Laboratório Nº 1: Revisões JavaScript e Desenvolvimento em Node JS

Programação e Integração de Serviços
2020/2021

Prof. Filipe Mariano

Objetivos

- Utilização do **Visual Studio Code** como IDE para desenvolvimento em JavaScript/Node JS
- Desenvolvimento em Node JS
- Criação de módulos em Node JS
- Criação de Classes em JavaScript

1. Instalação e Configuração do Ambiente de Desenvolvimento

Como editor de texto para será utilizado o editor multiplataforma Visual Studio Code (VSCode), algumas das vantagens da sua utilização podem ser percebidas em http://code.visualstudio.com/docs/editor/whyvscode.

O VSCode encontra-se disponível para download para diversos sistemas operativos em http://code.visualstudio.com/Download.

Para se desenvolver em JavaScript um programa que irá ser executado no servidor necessitamos do ambiente Node.js. Este é disponibilizado, tal como o VSCode, para multiplataforma e podemos fazer o seu download em https://nodejs.org/en/download (convém fazer o download da versão LTS - Long Term Support, escolhendo, naturalmente, a versão de sistema operativo desejada).

2. Iniciar um Projeto em Node JS, Instalação e Utilização de Módulos

2.1. Iniciar Projeto

O Node JS assenta numa abordagem de programação e gestão de eventos de forma assíncrona, na tecnologia de JavaScript e que permite a criação de aplicações de servidor numa linguagem outrora conhecida pelo seu desenvolvimento web (apenas) na perspetiva do cliente. Possui um sistema de gestão de módulos (NPM), que fomenta a reutilização de código através de módulos totalmente grátis, sendo esta uma das principais razões que levou ao crescimento exponencial do Node nos seus poucos anos de existência, principalmente no âmbito da programação para a internet.

No desenvolvimento de aplicações web (e não só) é comum a utilização de ficheiros de configuração que permitem descrever como o website se irá comportar, os módulos que tem associados e suas versões, assim como outros aspetos dependentes da tecnologia utilizada na implementação do mesmo. O Node não foge à regra e dispõe de um ficheiro de configuração essencial para os seus projetos, denominado `package.json`. É um ficheiro em JSON que deverá conter alguns dados acerca do projeto, tais como o nome, descrição, autor, etc, como também deverá conter outros dados referentes às dependências que o projeto possui. As dependências dizem respeito aos módulos que o projeto está a utilizar e que serão fundamentais para tornar o que for desenvolvido reutilizável e/ou partilhado por outros programadores.

Para criar o ficheiro de configuração do projeto deve-se executar, no terminal do Visual Studio Code (aceder através de View > Integrated Terminal ou do atalho CTRL + Ç), o seguinte comando:

```
npm init
```

No terminal serão colocadas algumas perguntas de *default* que permitirá criar o ficheiro de configuração já com algumas das informações essenciais. Se pretender que seja criado o ficheiro de configuração de *default* e mais tarde alterá-lo, também pode fazê-lo através de:

```
npm init --yes
```

Os valores criados serão:

- **name**: a diretoria do projeto (obrigatório)
- **version**: 1.0.0 (obrigatório)
- **description**: string vazia
- **main**: index.js (o ficheiro que irá ser executado quando se executar o projeto)
- **scripts**: por *default* é criado um script de teste vazio
- **keywords**: vazio
- **author**: vazio
- **license**: ISC

2.2. Instalação de Módulos

Sabendo como iniciar um projeto em Node falta conhecer como se pode usufruir da panóplia de bibliotecas existentes, possibilitando um desenvolvimento mais rápido recorrendo a módulos já desenvolvidos. Por omissão o Node JS tem disponíveis um conjunto de módulos que implementam funcionalidades genéricas (http://nodejs.org/api/). Contudo, existe um repositório público que permite a publicação de módulos desenvolvidos por terceiros para utilização comum e universal (https://www.npmjs.com/), sendo necessário efetuar uma instalação local dos mesmos. A instalação dos módulos é feita utilizando o seguinte comando:

```
npm install <nome_do_modulo> --save
```

A colocação do parâmetro `--save` não é obrigatório mas faz com que seja criada uma dependência no projeto com o módulo instalado, sendo esta registada no ficheiro de configuração `package.json` (segue as regras do sistema de versionamento semântico), beneficiando a portabilidade do projeto e a partilha do mesmo com outros programadores. A instalação de um módulo faz com que o mesmo seja colocado numa nova diretoria criada no projeto, cujo nome é `node_modules`.

2.3. Utilização de Módulos

Para a utilização de módulos no Node é utilizado a função `require` com o nome do módulo como parametro, tal como exemplifica o seguinte excerto de código:

```
const http = require('http');

http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.end('Hello World!');
}).listen(8080);
```

No entanto, também existe a abordagem de o módulo estar incluído numa diretoria (módulo local), o que ocorre diversas vezes quando se cria os próprios módulos incorporados no próprio projeto. Neste caso, a definição do módulo é feita através da criação de um ficheiro onde se utiliza código JavaScript tradicional e que quando se pretende dar acesso externamente a algo (função, objeto, constante, etc.) utiliza-se o objeto `module.exports`. A este objeto podem ser acrescentadas propriedades correspondentes às funcionalidades que se pretendem exportar ou pode se substituir, por completo, o objeto exports com o conteúdo do que se pretende exportar. Observe os seguintes exemplos:

Exemplo de exportação de várias propriedades:

```
//Conteúdo do ficheiro circulo.js (exportar várias propriedades):
const PI = 3.14159;

function area(raio) {
    return PI * raio * raio;
}

function perimetro(raio) {
    return PI * 2 * raio;
}

module.exports.area = area;
module.exports.perimetro = perimetro;
```

```
//Conteúdo presente no ficheiro utilizaCirculo.js para utilizar o módulo criado
const circulo = require("./circulo.js"); //o .js pode ser omitido
console.log(circulo.area(10));
console.log(circulo.perimetro(5));
console.log(circulo.PI); //undefined
```

Outro exemplo de exportação:

```
//Conteúdo do ficheiro log.js (todo o objeto exports é substituído pela função):

module.exports = function (mensagem) {
    console.log(mensagem);
};
```

```
const msg = require("./log.js");
msg("Hello World!");
```

Os módulos criados também poderão posteriormente ser publicados no **npm**, e recorrendo ao comando `npm install` poderão ser instalados nos projetos que se pretender.

3. Utilização do Terminal - REPL

O REPL significa "Read Eval Print Loop" e representa um ambiente onde um comando é inserido e o sistema responde com a avaliação do comando introduzido. O Node já vem incluído com um ambiente REPL, sendo bastante útil para experimentar rapidamente código Node.js ou para *debug* de códigos JavaScript.

Para utilizar este terminal no Visual Studio Code, basta aceder através de **View > Integrated Terminal** ou pelo atalho **CTRL + Ç**. Ao introduzir o comando `node` está a iniciar o ambiente REPL incorporado no Node.

Exemplo de utilização do REPL para avaliação de expressões JavaScript: (cada linha que contém o `>`, no final foi pressionada a tecla *Enter*)

```
PS D:\DBM\Lab1>node
> 1 + 1
2
> const m = 2
undefined
> m + 1
3
```

Exemplo de utilização do REPL para executar ficheiros Node JS:

```
PS D:\DBM\Lab1>node utilizaCirculo.js
314.159
31.4159
undefined
```

4. Callbacks

O Node está assente num paradigma de eventos que são despoletados e geridos de forma assíncrona. Cada vez que há um evento executa-se o código a ele associado (*event handler*), passando-se de imediato ao tratamento do evento seguinte. A execução do *event handler* deve ser o mais rápida possível para se avançar para o evento seguinte, permitindo a resposta a inúmeras solicitações "simultâneas". Devem ser evitadas execuções de quaisquer tarefas que ocupem demoradamente o processamento do sistema. Estas devem ser substituídas por chamadas assíncronas, passando como argumento funções (*callback*) que serão executadas, no final, com o resultado do processamento.

Exemplo:

```
function exemplo(val, callback){
    if(val == 1){
        callback(true);
    }else{
        callback(false);
    }
}

exemplo(0, function (bool){
                if(bool){
                    alert('Verdadeiro');
                }else {
                    alert('Falso');
                }
});
```

Também se poderia fazer com que a função exemplo fosse uma função de *callback* se a associarmos um evento de clique:

```
document.body.addEventListener('click', exemplo);
```

Utilizando um dos módulos já incorporados no Node (módulo `fs`) que permite operações de leitura e escrita em ficheiro, é possível demonstrar facilmente a diferença numa abordagem síncrona e assíncrona.

Síncrona

```
const fs = require('fs'); //como é um módulo core do Node não requer instalação
let content = fs.readFileSync("readme.txt");
console.log(content);
console.log('A ler o ficheiro...');
```

Assíncrona

```
const fs = require('fs'); //como é um módulo core do Node não requer instalação
fs.readFile("readme.txt", function(err, content) {
                            if (err) {
                                console.log(err);
                            }
                            console.log(content);
});
console.log('A ler o ficheiro...');
```

A diferença entre ambos os excertos de código é que enquanto no primeiro a execução do código é bloqueada na linha do `readFileSync`, na segunda o console.log "A ler o ficheiro..." surge primeiro que o

conteúdo do ficheiro lido, sendo que quando o ficheiro é lido é despoletada a função de *callback* que tinha sido passada como argumento.

5. Classes e Objetos em JavaScript

5.1. Síntaxe

Classe implementada através do Prototype

```
function Pessoa (nome) {
    this.nome = nome;
}

Pessoa.prototype.ola = function () {
    alert(this.nome);
}
```

Classe implementada em ES6

```
class Pessoa {
    constructor(nome) {
        this.nome = nome;
    }

    ola () {
        alert(this.nome);
    }
}
```

5.2. Criação de Objetos

Inicialização de um objeto do tipo Pessoa

```
let pessoa = new Pessoa('Filipe Mariano');
console.log(pessoa.nome);
```

Inicialização de um objeto com duas propriedades (nome, versao) e um método na notação literal

```
let proj = {
    nome: "Projeto Exemplo",
    versao: "1.0.0",
    descricao: function() {
        return `${this.nome} na versão ${this.versao}`;
    }
}
console.log(proj.descricao());
```

6. Exercícios

1. Crie uma pasta no seu computador chamada `Lab1` para guardar os exercícios que for desenvolvendo e inicie o projeto de forma a criar o ficheiro de configuração `package.json`. Crie também um ficheiro `index.js` (por enquanto vazio).

2. Na secção 2 existe um exemplo de um módulo `circulo` e de um módulo `mensagem` com os respetivos códigos. É pretendido que seja criado o `circulo.js` e `mensagem.js` colando os códigos propostos e criando um novo ficheiro `index.js` que deverá importar ambos os módulos e executá-los.

3. O objetivo deste exercício é escrever o código fonte de uma página html através de um *request* feito a uma determinada página. Requer a utilização dos módulos `request` (requer instalação) e `fs`.
a. Instale o módulo `request` (https://www.npmjs.com/package/request) pelo comando do **npm install** de forma a criar a dependência no ficheiro de configuração.
b. Utilize o bloco de código num ficheiro `pedido.js` para recorrer ao módulo `native-request` e na função de *callback* deverá escrever o código para colocar o body num ficheiro chamado `pagina.html`, criado através do módulo `fs` (utilizando a função writeFile ou writeFileSync).

```
const request = require('native-request');
//importar módulo native-request
request.get('http://www.google.com', function(err, data, status, headers) {
    if (err) {
        console.log(err);
    }

    //chamar a função writeFile ou writeFileSync para escrever em ficheiro
});
```

c. Experimente o código implementado através do terminal executando o ficheiro `index.js` (necessário importar o módulo `pedido.js`).

4. Crie um novo ficheiro `arrays.js` e crie um array com 5 elementos ("vermelho","amarelo","azul","verde","preto").
a. Faça uma função `concatenarElementos` que escreva numa string todos os elementos do array separado por `,`. Utilize a função `join` da classe `Array`.
b. Faça uma função `arrayEmMaiusculas` que devolve o mesmo array com todos os seus valores escritos em maiúsculas. Utilize a função `toUpperCase` da `string`.
c. Escreva uma função `verificaNumeros` que recebe um array e devolve um array com o mesmo número de elementos que a função recebeu, indicando se cada elemento é ou não um número. Utilize a função `isNaN` para testar cada elemento do array.
d. Teste as funções desenvolvidas no terminal executando o ficheiro `index.js`.

5. Criar a classe `Aluno` (no ficheiro `aluno.js`) com as propriedades `numero`, `nome`, `morada` e respetivo construtor. Crie um método com o nome `descritivo` que devolve uma string "Nome: <nome_aluno>;

Número: <numero_aluno>".

6. Importar a classe Aluno para o ficheiro `index.js`.
a. Criar uma variável `alunos` que seja um array com 5 objetos do tipo `Aluno`.
b. Escrever na consola o `descritivo` de todos os alunos.