

## Aula 1 - Criando o projeto Node

### Docupedia Export

Author:Ferro Alisson (CtP/ETS)

Date:15-Aug-2023 16:45

## Table of Contents

<b>1 O que é o Node.js?</b>	<b>3</b>
<b>2 Como funciona?</b>	<b>4</b>
<b>3 Características</b>	<b>5</b>
<b>4 Node.js Module</b>	<b>6</b>
<b>5 Objeto exports</b>	<b>7</b>
<b>6 Iniciando o projeto</b>	<b>12</b>
<b>7 NPM - Node Packet Manager</b>	<b>17</b>

# 1 O que é o Node.js?

De acordo com sua definição oficial, o Node é um runtime, que nada mais é do que um conjunto de códigos, API's, ou seja, são bibliotecas responsáveis pelo tempo de execução (é o que faz o seu programa rodar) que funciona como um interpretador de JavaScript fora do ambiente do navegador web.

É importante frisar que o Node.JS é um ambiente de execução assíncrono, isto é, ele trabalha de modo a não bloquear no momento da execução da aplicação, delegando os processos demorados a um segundo plano.



## 2 Como funciona?

O Node é capaz de interpretar um código JavaScript, igual ao que o navegador faz. Sendo assim, quando o navegador recebe um comando em JavaScript, ele o interpreta e depois executa as instruções fornecidas. Ele torna possível o envio de instruções (os nossos códigos) sem precisar de um navegador ativo, basta ter o Node.JS instalado e utilizar o terminal para executar um programa construído em JavaScript.

Além disso, você pode utilizar apenas uma linguagem de programação para tratar requisições entre cliente e servidor.

### 3 Características

- **Multiplataforma:** permite criar desde aplicativos desktop, aplicativos móveis e até sites SaaS;
- **Multi-paradigma:** é possível programar em diferentes paradigmas, como: Orientado a Objetos, funcional, imperativo e dirigido à eventos;
- **Open Source:** é uma plataforma de código aberto, isso significa que você pode ter acesso ao código fonte do Node.JS e realizar suas próprias customizações ou mesmo contribuir para a comunidade de forma direta;
- **Escalável:** Node.JS foi criado para construir aplicações web escaláveis, como podemos ver na sua [documentação oficial](#).

## 4 Node.js *Module*

Um *module* é uma coleção de funções e objetos do JavaScript que podem ser utilizados por aplicativos externos. Descrever um trecho de código como um módulo se refere menos ao que o código é do que aquilo que ele faz — qualquer arquivo Node.js pode ser considerado um módulo caso suas funções e dados sejam feitos para programas externos.

## 5 Objeto *exports*

Utilizando o objeto *exports* dentro do módulo é possível exportar e importar uma informação entre módulos como nos exemplos abaixo, utilizando variáveis e funções:

### Exportando variável

```
1  const nome = "João";
2  const sobrenome = "Siqueira";
3
4  const nomeInteiro = () => {
5      console.log(nome, sobrenome);
6  };
7
8  module.exports.nome = nome;
9  module.exports.sobrenome = sobrenome;
10 module.exports.nomeInteiro = nomeInteiro;
11 console.log(module)
```

A informação exportada pode ser recebida em outro arquivo pelo código abaixo:

### Importando

```
1  const mod1 = require('./mod1');
2  console.log(mod1.nome);
3  console.log(mod1.sobrenome);
4  mod1.nomeInteiro();
5
6  //const {nome, sobrenome, nomeInteiro}= require('./mod1')
7  //console.log(nome)
8  //console.log(sobrenome)
9  //nomeInteiro()
```

```
[Running] node "c:\Users\siq9ct\Documents\Node\tempCodeRunnerFile.js"
Module {
  id: '.',
  path: 'c:\\Users\\siq9ct\\Documents\\Node',
  exports: {
    nome: 'João',
    sobrenome: 'Siqueira',
    nomeInteiro: [Function: nomeInteiro]
  },
  filename: 'c:\\Users\\siq9ct\\Documents\\Node\\tempCodeRunnerFile.js',
  loaded: false,
  children: [],
  paths: [
    'c:\\Users\\siq9ct\\Documents\\Node\\node_modules',
    'c:\\Users\\siq9ct\\Documents\\node_modules',
    'c:\\Users\\siq9ct\\node_modules',
    'c:\\Users\\node_modules',
    'c:\\node_modules'
  ]
}
```

```
[Running] node "c:\Users\siq9ct\Documents\Node\app.js"
João
Siqueira
João Siqueira
```

Como estamos utilizando o módulo padrão *module* podemos referenciá-lo utilizando apenas *exports* e também utilizar a palavra reservada *this* para substituir ambos os termos.

Entretanto, isso não nos permite criar diretamente um novo objeto com todos os dados a serem exportados, como pode ser visto nos exemplos abaixo:



```
1 const nome = "João";
2 const sobrenome = "Siqueira";
3
4 //module.exports = {
5 //  nome, sobrenome
6 //};
7
8 //This.qualquerCoisa = "Qualquer coisa"
9
10 exports.outraCoisa = "Outra coisa"
```

Também é possível exportar e importar classes:

```
1 class Pessoa{
2   constructor(nome){
3     this.nome = nome;
4   }
5 }
6
7 exports.Pessoa = Pessoa
```

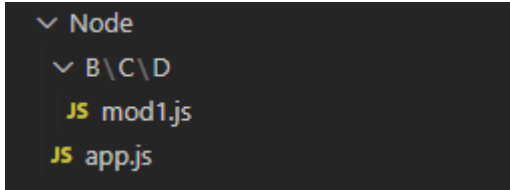
```
1 const { Pessoa } = require('./mod1');
2 console.log(Pessoa);
3
4 const p1 = new Pessoa('João');
5 console.log(p1);
```

Assim como funções também:

```
1 module.exports = function(x,y) {
2   return x * y;
3 };
```

```
1 const multiplicacao = require('./mod1');
2
3 console.log(multiplicacao(2,2));
```

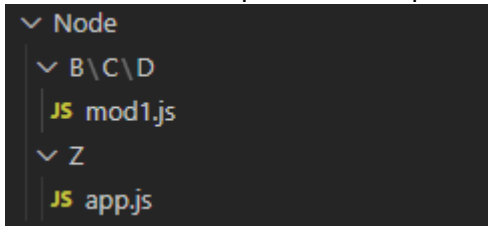
É importante sempre nos atentarmos ao diretório do arquivo que estamos requerindo:



Caso ele esteja dentro de várias pastas deveremos endereçá-lo corretamente.

```
1 const multiplicacao = require('./B/C/D/mod1');
2
3 console.log(multiplicacao(2,2));
```

Para voltar em uma pasta anterior podemos utilizar "../", como no caso abaixo:



```
1 const multiplicacao = require('../B/C/D/mod1');
2
3 console.log(multiplicacao(2,2));
4
5 // ./ Pasta atual
6 // ../ Pasta anterior
```

Para não nos confundirmos podemos usar os comandos que nos retornam o caminho do diretório ou arquivo em questão. Também podemos usar o diretório atual para referenciar algum outro.

```
1 console.log(__filename);
2 console.log(__dirname);
3
4 const path = require('path')
5 console.log(path.resolve(__dirname));
6
7 console.log(path.resolve(__dirname, "..", ".."));
8 console.log(path.resolve(__dirname, "..", "..", "Downloads"));
```



## 6 Iniciando o projeto

Para iniciar o projeto em node basta criarmos uma pasta e rodar o comando "npm init", o npm fará algumas perguntas para usar default em todas basta adicionarmos a *flag -y* no final do comando,então o npm criará o package.json, um arquivo de configuração importante.

```
{
  "name": "mongodb",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Aqui podemos ver algumas informações importantes, como o nome, a versão, os scripts e o ponto de entrada da API.

Antes de iniciar a API, vamos instalar algumas dependências importantes para continuar, para isso vamos rodar o seguinte comando

```
npm i body-parser config express nodemon
```

Após a instalação dos pacotes, precisamos adicionar um script no *package.json*, vamos colocar o seguinte script logo após o script de test que já tem

```
"start": "nodemon index.js"
```

agora sim podemos rodar npm start

Neste momento ainda nada será exibido pois não possui nada no arquivo index.js

Então vamos configurar o index.js

Para configurar vamos importar o express e chamar o express em uma variável, logo em seguida vamos definir a porta que irá rodar e por fim rodar a porta e exportar o servidor, portanto o código ficará semelhante ao seguinte

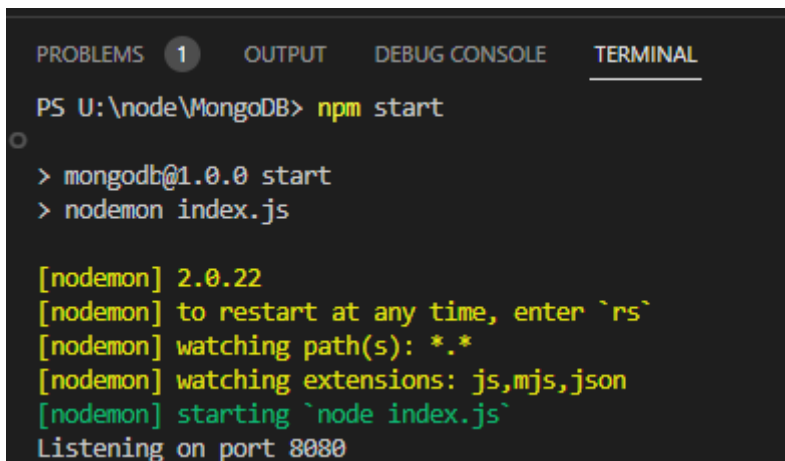
```
const express = require('express');
const router = require('./routes');
const app = express();

router(app);

const port = 8080;
const server = app.listen(port, () => console.log(`Listening on port ${port}`));

module.exports = server;
```

Com isso, ao subir a aplicação, no console irá ter uma mensagem semelhante a seguinte



```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
PS U:\node\MongoDB> npm start
> mongod@1.0.0 start
> nodemon index.js

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Listening on port 8080
```

Se receber essa mensagem significa que está rodando a API neste caso na porta 8080 que foi a que definimos. Para começar as rotas, vamos criar o arquivo routes/index.js com isso vamos configura-lo. Em routes/index.js vamos importar body-parser e exportar o módulo

```
const bodyParser = require('body-parser');
const person = require('./person');

module.exports = (app) => {
  app.use(
    bodyParser.json(),
    person
  )
}
```

Agora temos configurada o ponto de acesso das rotas, vamos configurar nossa rota. Vamos criar um arquivo em *routes/person.js*, onde vamos colocar nosso primeiro código.

```
const express = require('express');
const router = express.Router();

router
  .get('/api/person/first', (req, res) => {
    console.log("Hello in console");
    return
  })

module.exports = router
```

Observe que temos nosso `router.get` possui dois parâmetros, o primeiro parâmetro é o endpoint da API, o segundo parâmetro é uma função *callback* onde é chamada toda vez que o endpoint for acessado.

Se fomos no navegador e digitar o endpoint que configuramos `localhost:8080/api/person` e pressionar enter, no navegador nenhuma mensagem será exibida, porém no console da nossa API será exibida uma mensagem,

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

> mongod@1.0.0 start
> nodemon index.js

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Listening on port 8080
Hello in console
█
```

Ou seja, o endpoint foi acessado e executou nossa função callback uma vez, pois foi acessado somente uma vez, se recarregarmos o navegador outro "Hello in console" será exibido. Com isso temos nossa primeira API rodando e respondendo a requisições. Podemos também, realizar cálculos nas nossas requisições. Por exemplo:

```
router
  .get('/api/person/first', (req, res) => {
    console.log(8+5);
    return
  })
```

E ao acessar o endpoint, teremos no console a seguinte mensagem:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

> mongod@1.0.0 start
> nodemon index.js

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Listening on port 8080
13
█
```



## 7 NPM - Node Packet Manager

O NPM, que significa *Node Package Manager* é o gerenciador de pacotes padrão para o Node.js e consiste em um cliente de linha de comando, também chamado de npm, e um banco de dados online de pacotes públicos e privados pagos, chamado de registro npm. O registro é acessado por meio do cliente e os pacotes disponíveis podem ser navegados e pesquisados no site do npm.

O NPM permite, para além da criação de módulos, executar instruções ou conjuntos de instruções através de um comando criado pelo utilizador conforme a sua necessidade.

Agora já podemos fazer uma página simples.

Existem 3 formas de recebermos informações o nosso site

.params é para enviarmos junto de mais uma barra

```
const express = require('express');
const router = express.Router();

router.get('/:numero?', (req, res) => { // "?" serve para o dado ser opcional
  const { numero } = req.params
  res.send(`Número recebido: ${numero}`);
});

module.exports = router
```

← → ↺ 🏠 ⓘ localhost:3000

Número recebido: undefined

← → ↺ 🏠 ⓘ localhost:3000/12

Número recebido: 12

.query é para enviarmos dados mais simples pela url

```
const express = require('express');
const router = express.Router();

router.get('/:numero?', (req, res) => { // "?" serve para o dado ser opcional
  const { numero } = req.query
  res.send(`Número recebido: ${numero}`);
});
```

```
});  
  
module.exports = router
```

← → ↻ 🏠 ⓘ localhost:3000/?numero=1234

Número recebido: 1234