware* express, com base nas funções passadas por parâmetro ao método add (**não pode usar o tipo Router do express**).

   Assuma que as funções adicionadas (e.g. foo, bar, ou outras) recebem um único parâmetro, que é uma função que obedece à convenção de chamada com *callback* do node.js.

```
function foo(cb){ cb(null, 'I am foo')}
function bar(cb){ cb(null, 'I am bar')}
function zaz(cb){ cb(new Error('Bum'))}
```

```
const app =  require('express')()
const restMw = require('./restMw')
const router = restMw('put', foo).add('get', zaz).add('get', bar)
app.use(router)
app.use((req, res) => res.status(404).send('Not Found'))
app.use((err, req, res, next) => res.status(500).send(err.message))
app.listen(3000)
```

Exemplos:

Pedido HTTP GET /bar tem uma resposta com *body* 'I am bar' e *status code* 200
Pedido HTTP GET /foo tem uma resposta com *body* 'Not Found' e *status code* 404
Pedido HTTP PUT /foo tem uma resposta com *body* 'I am foo' e *status code* 200
Pedido HTTP GET /xpto tem uma resposta com *body* 'Not Found' e *status code* 404
Pedido HTTP GET /zaz tem uma resposta com *body* 'Bum' e *status code* 500

   O *path* para cada função corresponde ao nome da função e ao método HTTP especificado, não sendo portanto suportadas funções anónimas.
   Exemplo: …`add('get', zaz)` adiciona uma rota para o path `'/zaz'` e o método HTTP `GET`.
   Em caso de sucesso o resultado da função é retornado em **JSON** como resposta ao pedido HTTP.
   Se um pedido HTTP não for respondido por nenhuma das funções do *middleware* construído por `restMw` então o atendimento é passado ao próximo *middleware* da aplicação `express`.

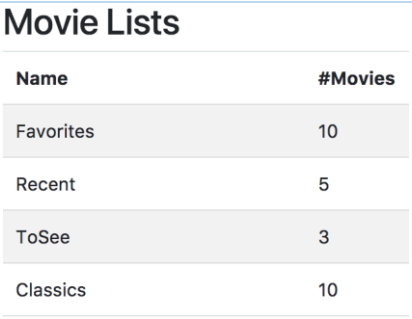
5. [2] Implemente o módulo `memMw` que retorna um *middleware* express que memoriza a resposta dada aos pedidos HTTP que retornam conteúdos em JSON, como é o caso do *middleware* construído por `restMw`.

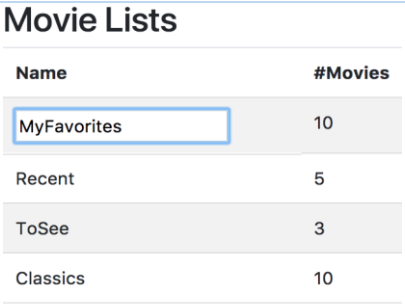
   Pedidos futuros para o mesmo *path* deverão retornar a resposta memorizada, sem que o pedido HTTP chegue ao *middleware* que o produziu num pedido anterior.
   O middleware dado `memMw` deve ser adicionado a uma aplicação `Express` antes de qualquer outro que produza respostas HTTP no formato JSON e que se queiram memorizar.

Grupo 3 [8 valores]

6. [8] Considere a funcionalidade de listas de filmes desenvolvida no trabalho prático. A *path* do URI para acesso às listas de filmes de um utilizador é: /movies/lists. A listagem lists.hbs contém o código da view que apresenta as listas de filmes de um utilizador.

```
<h2>Movie Lists</h2>
<table class="table">
   <tr>
       <th>Name</th>
       <th>#Movies</th>
   </tr>
   {{#each lists}}
   <tr>
       <td><span>{{name}}</span></td>
       <td>{{numMovies}}</td>
   </tr>
   {{/each}}
</table>
```





a. [2,5] Implemente em Node.js o endpoint que apresenta esta *view*. Assuma que existe uma variável global app, com a instância da aplicação Express. Assuma também que existe um módulo (a DAL da solução) que exporta um objecto que contém um método que retorna todas as listas do utilizador actual. Descreva sucintamente a assinatura e o comportamento desse método.

b. [2.5] Pretende-se que, quando se clica no nome de uma lista, a UI fique em modo de edição e seja possível alterar o nome da lista clicada, conforme ilustrado nas figuras acima. O modo de edição é abandonado se o utilizador clicar em qualquer outro local da página. Considere a implementação desta funcionalidade apresentada na listagem seguinte. Faça as alterações necessárias à view para que a UI suporte o modo de edição conforme descrito.

```
(function() {
   const forEach = Array.prototype.forEach.call;
   forEach(document.getElementsByClassName("view"), e => { e.onclick = edit; });
   forEach(document.getElementsByClassName("edit"), e => { e.onclick =(e)=> e.stopPropagation()});
   let listInEditMode = null; let originalName;
   function edit(event) {
       changeVisibility(this.id, "none", "inline")
       originalName = this.innerText; listInEditMode = this;
       event.stopPropagation();
   }
   document.onclick = function () {
       if(listInEditMode) {
           changeVisibility(listInEditMode.id, "inline", "none")
           updateValue(listInEditMode, originalName); listInEditMode = null;
       }
   }
   function changeVisibility(listId, viewVisibility, editVisibility) {
       console.log(listId)
       document.getElementById(`${listId}_edit`).style.display = editVisibility;
       document.getElementById(`${listId}`).style.display = viewVisibility;
   }
   function updateValue(listInEditMode, prevName) {
       let currentValue = document.getElementById(`${listInEditMode.id}_edit`).value;
       if(currentValue != prevName) {
           listInEditMode.innerText = currentValue;
           updateListNameInServer(listInEditMode.id, currentValue, originalName)
       }
   }
   function updateListNameInServer(listId, currName, prevName) { /* TODO */ }
})()
```

c. [3] Implemente a função updateListNameInServer que atualiza o nome de lista no servidor. Defina também o endpoint HTTP que recebe este pedido, indicando a path do URI, o método HTTP e o conteúdo do corpo do pedido.