



Essencial

Cláudio Luís Vieira Oliveira

Conceitos fundamentais

O Node.js é um ambiente de servidor de código aberto e possibilita a execução de aplicações escritas em JavaScript podendo atuar, em aplicações para a Internet, como uma linguagem de programação server-side. Tecnicamente, consiste em um runtime JavaScript desenvolvido com o Chrome's V8 JavaScript engine.



Dica!

O **Node.js** é gratuito, multiplataforma e o instalador, tutoriais e documentação estão disponíveis em <https://nodejs.org/pt-br/>

A edição do código-fonte dos programas pode ser realizada no próprio bloco de notas ou em editores que apresentam mais recursos e são voltados para a programação como, por exemplo, o Notepad++, disponível em notepad-plus-plus.org ou o Sublime Text (www.sublimetext.com).

Porém, o mais recomendado é utilizar Ambientes Integrados de Desenvolvimento (IDEs) como o NetBeans (netbeans.org) que apresenta suporte ao desenvolvimento em Node.js ou o Microsoft Visual Studio Code (code.visualstudio.com) entre outras opções.



Atenção!

Os programas que serão desenvolvidos na sequência presumem que o Node.js está devidamente instalado e configurado no seu computador.

Olá Node.js

Este primeiro programa tem como intuito propiciar um primeiro contato com o Node.js e verificar se a instalação foi bem-sucedida.

```
{Node.js}
```

```
console.log('Olá Node.js');
```

ola.js

Abra o terminal de comandos do seu sistema operacional e digite:



```
node ola.js
```

Na Figura 1 temos o resultado da execução do programa criado.

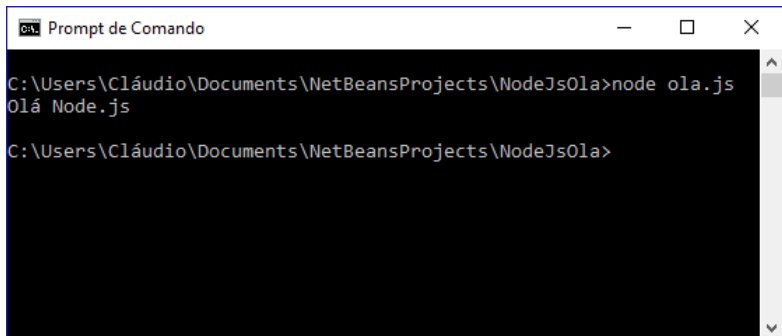


Figura 1: Resultado da execução do programa

Cláudio Luís V. Oliveira

Caso esteja usando uma IDE, como o Visual Studio Code, é possível executar o programa diretamente conforme mostrado na Figura 2.

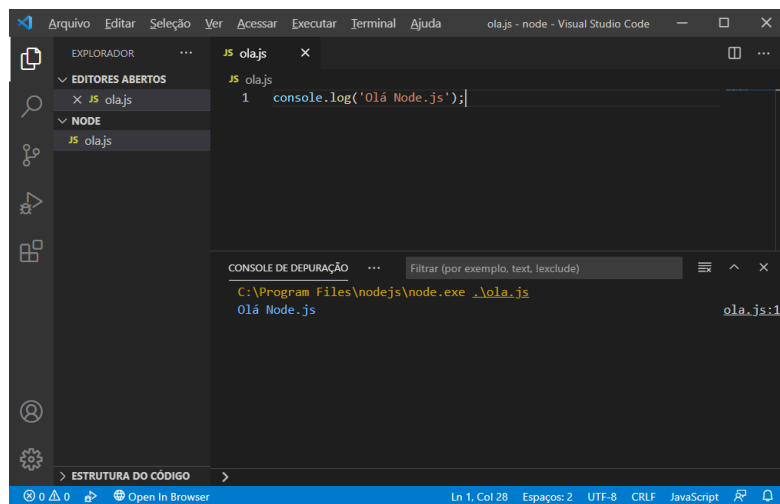


Figura 2: Execução do programa no Visual Studio Code

JavaScript

O JavaScript é uma linguagem de programação orientada a objetos que é interpretada e executada pelo navegador Web (client-side script) ou através como uma aplicação (Node.js). Apresenta uma sintaxe similar a linguagem Java e tem como objetivo principal dar uma maior interatividade às páginas.

Uma característica importante da linguagem JavaScript reside no fato de não possuir tipos de dados, sendo que qualquer variável definida é do tipo variante, ou seja, o tipo de dados é definido de acordo com a informação

de está armazenada naquele momento. O tipo de dados de determinada variável pode ser modificado ao longo da execução da aplicação, conforme o seu conteúdo vai sendo alterado.

Variáveis

Conforme abordado anteriormente, as variáveis em JavaScript não possuem um tipo de dados no momento de sua declaração, elas passam a assumir o tipo de dados dos valores que estão armazenados naquele instante. No exemplo a seguir, será possível observar que a variável `x` recebe inicialmente uma cadeia de caracteres, ou seja, neste momento, ela é do tipo de dados `String`.

Posteriormente ela recebe um valor numérico assumindo, a partir deste momento, o tipo de dados inteiro. Neste exemplo ela finaliza como sendo do tipo de dados `float`.

{Node.js}

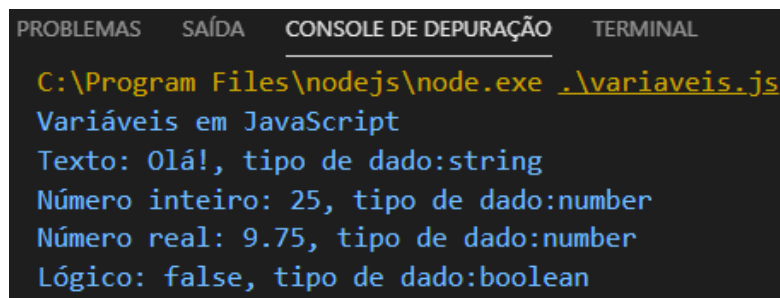
```
var x = "Olá!";
console.log("Variáveis em JavaScript");
console.log("Texto: " + x + ",
  tipo de dado:" + typeof x);
x = 12 + 13;
console.log("Número inteiro: " + x +
  ", tipo de dado:" + typeof x);
x = 4.50 + 5.25;
console.log("Número real: " + x +
  ", tipo de dado:" + typeof x);
x = false;
```

Cláudio Luís V. Oliveira

```
console.log("Lógico: " + x + ", tipo de  
dado:" + typeof x);
```

variaveis.js

Na Figura 3 apresentamos o resultado obtidos após a execução do programa.



```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  
C:\Program Files\nodejs\node.exe .\variaveis.js  
Variáveis em JavaScript  
Texto: Olá!, tipo de dado:string  
Número inteiro: 25, tipo de dado:number  
Número real: 9.75, tipo de dado:number  
Lógico: false, tipo de dado:boolean
```

Figura 3: Variáveis

Observe agora que os valores para o tipo de dados String devem ser especificados entre aspas simples (') ou duplas ("). Por exemplo:

{Node.js}

```
var x = "Olá";  
var y = 'amigo! ' ;  
console.log(x + ' ' + y);
```

amigo.js

Valores numéricos, por sua vez, não possuem delimitadores e como separador decimal deve-se utilizar o caractere de ponto final (.), conforme mostrado a seguir.

{Node.js}

```
var x = 12 + 13;
var y = 4.50 + 5.25;
console.log('x = ' + x + ', y = ' + y);
```

numeros.js

Operadores

De mesma maneira que outras linguagens de programação, JavaScript implementa um grande conjunto de operadores, vamos apresentar aqui os mais utilizados.

Grupo	Operador	Descrição
Atribuição	=	Atribuição simples
	+=	Atribuição de adição
	-=	Atribuição de subtração
	*=	Atribuição de multiplicação
	/=	Atribuição de divisão
	%=	Atribuição de resto
	**=	Atribuição de exponenciação
Relacional	==	Igual
	===	Exatamente igual (conteúdo e tipo de dado)
	!=	Diferente
	!==	Exatamente diferente (conteúdo e tipo de dado)
	<	Menor
	<=	Menor ou igual
	>	Maior
	>=	Maior ou igual

Grupo	Operador	Descrição
Aritméticos	+	Adição
	-	Subtração
	*	Multiplificação
	/	Divisão
	%	Resto da Divisão
	**	Exponenciação
	++	Incremento
	--	Decremento
Lógicos	&&	E (AND)
		OU (OR)
	!	NÃO (NOT)

No trecho de programa a seguir vamos ilustrar o uso de alguns operadores.

{Node.js}

```
var x = 10;  
x -= 4;  
var y = x ** 2;  
console.log('x = ' + x + ', y = ' + y);
```

operadores.js

Na primeira linha temos o operador de atribuição (=), isto é, a variável x irá receber o valor 10. Na linha seguinte usamos o operador de atribuição de subtração (-=), neste caso, x irá receber o valor de 10 (que é o valor atual de x) menos 4. Ou seja, ao final da operação x armazenará o valor 6.

Na terceira linha usamos o operador de exponenciação (**) e elevamos o valor de x, que é 6, ao quadrado (6²), desta forma, y irá armazenar 36.



Dica!

A **Mozilla Developer Network**, apresenta a relação completa dos operadores, acesse

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>

Entrada de dados através do console

Quando há necessidade de obtermos alguma entrada de dados, através do console (teclado), devemos utilizar a interface `readLine`, conforme ilustra o exemplo a seguir.

{Node.js}

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

const teclado = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

teclado.question('Digite o seu nome: ',
(resposta) => {
  console.log('Olá ' + resposta);
  teclado.close();
});
```

teclado.js

Observe que após a criação do objeto `teclado`, devemos usar o método `question` para permitir que o usuário realize a digitação.

Cláudio Luís V. Oliveira

No exemplo seguinte vamos solicitar ao usuário para digitar dois números e, em seguida, iremos calcular e exibir o valor da soma.

{Node.js}

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

const teclado = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

teclado.question('Digite o primeiro valor:
', (valor1) => {
  teclado.question('Digite o segundo valor:
', (valor2) => {
    var soma = parseInt(valor1) +
    parseInt(valor2);
    console.log("A soma é " + soma);
    teclado.close();
  });
});
```

somar.js

Estruturas de controle

A linguagem JavaScript implementa as tradicionais estruturas para controle de fluxo do programa, são elas:

if: permite definir a execução ou não de determinado bloco de código de acordo com a condição especificada. No

exemplo abaixo se testa o valor da variável `x` e, de acordo com o seu valor, o comando `if` determina qual bloco de código deverá ser executado.

{Node.js}

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

const teclado = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

teclado.question('Digite um número: ', (num)
=> {
  if (num > 5) {
    console.log("O valor é maior que
cinco.");
  }
  else if (num < 5) {
    console.log("O valor é menor que
cinco.");
  }
  else {
    console.log("O valor é igual a
cinco.<br>");
  }
  teclado.close();
});
```

estrutura-if.js

switch: de acordo com o valor da variável testada pelo comando, é determinado o bloco de código que será executado. Neste exemplo, o número que representa o dia

Cláudio Luís V. Oliveira

da semana é obtido e, com base em seu valor, o nome do dia da semana é armazenado em uma variável que será exibida no console.

{Node.js}

```
var hoje = new Date();
var diaSemana = hoje.getDay();
var nomeDia = "";
switch(diaSemana) {
  case 0:
    nomeDia = "Domingo";
    break;
  case 1:
    nomeDia = "Segunda-feira";
    break;
  case 2:
    nomeDia = "Terça-feira";
    break;
  case 3:
    nomeDia = "Quarta-feira";
    break;
  case 4:
    nomeDia = "Quinta-feira";
    break;
  case 5:
    nomeDia = "Sexta-feira";
    break;
  case 6:
    nomeDia = "Sábado";
}
console.log ("Hoje é " + nomeDia);
```

estrutura-switch.js

for: o comando permite repetição de um bloco de comandos enquanto a condição especificada seja verdadeira. No exemplo abaixo são mostrados os números inteiros de 0 a 4:

{Node.js}

```
console.log("Contando ... ");  
for (var i = 0; i < 5; i++) {  
    console.log(i);  
}
```

estrutura-for.js

while: similar ao comando for, executa a repetição de um bloco de programa. Neste exemplo mostramos os números pares entre 2 e 10:

{Node.js}

```
console.log("Pares ... ");  
var num = 2;  
while (num <= 10) {  
    console.log(num);  
    num += 2;  
}
```

estrutura-while.js

do-while: é similar aos comandos for e while apresentados anteriormente, porém, apresenta uma diferença fundamental, como o a verificação da condição ocorre apenas após a execução do bloco de programa a ser repetido, ele sempre será executado pelo menos uma vez. Quando se utiliza for e while se a condição inicial resultar em

Cláudio Luís V. Oliveira

falso o bloco a ser repetido nunca é executado. O exemplo a seguir exibe a tabuada do 3:

{Node.js}

```
console.log('Tabuada do 3... ');
var num = 1;
do {
    console.log('3 x ', num, ' = ', num * 3);
    num += 1;
} while (num <= 10);
```

estrutura-do.js

Vetores

O objeto Array permite a definição de vetores cujos elementos poderão ser acessados posteriormente através do respectivo índice. Neste exemplo um vetor é declarado e inicializado com os meses do ano, posteriormente, a instrução for será usada para acessar cada um dos elementos do vetor e os exibir. Note também que o primeiro elemento do vetor possui o índice 0 (zero).

{Node.js}

```
var meses = new Array("Janeiro",
    "Fevereiro", "Março", "Abril", "Maio",
    "Junho", "Julho", "Agosto", "Setembro",
    "Outubro", "Novembro", "Dezembro");
console.log("Meses do ano:");
for (var i = 0; i < meses.length; i++) {
    console.log (meses[i]);
}
```

```
}
```

vetor.js

Há também uma outra forma de usar a instrução for que é muito útil quando trabalhamos com vetores. Neste caso, não é preciso se preocupar com o tamanho do vetor ou mesmo ter que garantir que apenas índices válidos sejam acessados. No exemplo a seguir notamos que a variável `dia` irá conter, a cada repetição, um valor válido para o índice do vetor.

{Node.js}

```
var semana = new Array("Domingo",  
    "Segunda-feira", "Terça-feira",  
    "Quarta-feira", "Quinta-feira",  
    "Sexta-feira", "Sábado");  
console.log("Dias da semana:");  
for (var dia in semana) {  
    console.log (semana[dia]);  
}
```

vetor-for-in.js

Números aleatórios

Muitas vezes é necessário a geração de números aleatórios também conhecidos como randômicos. Dentre muitas aplicações, é muito útil quando precisamos gerar longas sequências numéricas. No exemplo a seguir vamos aplicar este conceito para popular um vetor com 100 números inteiros entre 1 e 500.

{Node.js}

```
var num = Array();
console.log ('Tamanho inicial: ' +
  num.length);
for (var i = 0; i < 100; i++) {
  num[i] = Math.floor(
    Math.random() * 500 + 1)
  console.log(num[i]);
}
console.log ('Tamanho final: ' +
  num.length);
```

vetor-numeros.js

Observe que no JavaScript podemos declarar o vetor, sem precisar definir o seu tamanho máximo e vamos inserindo os novos elementos conforme for necessário.

Modularização

A modularização é um importante recurso que é encontrado em linguagens de programação e permite estruturar, organizar e reutilizar funções que serão utilizadas no desenvolvimento dos programas.

Desta forma, é possível a criação de funções, conforme o exemplo apresentado a seguir.

{Node.js}

```
function somar (num1, num2) {
  return (num1 + num2);
```



```

}

var res = somar(4, 5);
console.log(res);

```

funcao-uso.js

As funções podem ser compartilhadas para vários programas diferentes através de módulos. Além dos inúmeros módulos prontos que estão disponíveis para o Node.js, é possível criarmos os nossos próprios módulos. Com o intuito de ilustrar este conceito, vamos criar um módulo que irá conter funções que nos permitirá simplificar a geração de números randômicos.

Conforme podemos observar no código-fonte a seguir, no arquivo a ser criado iremos implementar duas funções que serão exportadas. A função real irá retornar um valor real entre 0 e 1, enquanto a função inteiro irá retornar um número inteiro entre os valores de início e fim, passados como parâmetros.

{Node.js}

```

exports.real = function () {
  return Math.random();
};

exports.inteiro = function (inicio, fim) {
  return Math.floor(Math.random() * fim +
    inicio);
};

```

modulo-sortear.js

Cláudio Luís V. Oliveira

Em seguida, vamos criar um programa que fará uso do módulo que foi criado.

{Node.js}

```
const sortear = require('./modulo-  
sortear.js');
```

```
console.log(sortear.real());  
console.log(sortear.inteiro(1, 10));
```

modulo-sortear.js

Observe que a função require será responsável pelo carregamento do módulo. Posteriormente, podemos utilizar as funções do módulo no nosso programa.

Exercícios

- 1) Desenvolver uma aplicação Node.js que receba quatro números reais digitados pelo usuário, calcule e exiba o valor da média desses números.
- 2) Desenvolver um programa para uma determinada loja que precisa calcular o preço de venda de um produto. O cálculo deverá ser efetuado através da multiplicação do preço unitário pela quantidade vendida e, posteriormente, subtrair o valor do desconto. Considerar todas as variáveis do tipo de dado real, que serão digitadas pelo usuário.

- 3) A Lei de Ohm define que a resistência (R) de um condutor é obtida através da divisão da tensão aplicada (V) dividida pela intensidade de corrente elétrica (A). Desta forma, a partir de uma tensão e corrente, digitadas pelo usuário, calcular e mostrar o valor da resistência.
- 4) Elaborar uma aplicação Node.js que a partir de um valor digitado pelo usuário e o respectivo prefixo, mostrar a representação do valor nos demais prefixos, por exemplo:

Digite o valor:

Digite o prefixo:

Resultado:

10.000.000 k

10 G

0,01 T

Adotar, como referência a tabela mostrada abaixo:

Prefixo	Valor (Decimal)
k (kilo)	10^3 (1000)
M (mega)	10^6 (1,000,000)
G (Giga)	10^9 (1,000,000,000)
T (Tera)	10^{12} (1,000,000,000,000)

- 5) Considerando que a aprovação de um aluno em determinada disciplina requer uma média final maior ou igual a 6,0 (seis). Elaborar um programa que receba duas notas, realize o cálculo da média, exiba o valor calculado, indicando se o aluno está aprovado ou reprovado.
- 6) A partir dos lados de um retângulo ou quadrado, digitados pelo usuário, elaborar uma aplicação que calcule e exiba o valor da área da figura e informe se é um retângulo ou um quadrado. Lembrando que a área é obtida pela multiplicação da base (L) pela altura (A).
- 7) Considerando um número inteiro digitado pelo usuário, calcular e exibir o valor da sua fatorial. Por exemplo, se o usuário digitar 4, temos que a fatorial é $4 \times 3 \times 2 \times 1$, isto é, 24.
- 8) Considerando três números inteiros digitados pelo usuário, determinar e exibir o maior número.
- 9) Criar um programa que mostre a data e hora atuais da maneira que é utilizada no Brasil, por exemplo: "25/01/2021 19:45".
- 10) Criar um programa que mostre a data atual por extenso, por exemplo: "Hoje é sexta-feira, dia 26 de fevereiro de 2021".
- 11) Considerando um vetor contendo 200 números inteiros com valores entre 1 e 1000, gerado

aleatoriamente, exibir o valor do maior elemento armazenado no vetor.

- 12) A partir de um vetor com 50 números inteiros com valores entre 1 e 500, gerado aleatoriamente, exibir a média dos valores armazenados neste vetor.
- 13) Implementar um módulo que apresente as seguintes funções:
 - **data (dt)** - retornar a data, passada como parâmetro, no formato dd/mm/aaaa.
 - **hora(dt)** - retornar a hora, a partir de uma data passada como parâmetro, no formato hh:mm.
 - **dataExtenso(dt)** - retornar a data, passada como parâmetro, por extenso, por exemplo: sexta-feira, 26 de fevereiro de 2021.
- 14) Desenvolver um módulo que possua as seguintes funções:
 - **media (vetor)** - retornar a média dos valores contidos em um vetor passado como parâmetro.
 - **menor (vetor)** - retornar o menor valor contido no vetor passado como parâmetro.
 - **maior (vetor)** - retornar o maior valor contido em um vetor passado como parâmetro.

Aplicações para a Internet

O Node.js é amplamente utilizado no desenvolvimento de aplicações para a Internet, se integrando com páginas desenvolvidas em HTML, CSS e JavaScript executando no cliente, ou seja, no navegador. Além disso, possui disponível em grande número de pacotes (módulos) que irão facilitar o desenvolvimento destas aplicações.



Dica!

O **npm** (Node Package Manager) é um gerenciador de pacotes para o Node.js.

Implementando um servidor

Vamos usar o módulo express para implementar um servidor de páginas web através do Node.js. Desta maneira, o primeiro passo consiste em instalar o express que é projetado de modo a facilitar a criação de aplicações para a web. A instalação deve ser feita na pasta criada para a aplicação, onde o npm deverá ser executado da maneira indicada a seguir.



```
.....  
npm install express  
.....
```

Em seguida, vamos elaborar o programa que irá criar um servidor local executando da porta 8080.

{Node.js}

```
var fs = require('fs');
var express = require('express');
var app = express();

var servidor = app.listen(8080, function() {
  var porta = servidor.address().port;
  console.log("Servidor executando na porta
%s", porta);
});

app.get('/', function (req, res) {
  fs.readFile('ola.html', function(erro,
dado) {
    res.writeHead(200, {'Content-Type':
'text/html'});
    res.write(dado);
    res.end();
  });
});
```

servidor-ola.js

Analizando o programa, observe que o método app.listen irá criar um servidor na porta TCP indicada, enquanto app.get determina a ação que será tomada de acordo com o caminho passado que, no nosso exemplo, é a raiz do site (caractere '/'). Então o método fs.readFile irá abrir o arquivo HTML mostrado a seguir que, então, será enviado para o navegador através do método res.write.



```
<!DOCTYPE html>
<html>
  <head>
    <title>Olá Node.js</title>
    <meta charset="UTF-8">
    <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/boo
tstrap/4.3.1/css/bootstrap.min.css">
  </head>
  <body>
    <div class="container">
      <div class="jumbotron">
        <h1>Olá Node.js</h1>
      </div>
    </div>
  </body>
</html>
```

ola.html

Utilize o Node.js para executar o arquivo servidor-ola.js conforme mostrado a seguir.



```
node servidor-ola.js
```

Então, vá no navegador e abra o site local na porta 8080, isto é, digite localhost:8080/ na barra de endereço e

observe que a página HTML será exibida, conforme ilustra a Figura 4.

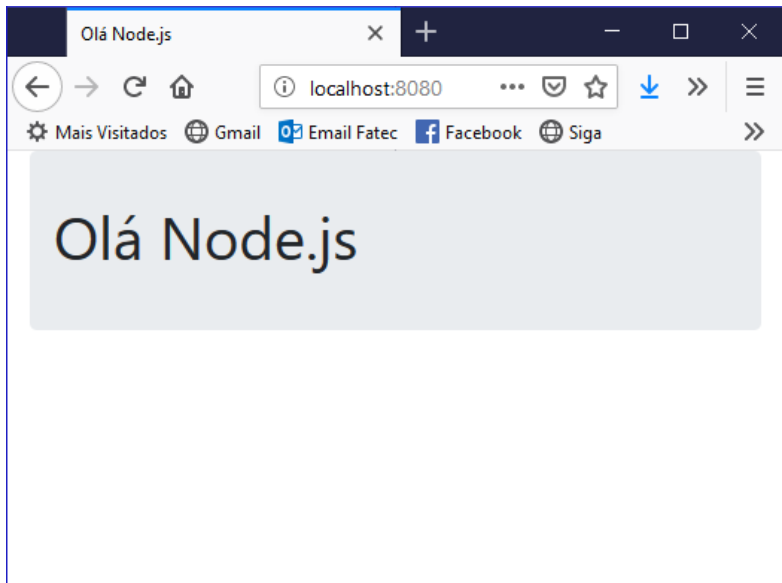


Figura 4: Página carregada

Quando desejar, utilize a combinação de teclas “Ctrl C” para finalizar a execução do servidor.

Interação com HTML

Através do Express podemos interagir com elementos HTML e com outros arquivos frequentemente presentes em projetos de aplicações para a Internet como, por exemplo, outros arquivos HTML, folhas de estilo (CSS), imagens e Javascript. No exemplo seguinte ilustramos este

Cláudio Luís V. Oliveira

recurso, possibilitando carregar aleatoriamente uma imagem que será exibida em uma página HTML.

Antes de implementar o programa, crie uma pasta chamada `img` e copie para ela imagens representando cada uma das faces de um dado.

{Node.js}

```
var express = require('express');
var app = express();

var servidor = app.listen(8080, function() {
  var porta = servidor.address().port;
  console.log("Servidor executando na porta
%s", porta);
});

app.use(express.static('img'));
app.get('/', function (req, res) {
  res.writeHead(200, {'Content-Type':
'text/html'});
  var face = Math.floor(
    Math.random() * 6 + 1);
  res.end('<h1>Jogo de Dados</h1>');
});
```

servidor-estatico.js

Note que o programa é bastante parecido com o que foi apresentado no exemplo anterior. Mas, neste caso, usamos o método `app.use` para informar ao servidor que iremos usar um conteúdo estático, ou seja, os arquivos que estão na pasta `img`. Em seguida, sorteamos um valor entre 1

e 6 que corresponderá à face do dado que deverá ser exibida. Por fim, o método res.end será usado para exibir a respectiva imagem, conforme ilustra a Figura 5.

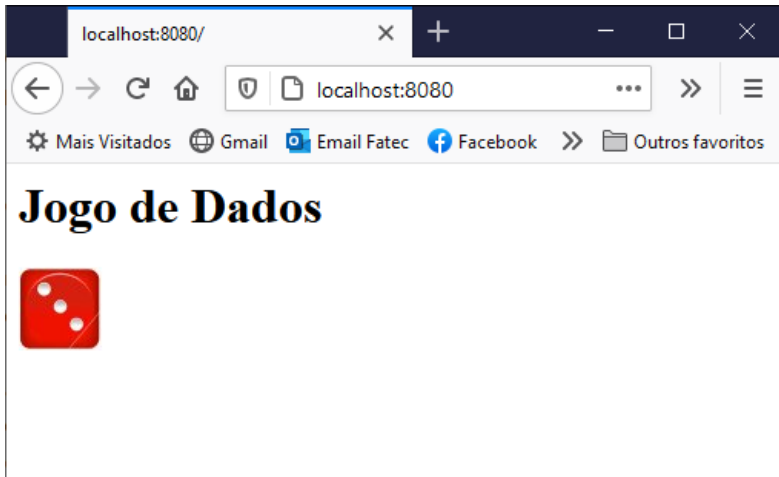


Figura 5: Página exibindo a face sorteada

Experimente recarregar a página várias vezes e observe que uma nova face será mostrada.

Construção de uma aplicação

Através do Node.js é bastante simples trabalhar com os dados provenientes de formulários HTML. Esta próxima aplicação irá demonstrar como podemos obter os valores dos campos de formulários e processá-los no servidor.

Começamos criando a página HTML que irá implementar o formulário, vamos usar apenas dois campos, o nome da pessoa e o ano que ela nasceu, além de um botão para realizar a submissão (envio) do formulário.



```
<!DOCTYPE html>
<html>
  <head>
    <title>Idade</title>
    <meta charset="UTF-8">
    <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/boo
tstrap/4.3.1/css/bootstrap.min.css">
  </head>
  <body>
    <div class="container">
      <div class="jumbotron">
        <h1>Idade</h1>
      </div>
      <form id="form" method="POST"
action="/idade">
        <div class="form-group">
          Nome: <input type="text"
name="nome" class="form-control" />
        </div>
        <div class="form-group">
          Ano de Nascimento: <input
type="text" name="anonasc" class="form-
control" />
        </div>
        <input type="submit"
value="Calcular" class="btn btn-primary" />
      </form>
    </div>
```

```
</body>
</html>
```

idade-form.html

Observe no trecho de código-fonte destacado a seguir, que os dados do formulário serão submetidos (enviados) para o endereço “/idade” do servidor local, que será posteriormente implementado.



```
<form id="form" method="POST"
action="/idade">
```

Em seguida, já vamos criar a página HTML que irá receber os dados processados pelo servidor. Observe que os dados que serão posteriormente enviados para esta página estão identificados como {{nome}}, {{anonym}} e {{idade}}.



```
<!DOCTYPE html>
<html>
  <head>
    <title>Idade</title>
    <meta charset="UTF-8">
    <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/boo
tstrap/4.3.1/css/bootstrap.min.css">
  </head>
  <body>
```

Cláudio Luís V. Oliveira

```
<div class="container">
  <div class="jumbotron">
    <h1>Idade</h1>
  </div>
  Nome: {{nome}}<br>
  Ano de Nascimento: {{anonasc}}<br>
  Idade: {{idade}} anos.
</div>
</body>
</html>
```

idade-res.html

Uma vez criadas as páginas com o conteúdo HTML passaremos para a criação da aplicação Node.js.

{Node.js}

```
var fs = require('fs');
var http = require("http");
var express = require('express');
var app = express();
var bodyParser = require('body-parser');
var urlencodedParser =
bodyParser.urlencoded({ extended: true });

var servidor = app.listen(8080, function() {
  var porta = servidor.address().port;
  console.log("Servidor executando na porta
%s", porta);
});

app.get('/', function (req, res) {
  fs.readFile('idade-form.html',
function(erro, dado) {
```

```
        res.writeHead(200, {'Content-Type':  
'text/html'});  
        res.write(dado);  
        res.end();  
    });  
});  
  
app.post('/idade', urlencodedParser,  
function (req, res){  
    fs.readFile('idade-res.html',  
function(erro, dado) {  
    var hoje = new Date();  
    var valores = {  
        'nome': req.body.nome,  
        'anonasc': req.body.anonasc,  
        'idade': (hoje.getFullYear() -  
parseInt(req.body.anonasc))  
    };  
    for (var chave in valores) {  
        dado = dado.toString().replace("{}" +  
chave + "}", valores[chave]);  
    }  
    res.writeHead(200, {'Content-Type':  
'text/html'});  
    res.write(dado);  
    res.end();  
    });  
});  
});
```

servidor-idade.js

Utilize o Node.js para executar o arquivo servidor-idade.js. conforme mostrado a seguir.



node servidor-idade.js

No navegador, abra o site digitando localhost:8080/ na barra de endereço. Preencha o formulário de maneira similar à mostrada na Figura 6.

The screenshot shows a web browser window with the title 'Idade'. The address bar displays 'localhost:8080'. Below the address bar, there are several icons for 'Mais Visitados', 'Gmail', 'Email Fatec', 'Facebook', and 'Siga'. The main content area of the browser displays a form with the title 'Idade' in a large font. Below the title, there are two input fields: 'Nome:' with the value 'José da Silva' and 'Ano de Nascimento:' with the value '1990'. At the bottom of the form, there is a blue button labeled 'Calcular'.

Figura 6: Formulário para cálculo da idade

Quando pressionar o botão “Calcular” o servidor irá processar os dados e enviar o resultado para a página “idade-res.html”, conforme ilustra a Figura 7.

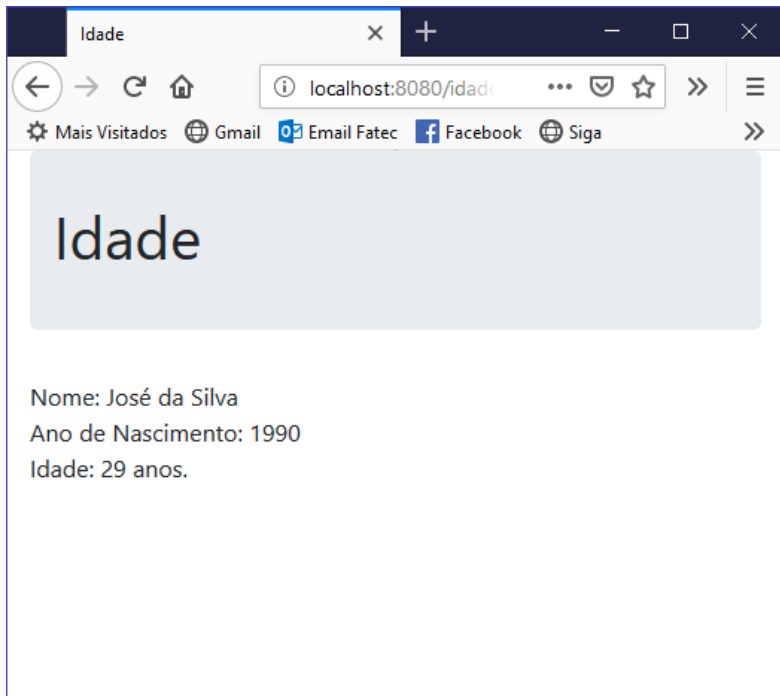


Figura 7: Exibição dos dados processados

Após a execução desta aplicação vamos analisar, em detalhes, o seu funcionamento. O primeiro bloco de programa, mostrado a seguir, consiste em realizar a carga dos módulos e definir a variável `app` e as variáveis para realizar o parse, ou seja, a obtenção dos elementos que compõem o endereço da página (URL) e os dados submetidos.

{Node.js}

```
var fs = require('fs');  
var http = require("http");  
var express = require('express');  
var app = express();
```

Cláudio Luís V. Oliveira

```
var bodyParser = require('body-parser');  
var urlencodedParser =  
bodyParser.urlencoded({ extended: true });
```

A seguir, observe no trecho a seguir, que ele será o responsável pela criação do servidor que irá receber as solicitações do navegador através da porta TCP 8080.

{Node.js}

```
var servidor = app.listen(8080, function() {  
  var porta = servidor.address().port;  
  console.log("Servidor executando na porta  
%s", porta);  
});
```

Quando o servidor receber a requisição do navegador para acessar a raiz do site, o trecho de programa mostrado a seguir será responsável por enviar o conteúdo do arquivo “formulario.html” para o navegador.

{Node.js}

```
app.get('/', function (req, res) {  
  fs.readFile('formulario.html',  
function(erro, dado) {  
  res.writeHead(200, {'Content-Type':  
'text/html'});  
  res.write(dado);  
  res.end();  
});  
});
```

Concluindo esta aplicação, no trecho de código-fonte, apresentado a seguir, quando o servidor receber a requisição “/idade” este irá carregar o arquivo “idade.html” e obter os dados que foram enviados através da submissão do formulário. Para obter os dados, usamos a propriedade req.body.nome, onde nome corresponde ao valor da propriedade id do campo do formulário.

Em seguida, usamos o método `replace` para trocar os marcadores `{{nome}}`, `{{anonasc}}` e `{{idade}}` pelos dados obtidos e processados pelo servidor.

{Node.js}

```
app.post('/idade', urlencodedParser,
function (req, res){
  fs.readFile('idade.html', function(erro,
dado) {
    var hoje = new Date();
    var valores = {
      'nome': req.body.nome,
      'anonasc': req.body.anonasc,
      'idade': (hoje.getFullYear() -
parseInt(req.body.anonasc))
    };
    for (var chave in valores) {
      dado = dado.toString().replace("{}" +
chave + "}", valores[chave]);
    }
    res.writeHead(200, {'Content-Type':
'text/html'});
    res.write(dado);
    res.end();
  });
});
```

Exercícios

- 1) Desenvolver uma aplicação Internet em Node.js que receba quatro números reais digitados pelo usuário através de um formulário HTML e, em seguida, calcule e exiba a valor da média dos números em uma nova página HTML.
- 2) Desenvolver uma aplicação web em Node.js para uma determinada loja que precisa calcular o preço de venda de um produto. O cálculo deverá ser efetuado através da multiplicação do preço unitário pela quantidade vendida e, posteriormente, subtrair o valor do desconto. Considerar todas as variáveis do tipo de dado real, que serão digitadas pelo usuário através de um formulário HTML.
- 3) Considerando o desenvolvimento de uma aplicação Node.js para a Internet e sendo que a Lei de Ohm define que a resistência (R) de um condutor é obtida através da divisão da tensão aplicada (V) dividida pela intensidade de corrente elétrica (A). Desta forma, a partir de uma tensão e corrente, digitadas pelo usuário através de um formulário HTML, calcule e mostre o valor da resistência.
- 4) Desenvolver um servidor, usando Node.js, que apresente uma página HTML contendo a data e hora atuais.
- 5) Desenvolver um servidor, usando Node.js, que simule um uma página HTML um dado de jogo sendo lançado aleatoriamente por 50 vezes. A mesma

página deverá exibir quantas vezes cada uma das faces foi sorteada.

- 6) Desenvolver um servidor, usando Node.js, que simule um uma página HTML uma moeda sendo lançada aleatoriamente por 200 vezes. A mesma página deverá exibir a porcentagem de caras e coroas que foram sorteadas.
- 7) Considerando que a aprovação de um aluno em determinada disciplina requer uma média final maior ou igual a 6,0 (seis). Elaborar uma aplicação Node.js para a web que receba através de um formulário HTML, duas notas, realize o cálculo da média, exiba o valor calculado e uma imagem indicando se o aluno está aprovado ou reprovado.
- 8) Realizar o desenvolvimento de um servidor Node.js que disponibilize uma página HTML que irá receber as informações através de um formulário e realizar a validação dos dados digitados, considerando que uma seguradora de veículos precisa calcular o valor da apólice com base nas seguintes informações: nome, sexo e ano de nascimento do segurado, marca, modelo, ano de fabricação, valor do veículo e porcentagem do bônus. As seguintes validações deverão ser realizadas na própria página HTML que contém o formulário:
 - a) O campo sexo deverá aceitar apenas F (Feminino) ou M (Masculino).
 - b) O campo ano de nascimento deve aceitar um valor entre 2001 e 1901.

- c) O campo ano de fabricação deverá ser um valor inteiro positivo.
- d) O campo valor do veículo deve ser um número real positivo.
- e) O campo porcentagem do bônus deverá ser um número real entre 0 e 25.

Quando o formulário for submetido o servidor deverá determinar o valor da apólice, a partir dos seguintes critérios para cálculo:

- a) Para veículos 2010 ou mais recentes o valor da apólice é de 1,25% do valor do veículo, veículos entre 2009 e 2000 o valor da apólice é de 1,75% do valor do veículo, veículos entre 1999 e 1980 o valor da apólice é de 2,00% e para os demais anos de fabricação devemos utilizar 2,50% como base de cálculo.
- b) Caso o segurado seja do sexo feminino aplicar um desconto 10% sobre o valor calculado no item a, caso contrário, acrescer 5% ao valor calculado no item a.
- c) Se o segurado possuir menos de 30 anos ou mais de 60 anos, acrescentar 20% ao valor da apólice após os cálculos realizados no item a e no item b.
- d) A partir do valor apurado nos itens a, b e c aplicar o desconto com base na porcentagem de bônus informada pelo usuário.

Por fim, uma página HTML deverá ser exibida apresentando o valor da apólice.

JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) ou traduzindo Notação de Objetos JavaScript é uma notação leve para representação e troca de dados. É de fácil compreensão, leitura e escrita. Adota formato texto, é auto descritivo e completamente independente de linguagem.

Uma especificação em JSON está constituída em duas estruturas. A primeira delas, é chamada de objeto e consiste em uma coleção de pares (nome e valor) o que pode caracterizar um objeto, registro, dicionário ou arrays associativos. Um item da coleção deve ser delimitado por chaves. Cada nome é seguido pelo caractere ':' (dois pontos) e, em seguida, o valor deve ser especificado. Vários pares (nome/valor) devem ser separados por vírgula, conforme ilustra o próximo exemplo.

{JSON}

```
{ codigo: 10, descricao: "Televisor", preco: 1990.00 }
```

Desta forma, analisando a estrutura, podemos entender que estamos especificando um par que apresenta o nome 'codigo' que possui o valor 10. Depois temos o par cujo nome é 'descricao' e que tem o valor "Televisor" e o par chamado 'preco' que tem o valor 1990.00.

Manipulação no Node.js

Utilizar JSON no Node.js é muito simples, basta referenciar o objeto e a respectiva chave, conforme podemos ver no programa a seguir.

{Node.js}

```
var produto = { codigo: 10,  
  descricao: "Televisor",  
  preco: 1990.00 };  
console.log("Dados do produto: ");  
console.log("Código: " + produto.codigo);  
console.log("Descrição: " +  
produto.descricao);  
console.log("Preço: " + produto.preco);
```

json-basico.js

A notação JSON permite obter informações de maneira dinâmica através do uso de funções. Note no exemplo a seguir que o valor da chave date será obtido através da função hoje() que, por sua vez, irá possibilitar obter a data atual.

{Node.js}

```
function hoje() {  
  var agora = new Date();  
  return (agora.getDate() + "/" +  
    (agora.getMonth() + 1) + "/" +  
    agora.getFullYear());  
}
```



```
var produto = { codigo: 10, descricao:
"Televisor", preco: 1990.00,
data: function () { return hoje(); } };
console.log("Dados do produto: ");
console.log("Código: " + produto.codigo);
console.log("Descrição: " +
produto.descricao);
console.log("Preço: " + produto.preco);
console.log("Data Cadastro: " +
produto.data());
```

json-funcao.js

A segunda estrutura do JSON é chamada de array e consiste em uma lista ordenada de valores que pode caracterizar array, vetor, lista, sequência ou mesmo uma coleção de valores ordenados. Um array é delimitado por colchetes onde os valores são separados por vírgula, conforme podemos observar no exemplo a seguir.

{JSON}

```
[ 10, 20, 30, 40, 50 ]
```

Um array pode conter objetos, desta maneira, podemos realizar a seguinte notação:

{JSON}

```
[ { codigo: 11, descricao: "Geladeira",
preco: 2990.00 }, { codigo: 12, descricao:
"Fogao", preco: 840.00 }, { codigo: 13,
descricao: "Computador", preco: 3500.00 } ]
```

Manipular objetos e arrays JSON em JavaScript é uma tarefa bastante simples. Neste próximo programa vamos mostrar como percorrer o array de objetos armazenado na variável produtos.

{Node.js}

```
var produtos = [ { codigo: 11, descricao:
"Geladeira", preco: 2990.00 },
{ codigo: 12, descricao: "Fogao", preco:
840.00 },
{ codigo: 13, descricao: "Computador",
preco: 3500.00 } ];

console.log("Produtos: ");
for (var i = 0; i < produtos.length; i++) {
    console.log("---");
    console.log("Código: " +
produtos[i].codigo);
    console.log("Descrição: " +
produtos[i].descricao);
    console.log("Preço: " +
produtos[i].preco);
}
```

json-colecao.js

Aprofundando um pouco mais na interação entre JSON e JavaScript, vamos implementar um exemplo de aplicação para a Internet que mostrará como podemos inserir e remover itens à um array e listar o seu conteúdo, considerando um conjunto de produtos que apresentam, como características, código e descrição e preço. É importante observar que, neste exemplo, iremos processar

o JavaScript no lado cliente (navegado), sem utilizar o Node.js.

A página HTML, mostrada a seguir, irá possuir três caixas de texto para que o usuário possa digitar as características do produto e um botão para inserção. A div identificada como “tabela” irá, posteriormente, mostrar o conjunto de dados que estarão no array JSON.



```
<html>
  <head>
    <title>Produtos</title>
    <meta charset="UTF-8">
    <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
    <script type="text/javascript"
src="json-produto.js"></script>
  </head>
  <body>
    <h1>Produtos</h1>
    Código: <input type="text" id="cod"
size="5" />
    Descrição: <input type="text" id="desc"
size="20" />
    Preço: <input type="text" id="prec"
size="10"/>
    <input type="submit" value="Inserir"
onclick="inserir()" /><br>
    <h2>Lista de Produtos</h2>
    <div id="tabela"></div>
  </body>
</html>
```

As funções JavaScript serão implementadas em um arquivo a parte, cujo conteúdo é mostrado a seguir. A variável produtos irá conter o array. Observe na função inserir, que o método push irá adicionar um objeto ao array. Por outro lado, o método splice será responsável pela remoção de objetos do array produtos. Este método deve receber, como parâmetros, a posição do item no array e quantos itens devem ser removidos que, neste exemplo, deixamos fixo o valor 1. Ou seja, apenas será apagado o item que está na posição que foi passada como parâmetro.

{JS}

```
var produtos = [];  
  
function inserir() {  
    produtos.push({codigo:  
    parseInt(cod.value), descricao: desc.value,  
    preco: parseFloat(prec.value)});  
    mostrar();  
}  
  
function remover(i) {  
    produtos.splice(i, 1);  
    mostrar();  
}  
  
function mostrar() {  
    console.log(JSON.stringify(produtos));  
    var conteudo = "<table cellpadding='0'  
    cellspacing='4' border='1'>" +  
        "<tr><td>Código</td>" +
```

```

        "<td>Descrição</td>" +
        "<td>Preço</td></tr>";
    for (var i in produtos) {
        conteudo += "<tr><td><button
onclick='remover(" +
            i + ")'><img src='deletar.png'
height='12px'></button> " +
            produtos[i].codigo +
            "</td><td> " + produtos[i].descricao +
            "</td><td align='right'>" +
            produtos[i].preco.toFixed(2) + "</td></tr>";
    }
    conteudo += "</table>";
    tabela.innerHTML = conteudo;
}

```

.....
json-produto.js

A função mostrar irá montar uma tabela para exibir os dados que estão armazenados no array produtos. Esta função é chamada sempre que ocorre uma inserção ou remoção no array, de modo que as informações exibidas na página web permaneçam sempre atualizadas. Na Figura 8 temos um exemplo da página criada.



Produtos

Código: Descrição: Preço:

Lista de Produtos

Código	Descrição	Preço
 10	Televisor	3000.00
 11	Computador	1800.00
 12	Mesa com 4 cadeiras	600.00

Figura 8: Dados do array JSON



Aprenda mais sobre **JSON** acessando
<https://www.json.org/json-pt.html>

Dica!

Exercícios

- 1) Considerando um aluno com nome, ra, duas notas e situação acadêmica (aprovado ou reprovado), elaborar uma representação JSON considerando uma sala de aula com 5 alunos.
- 2) A partir da representação JSON desenvolvida no exercício anterior, elaborar um programa em modo

console de texto que mostre o nome dos alunos reprovados.

- 3) Considerando o estoque de uma loja, elaborar a representação JSON para um conjunto de produtos, onde cada um possui as seguintes características: código, descrição, preço unitário, quantidade e fornecedor.
- 4) Considerando a representação JSON desenvolvida no exercício anterior, elaborar uma página web com JavaScript no lado cliente que mostre apenas os produtos que apresentem estoque inferior a 10 unidades.

Uso do banco de dados MariaDB

Uma das grandes vantagens de processar o JavaScript no lado servidor, através do Node.js, é a possibilidade de acesso aos Sistemas Gerenciadores de Banco de Dados (SGBDs). Com o objetivo de ilustrar os conceitos que envolvem a inclusão, alteração, exclusão e consulta ao banco de dados vamos desenvolver uma aplicação em Node.js que irá demonstrar estas operações sendo realizadas em um banco de dados implementado no MariaDB, que consiste na versão gratuita e com código-fonte aberto do MySQL.



Dica!

Instale o **XAMPP Apache + MariaDB + PHP + Perl** que contém, entre outros programas, o MariaDB e o phpMyAdmin que é ambiente web para administração do banco de dados. O XAMPP é gratuito, multiplataforma e pode ser baixado de

https://www.apachefriends.org/pt_br/index.html

Considerando o MariaDB devidamente instalado e a partir do ambiente de gerenciamento, crie o banco de dados **loja** e a tabela **produto** utilizando a estrutura apresentada a seguir.

Tabela: produto			
<i>Nome do Campo</i>	<i>Tipo de Dados</i>	<i>Anulável</i>	<i>Chave Primária</i>
codigo	INT	Não	Sim
descricao	VARCHAR(40)	Não	Não
preco	FLOAT	Sim	Não

Aplicação Node.js com acesso ao MariaDB

Uma aplicação Node.js pode facilmente acessar um banco de dados do MariaDB. O primeiro passo é instalar o módulo `mysql`, para isso vá até a pasta da aplicação Node.js que será criada e use o comando `npm` conforme mostrado a seguir.



```
cd suaaplicacao
npm install mysql --save
```

Em primeiro lugar vamos mostrar uma aplicação simples que irá exibir, no próprio console, os registros armazenados na tabela `produto` que foi criada anteriormente. Observe que iremos realizar a conexão ao MariaDB, para isso devemos especificar o endereço (url) do servidor, usuário, senha e nome do banco de dados a ser acessado como parâmetros do método `mysql.createConnection`.

{Node.js}

```
var mysql = require('mysql');

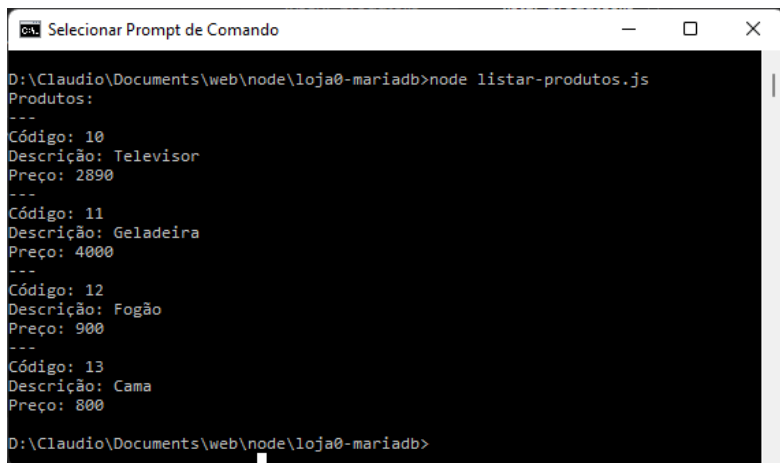
var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "",
  database: "loja"
});

con.connect(function(erro) {
  if (erro) throw erro;
  con.query("SELECT * FROM produto",
function (erro, resultado) {
  console.log("Produtos: ");
  for (var i = 0; i < resultado.length;
i++) {
    console.log("---");
    console.log("Código: " +
resultado[i].codigo);
    console.log("Descrição: " +
resultado[i].descricao);
    console.log("Preço: " +
resultado[i].preco);
  }
  con.end();
});
});
```

listar-produtos.js

Após a conexão é realizada a execução da instrução que irá realizar a consulta ao banco de dados, utilizando, para isso, o método query. Concluindo o exemplo, usamos a

instrução `for` para percorrer os registros obtidos pela consulta, que foram recuperados em formato JSON. A execução da aplicação deverá produzir um resultado similar ao mostrado pela Figura 9.



```
Selecionar Prompt de Comando
D:\Claudio\Documents\web\node\loja0-mariadb>node listar-produtos.js
Produtos:
---
Código: 10
Descrição: Televisor
Preço: 2890
---
Código: 11
Descrição: Geladeira
Preço: 4000
---
Código: 12
Descrição: Fogão
Preço: 900
---
Código: 13
Descrição: Cama
Preço: 800
D:\Claudio\Documents\web\node\loja0-mariadb>
```

Figura 9: Exibição dos registros da tabela produto

Inserção, alteração e exclusão através do Node.js

A aplicação seguinte irá inserir um registro na tabela produto. Note o uso do `mysql.createConnection` para passarmos as informações para conexão ao banco de dados. Em seguida, através do método `connect`, realizamos a conexão e no método `query` executamos o comando que irá realizar a inserção.

{Node.js}

```
var mysql = require('mysql');
```

Cláudio Luís V. Oliveira

```
var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "",
  database: "loja"
});

con.connect(function(erro) {
  if (erro) throw erro;
  var cmd = "INSERT INTO produto (codigo,
descricao, preco) VALUES (?, ?, ?)";
  var dados = [15, 'Cama de solteiro', 250];
  con.query(cmd, dados, function (erro) {
    if (erro) throw erro;
    console.log("Inserido!");
    con.end();
  });
});
```

inserir-produto.js

O próximo exemplo irá mostrar um exemplo de rotina de alteração realizada a partir do Node.js. Iremos executar o comando através do método query para aplicar um desconto de 10% para todos os produtos com preço maior que 2000.

{Node.js}

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "",
```

```

    database: "loja"
  });

  con.connect(function(erro) {
    if (erro) throw erro;
    var cmd = "UPDATE produto SET preco =
preco * 0.9 WHERE preco > 2000.00";
    con.query(cmd, function (erro, resultado)
    {
      if (erro) throw erro;
      console.log("O desconto foi aplicado
para " + resultado.affectedRows + "
produtos.");
      con.end();
    });
  });
});

```

alterar-produto.js

Concluindo os exemplos de operações básicas no MariaDB a partir do Node.js, o exemplo a seguir irá realizar a exclusão do registro na tabela produto que possui o código igual a 10.

{Node.js}

```

var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "",
  database: "loja"
});

```

Cláudio Luís V. Oliveira

```
con.connect(function(erro) {  
  if (erro) throw erro;  
  var cmd = "DELETE FROM produto WHERE  
codigo = 10";  
  con.query(cmd, function (erro) {  
    if (erro) throw erro;  
    console.log("Produto apagado!");  
    con.end();  
  });  
});
```

.....
apagar-produto.js

Exercícios

- 1) Considerando o banco de dados loja, desenvolver os programas que irão permitir realizar consulta (a todos registros), inclusão, alteração e exclusão na tabela cliente apresentada a seguir.

Tabela: cliente			
<i>Nome do Campo</i>	<i>Tipo de Dados</i>	<i>Anulável</i>	<i>Chave Primária</i>
cpf	VARCHAR(14)	Não	Sim
nome	VARCHAR(40)	Não	Não
telefone	VARCHAR(20)	Não	Não

- 2) Considerando o banco de dados secretaria e a tabela aluno, que contém nome (texto, 40), ra (texto, 15), média (float) e situação acadêmica (texto, 10), sendo

todos os campos obrigatórios, elaborar os programas em Node.js que irão realizar consulta (a todos registros), inclusão, alteração e exclusão na respectiva tabela.

Aplicação web com acesso ao MariaDB

Neste projeto vamos unir os conceitos de aplicações para a Internet e acesso à banco de dados para criar uma aplicação web que irá acessar e manipular os dados armazenados em banco de dados.

O primeiro passo consiste em instalar o módulo `mysql` e o framework `Express`, para isso vá até a pasta da aplicação `Node.js` que será criada e use o comando `npm` conforme mostrado a seguir.



```
cd suaaplicacao  
npm install mysql --save  
npm install express --save
```

Na pasta criada devemos implementar o arquivo `principal.html` apresentado a seguir que será responsável por apresentar os links para as páginas relacionadas a produtos e clientes.



```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>Loja</title>  
    <meta charset="UTF-8">  
    <meta name="viewport"  
content="width=device-width, initial-  
scale=1.0">
```



```

    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/boo
tstrap/4.3.1/css/bootstrap.min.css">
  </head>
  <body>
    <div class="container">
      <div class="jumbotron">
        <h1>Loja</h1>
      </div>
      <p><a
href="/produtos">Produtos</a></p>
      <p><a href="#">Clientes</a></p>
    </div>
  </body>
</html>

```

principal.html

O código-fonte inicialmente irá definir os elementos necessários à criação do servidor da aplicação, além dos dados para acesso ao banco de dados. Na sequência é realizada a conexão e a resposta à requisição da página raiz do site.

{Node.js}

```

ar fs = require('fs');
var http = require("http");
var express = require('express');
var mysql = require('mysql');
var app = express();
var bodyParser = require('body-parser');
var urlencodedParser =
bodyParser.urlencoded({ extended: true });

```

Cláudio Luís V. Oliveira

```
// https://icons.getbootstrap.com/
var caneta = 'obter ícone no formato svg';
var lixo = 'obter ícone no formato svg';

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "",
  database: "loja"
});

con.connect(function(erro) {
  var servidor = app.listen(8080, function()
  {
    var porta = servidor.address().port;
    console.log("Servidor executando na
    porta %s", porta);
  });

  app.get('/', function (req, res) {
    fs.readFile('principal.html',
function(erro, dado) {
    res.writeHead(200, {'Content-Type':
'text/html'});
    res.write(dado);
    res.end();
  });
});

  // Implementar o restante do programa
  // a partir deste ponto.
});
```

loja1.js (parcial)

Produtos cadastrados

O arquivo [produtos.html](#) contém a estrutura básica da página que irá exibir os dados dos produtos que estão cadastrados no banco de dados.



```

<!DOCTYPE html>
<html>
  <head>
    <title>Loja</title>
    <meta charset="UTF-8">
    <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/boo
tstrap/4.3.1/css/bootstrap.min.css">
  </head>
  <body>
    <div class="container">
      <div class="jumbotron">
        <h1>Produtos cadastrados</h1>
      </div>
      <p><a href="/novo-produto" class="btn
btn-primary btn-sm">Novo produto...</a></p>
      {{tabela}}
    </div>
  </body>
</html>

```

produtos.html

A rotina a seguir irá ser executada quando ocorrer quando o servidor receber uma requisição da página de produtos. Observe que a rotina irá realizar uma consulta e obter todos os registros cadastrados na tabela produto. Em seguida, a partir dos dados obtidos é montada uma tabela HTML que posteriormente será inserida na página [produtos.html](#).

{Node.js}

```
app.get('/produtos', function (req, res) {
  fs.readFile('produtos.html',
function(erro, dado) {
  con.query("SELECT * FROM produto",
function (erro, resultado) {
  if (erro) throw erro;
  var tabela = "<table class='table
table-hover'>" +
    "<tr class='table-primary'>" +
    "<td>Operações</td><td>Código</td>"
+
    "<td>Descrição</td>" +
    "<td>Preço</td></tr>";
  for (var i = 0; i < resultado.length;
i++) {
    tabela += "<tr><td>" +
      "<a href='/editar-produto?cod=" +
resultado[i].codigo +
      "' class='btn btn-primary btn-
sm'>" + caneta + "</a> " +
      "<a href='/apagar-produto?cod=" +
resultado[i].codigo +
```

```

        "' class='btn btn-primary btn-
sm'" + lixo + "</a></td>" +
        "<td>" + resultado[i].codigo +
"</td>" +
        "<td>" + resultado[i].descricao +
"</td>" +
        "<td align='right'" +
resultado[i].preco.toFixed(2) + "</td>";
        "</tr>";
    }
    tabela += "</table>";
    dado = dado.toString().replace(
"{{tabela}}", tabela);
    res.writeHead(200, {'Content-Type':
'text/html'});
    res.write(dado);
    res.end();
    });
    });
});

```

loja1.js (continuação)

Na Figura 10 apresentamos um exemplo sobre como será o aspecto da página que foi criada.

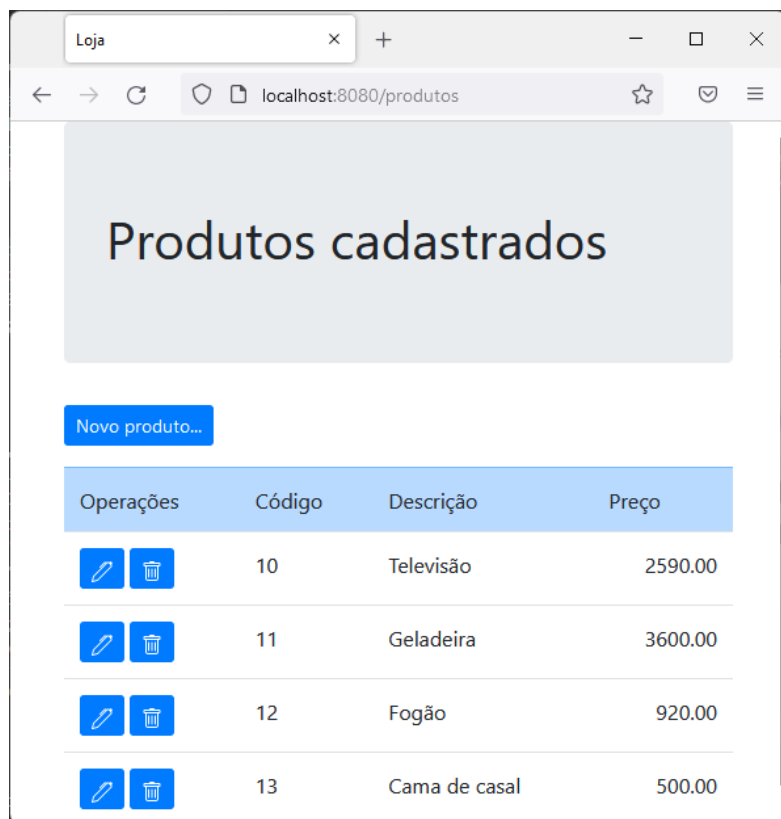


Figura 10: Relação de produtos

Inclusão de produtos

O arquivo novo-produto.html irá conter o formulário que possibilitará a digitação dos dados dos produtos que serão incluídos na respectiva tabela no banco de dados.



```
<!DOCTYPE html>
```

```

<html>
  <head>
    <title>Loja</title>
    <meta charset="UTF-8">
    <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/boo
tstrap/4.3.1/css/bootstrap.min.css">
  </head>
  <body>
    <div class="container">
      <div class="jumbotron">
        <h1>Novo produto</h1>
      </div>
      <form id="form" method="POST"
action="/incluir-produto">
        <div class="form-group">
          Código:
          <input type="text" name="codigo"
class="form-control" maxlength="3" />
        </div>
        <div class="form-group">
          Descrição:
          <input type="text"
name="descricao" class="form-control"
maxlength="20" />
        </div>
        <div class="form-group">
          Preço Unitário:
          <input type="text" name="preco"
class="form-control" maxlength="10" />
        </div>
      </form>
    </div>
  </body>
</html>

```

```
        <input type="submit" value="Salvar"
class="btn btn-primary" />
    </form>
</div>
</body>
</html>
```

novo-produto.html

Ao servidor criado em Node.js serão adicionadas as rotinas que irá tratar a requisição da página que implementa o formulário e a requisição que deverá realizar a inclusão dos dados quando o botão Salvar do formulário for clicado.

{Node.js}

```
app.get('/novo-produto', function (req, res)
{
    fs.readFile('novo-produto.html',
function(erro, dado) {
    res.writeHead(200, {'Content-Type':
'text/html'});
    res.write(dado);
    res.end();
});
});

app.post('/incluir-produto',
urlencodedParser, function (req, res){
    var cmd = "INSERT INTO produto (codigo,
descricao, preco) VALUES (?, ?, ?)";
    var dados = [req.body.codigo,
req.body.descricao, req.body.preco];
    con.query(cmd, dados, function (erro) {
        if (erro) throw erro;
```



```

    res.redirect('/produtos');
  });
});

```

loja1.js (continuação)

Neste trecho do programa é importante notar o uso do método res.redirect que fará que a página que apresenta os produtos cadastrados seja exibida no navegador ao término da inclusão.

Alteração de produtos

O arquivo editar-produto.html irá implementar o formulário que irá permitir a alteração dos dados dos produtos. Observe que o conteúdo é muito parecido com o formulário criado para a inclusão dos produtos. A única diferença é que na propriedade value das caixas de texto (código, descrição e preço) deixamos marcações para, posteriormente, a rotina em Node.js substituir pelos dados do registro que será alterado.



```

<!DOCTYPE html>
<html>
  <head>
    <title>Loja</title>
    <meta charset="UTF-8">
    <meta name="viewport"
content="width=device-width, initial-
scale=1.0">

```

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/boo
tstrap/4.3.1/css/bootstrap.min.css">
</head>
<body>
  <div class="container">
    <div class="jumbotron">
      <h1>Editar produto</h1>
    </div>
    <form id="form" method="POST"
action="/alterar-produto">
      <div class="form-group">
        Código:
        <input type="text" name="codigo"
class="form-control" maxlength="3"
value="{{codigo}}" readonly/>
      </div>
      <div class="form-group">
        Descrição:
        <input type="text"
name="descricao" class="form-control"
maxlength="20" value="{{descricao}}" />
      </div>
      <div class="form-group">
        Preço Unitário:
        <input type="text" name="preco"
class="form-control" maxlength="10"
value="{{preco}}" />
      </div>
      <input type="submit" value="Salvar"
class="btn btn-primary" />
    </form>
  </div>
</body>
```

</html>

editar-produto.html

No programa em Node.js vamos acrescentar a rotina que irá mostrar o formulário já com os dados, do produto que será alterado, devidamente preenchidos. A partir do código que foi passado na requisição (query string), ocorre a pesquisa no banco de dados e, por fim, o conteúdo das caixas de texto são preenchidos com os dados obtidos pela pesquisa.

{Node.js}

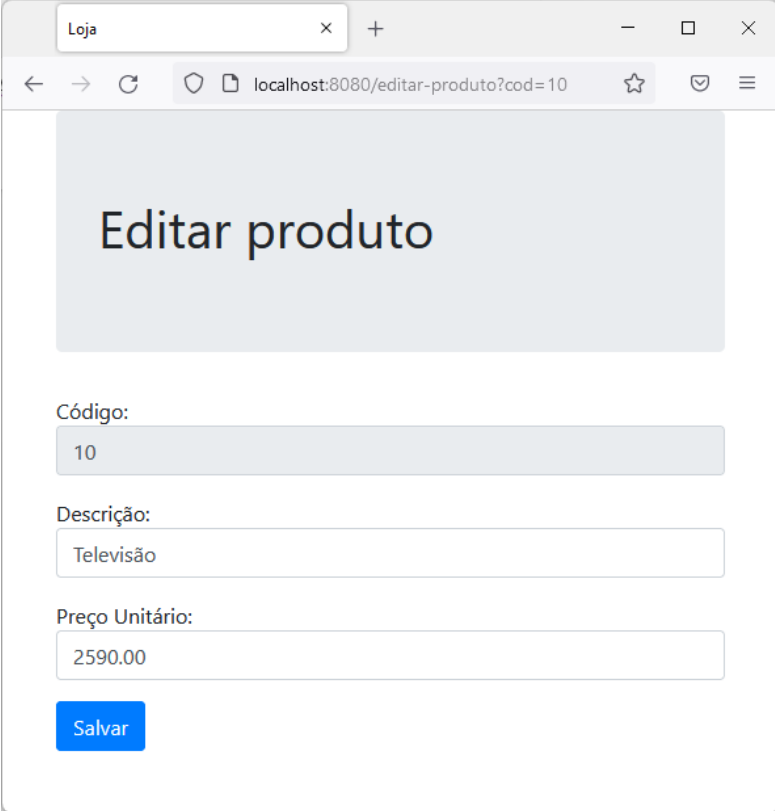
```
app.get('/editar-produto', function (req,
res) {
  fs.readFile('editar-produto.html',
function(erro, dado) {
  var codigo = req.query.cod;
  var descricao;
  var preco;
  con.query("SELECT * FROM produto WHERE
codigo=" + codigo, function (erro,
resultado) {
    if (erro) throw erro;
    if (resultado.length > 0) {
      descricao = resultado[0].descricao;
      preco =
resultado[0].preco.toFixed(2);
    }
    res.writeHead(200, {'Content-Type':
'text/html'});
    dado =
dado.toString().replace("{{codigo}}",
codigo);
```

Cláudio Luís V. Oliveira

```
        dado =  
        dado.toString().replace("{{descricao}}",  
        descricao);  
        dado =  
        dado.toString().replace("{{preco}}", preco);  
        res.write(dado);  
        res.end();  
    });  
});  
});
```

loja1.js (continuação)

Na Figura 11 apresentamos um exemplo do formulário com os campos já preenchidos a partir dos dados recuperados através da consulta ao banco de dados.



Loja

localhost:8080/editar-produto?cod=10

Editar produto

Código:

Descrição:

Preço Unitário:

Salvar

Figura 11: Alteração dos dados do produto

A rotina seguinte irá realizar a alteração dos dados na tabela produto, sendo que ela será executada quando o usuário clicar no botão Salvar.

{Node.js}

```
app.post('/alterar-produto',
  urlencodedParser, function (req, res){
    var cmd = "UPDATE produto SET descricao=?,
preco=? WHERE codigo=?";
```

Cláudio Luís V. Oliveira

```
var dados = [req.body.descricao,  
req.body.preco, req.body.codigo];  
con.query(cmd, dados, function (erro) {  
  if (erro) throw erro;  
  res.redirect('/produtos');  
});  
});
```

loja1.js (continuação)

Exclusão de produtos

Complementando o programa temos a rotina que irá realizar a exclusão de um registro na tabela produto.

{Node.js}

```
app.get('/apagar-produto', function (req,  
res) {  
  var codigo = req.query.cod;  
  con.query("DELETE FROM produto WHERE  
codigo=" + codigo, function (erro,  
resultado) {  
    if (erro) throw erro;  
    res.redirect('/produtos');  
  });  
});
```

loja1.js (continuação)

Exercícios

- 1) Considerando o programa desenvolvido, adicionar as rotinas e páginas HTML que irão permitir a exibição dos clientes cadastrados, inclusão, alteração e exclusão. Utilizar a tabela cliente apresentada a seguir e devidamente criada no banco de dados loja.

Tabela: cliente			
<i>Nome do Campo</i>	<i>Tipo de Dados</i>	<i>Anulável</i>	<i>Chave Primária</i>
cpf	VARCHAR(14)	Não	Sim
nome	VARCHAR(40)	Não	Não
telefone	VARCHAR(20)	Não	Não

Uso do banco de dados MongoDB

Como mencionamos anteriorme, uma das grandes vantagens de processar o JavaScript no lado servidor, através do Node.js, é a possibilidade de acesso aos Sistemas Gerenciadores de Banco de Dados (SGBDs).

Vamos desenvolver uma aplicação que usará como banco de dados o MongoDB, que é um banco de dados do tipo **noSQL**, usando JSON (JavaScript Object Notation) para realizar o armazenamento dos dados. É um banco muito utilizado em conjunto com o Node.js, pois, como adota JSON, trabalha com a notação do próprio JavaScript.

Apenas para fins didáticos e com o intuito de facilitar o entendimento, comparando com um banco de dados relacional, as tabelas no MongoDB são chamadas de coleções e os registros são conhecidos como documentos. Mas devemos ter em mente que a estrutura das coleções e dos documentos não é rígida como a estrutura das tabelas e registros implementados nos bancos de dados relacionais.



Dica!

O **MongoDB Community Server** é gratuito, multiplataforma e pode ser baixado de <https://www.mongodb.com/download-center/community>

Durante a instalação do MongoDB você irá perceber que pode optar por realizar a instalação do mesmo como um serviço do sistema operacional ou ser ativado manualmente através da linha de comandos. Caso faça a opção por colocar o serviço no ar de forma manual, use o comando mostrado a seguir. Onde dados deve indicar o caminho completo para a pasta que irá armazenar os dados.



```
mongod -dbpath=dados
```

O comando mongo permite acessar a interface em modo texto do banco de dados.



```
mongo
```

A visualização dos bancos de dados criados no MongoDB pode ser realizada através do comando show databases.



```
show databases
```

A criação ou seleção de um banco de dados é feita com o comando use. Por exemplo, o comando mostrado abaixo irá para criar ou selecionar o banco chamado loja.



```
use loja
```

A visualização das coleções de um determinado banco de dados é realizada pelo comando show collections.

Cláudio Luís V. Oliveira

Lembrando que, em primeiro lugar, devemos usar o comando use, para selecionar o banco desejado.



```
show collections
```

Considerando que acabamos de criar o banco loja, o resultado do comando show collections, neste momento, irá produzir um resultado vazio.

Inserção

No MongoDB não é necessário criarmos previamente a coleção. Desta forma, basta iniciarmos a inserção dos dados a serem armazenados, que a coleção será automaticamente criada. Desta maneira, no exemplo a seguir, vamos criar uma coleção chamada produtos e já inserir o primeiro documento utilizando o método insertOne.



```
db.produtos.insertOne({ codigo: 10,  
descricao: "Televisor", preco: 1990.00 })
```

Também é possível inserir vários documentos a partir de um único insert aplicando o conceito de array, neste caso

devemos usar o método insertMany, conforme ilustra o próximo exemplo.



```
db.produtos.insertMany([ { codigo: 11,  
  descricao: "Geladeira", preco: 2990.00 }, {  
  codigo: 12, descricao: "Fogao", preco:  
  840.00 }, { codigo: 13, descricao:  
  "Computador", preco: 3500.00 } ])
```

Consulta

O método find é usado para consultarmos os documentos que estão armazenados em determinada coleção. O exemplo a seguir irá mostrar todos os documentos que existem na coleção produtos.



```
db.produtos.find()
```

Na Figura 12 mostramos o resultado da execução do método find, considerando a inserção dos dados realizada nos exemplos anteriores.

```
Prompt de Comando - mongo
> db.produtos.find()
{ "_id" : ObjectId("5cd210696dea46ecf422c2fb"), "codigo" : 10, "descricao" : "Televisor", "preco" : 1900 }
{ "_id" : ObjectId("5cd49c9b4bec73e3ba8e4218"), "codigo" : 11, "descricao" : "Geladeira", "preco" : 2990 }
{ "_id" : ObjectId("5cd49c9b4bec73e3ba8e4219"), "codigo" : 12, "descricao" : "Fogao", "preco" : 840 }
{ "_id" : ObjectId("5cd49c9b4bec73e3ba8e421a"), "codigo" : 13, "descricao" : "Computador", "preco" : 3500 }
```

Figura 12: Exibição dos dados processados

No método find também podemos passar parâmetros que irão permitir filtrar os documentos que serão mostrados no resultado. Por exemplo, o comando a seguir irá exibir apenas o documento que apresenta o código igual a 12.



```
db.produtos.find({"codigo": 12})
```

Outra possibilidade é usar na pesquisa um dos operadores de operação que podem ser \$eq (igual), \$gt (maior), \$gte (maior ou igual), \$in (valores especificados em um array), \$lt (menor), \$lte (menor ou igual), \$ne (diferente) ou \$nin (nenhum dos valores especificados em um array). Aplicando estes conceitos, no próximo exemplo iremos mostrar os produtos com preço maior que 2000.



```
db.produtos.find({preco: {$gt: 2000}})
```

Observe a seguir que vamos utilizar o operador \$in para mostrar os documentos que tem código igual a 10 ou 11.



```
db.produtos.find({codigo: {$nin: [10, 11]}})
```

Alteração

O método updateOne deve ser usado quando pretendemos mudar alguma informação do armazenada no documento, por exemplo, a seguir vamos alterar o valor do preço do produto que tem o código igual a 12.



```
db.produtos.updateOne({codigo: 12}, { $set:  
{ preco: 790.00 } })
```

Outra opção é usar o método updateMany quando precisamos realizar a alteração simultânea de vários documentos.

Exclusão

Quando precisamos apagar um documento da coleção devemos usar o método deleteOne. No próximo

Cláudio Luís V. Oliveira

exemplo vamos remover o documento que possui o código igual a 12.



```
db.produtos.deleteOne({codigo: 12})
```

O método deleteMany deverá ser empregado quando desejamos excluir vários documentos.

Aplicação Node.js com acesso ao MongoDB

Uma aplicação Node.js pode facilmente acessar um banco de dados do MongoDB. O primeiro passo é instalar o drive, para isso vá até a pasta da aplicação Node.js que será criada e use o comando npm conforme mostrado a seguir.



```
cd suaaplicacao  
npm install mongodb --save
```

Em primeiro lugar vamos mostrar uma aplicação simples que irá exibir, no próprio console, os documentos armazenados na coleção produtos, criada anteriormente. Observe que iremos realizar a conexão ao MongoDB, para isso devemos especificar o endereço (url) do servidor.

A acesso à um banco de dados deve ser realizado através de rotinas assíncronas, pois, como se trata de um serviço de rede, não é possível para a aplicação garantir o

sucesso da conexão e os tempos de resposta. Desta maneira, precisamos criar a função `listar()` usando `async`.

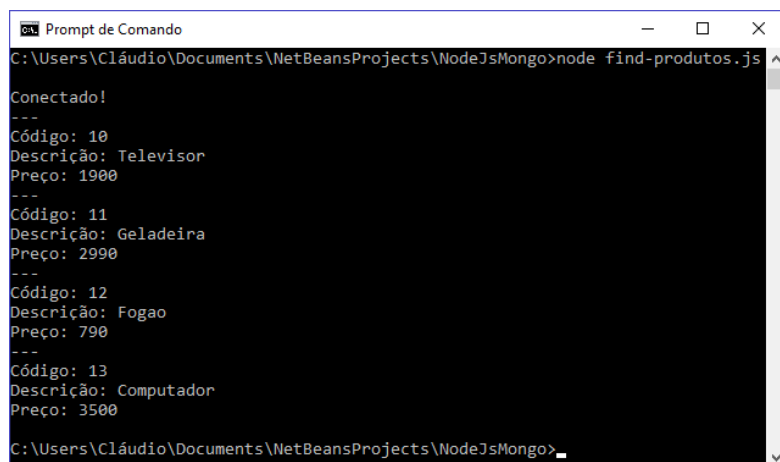
{Node.js}

```
const MongoClient =
require('mongodb').MongoClient;
const url = 'mongodb://localhost:27017';
const cliente = new MongoClient(url, {
  useNewUrlParser: true,
  useUnifiedTopology: true });

async function listar() {
  try {
    await cliente.connect();
    console.log("Conectado!");
    const banco = cliente.db('loja');
    var cursor =
      banco.collection('produtos').find();
    await cursor.forEach(function(doc) {
      console.log("---");
      console.log("Código: " + doc.codigo);
      console.log("Descrição: " +
doc.descricao);
      console.log("Preço: " + doc.preco);
    });
  }
  finally {
    await cliente.close();
  }
}
listar();
```

find-produtos.js

Após a conexão é realizada a seleção do banco que, no nosso exemplo, é loja utilizando, para isso, o método cliente.db. Para obtermos o conteúdo de uma coleção devemos criar um cursor que irá receber obter o retorno do método find. Concluindo o exemplo, o método forEach obtém os documentos recuperados. Observe que os métodos de acesso ao banco de dados são usados sempre com o uso do await, que pausa a execução da função assíncrona e espera pela conclusão do método em execução. A execução da aplicação deverá produzir um resultado similar ao mostrado pela Figura 13.



```

C:\Users\Cláudio\Documents\NetBeansProjects\NodeJsMongo>node find-produtos.js

Conectado!
---
Código: 10
Descrição: Televisor
Preço: 1900
---
Código: 11
Descrição: Geladeira
Preço: 2990
---
Código: 12
Descrição: Fogao
Preço: 790
---
Código: 13
Descrição: Computador
Preço: 3500
C:\Users\Cláudio\Documents\NetBeansProjects\NodeJsMongo>
```

Figura 13: Exibição do conteúdo da coleção produtos

Manipulação de documentos através do Node.js

A aplicação seguinte irá inserir um documento na coleção produtos. Note o uso do MongoClient para passarmos as informações para conexão ao banco de dados. Em seguida, no método connect do cliente realizamos a seleção do banco de dados e executamos a inserção, através do método insertOne, na coleção desejada.

{Node.js}

```
const MongoClient =
require('mongodb').MongoClient;
const url = 'mongodb://localhost:27017';
const cliente = new MongoClient(url, {
  useNewUrlParser: true,
  useUnifiedTopology: true });

async function inserir() {
  try {
    await cliente.connect();
    console.log("Conectado!");
    const banco = cliente.db('loja');
    await banco.collection('produtos')
      .insertOne({ codigo: 15,
        descricao: 'Cama de solteiro',
        preco: 250
      });
  }
  finally {
    console.log("Inserido!");
    await cliente.close();
  }
}
```

```
inserir());
```

insert-produto.js

O próximo exemplo irá mostrar um exemplo de rotina de alteração realizada a partir do Node.js. Neste caso, iremos utilizar o método `updateMany` para aplicar um desconto de 10% para todos os produtos com preço maior que 2000.

{Node.js}

```
const MongoClient =
require('mongodb').MongoClient;
const url = 'mongodb://localhost:27017';
const cliente = new MongoClient(url, {
  useNewUrlParser: true,
  useUnifiedTopology:true });

async function aplicarDesconto() {
  try {
    await cliente.connect();
    console.log("Conectado!");
    const banco = cliente.db('loja');
    await banco.collection('produtos')
      .updateMany(
        { preco: { $gt: 2000.00 } },
        { $mul: { preco: 0.9 } }
      );
  }
  finally {
    console.log("Alterado!");
    await cliente.close();
  }
}
```

```
}
```

```
aplicarDesconto());
```

update-produto.js

Concluindo os exemplos de operações básicas no MongoDB a partir do Node.js, o exemplo a seguir irá realizar a exclusão de um documento da coleção produtos, que possui o código igual a 10. Para realizar esta operação, vamos utilizar o método deleteOne.

{Node.js}

```
const MongoClient =
require('mongodb').MongoClient;
const url = 'mongodb://localhost:27017';
const cliente = new MongoClient(url, {
  useNewUrlParser: true,
  useUnifiedTopology: true });

async function apagar() {
  try {
    await cliente.connect();
    console.log("Conectado!");
    const banco = cliente.db('loja');
    await banco.collection('produtos')
      .deleteOne( {codigo: 10}
    );
  }
  finally {
    console.log("Apagado!");
    await cliente.close();
  }
}
```

Cláudio Luís V. Oliveira

apagar();

delete-produto.js

Aplicação web e acesso ao MongoDB

Neste projeto vamos unir os conceitos estudados até o momento e criar uma aplicação web que irá acessar e manipular os dados armazenados em uma coleção.

Estrutura de arquivos do projeto

Em primeiro lugar vamos instalar o driver para acesso ao MongoDB e o framework Express, ambos já utilizados anteriormente. Também vamos instalar o Embedded JavaScript templating (EJS), que consiste em um framework para visualização que oferece uma maneira fácil de transportar dados do lado do servidor para o cliente da aplicação.



```
cd suaaplicacao  
npm install mongodb --save  
npm install express --save  
npm install ejs --save
```

Após criar a pasta do projeto crie também as pastas public e views. Dentro da pasta public faça a criação das pastas images e scripts, conforme podemos observar na Figura 14. Também escolha e copie duas imagens que irão indicar as operações para editar e deletar.

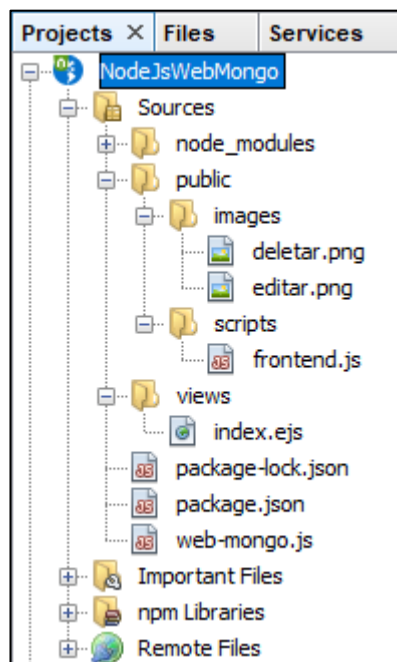


Figura 14: Estrutura de pastas do projeto

Página principal

A página HTML da aplicação (index.ejs), que é apresentada a seguir, deverá ser armazenada dentro da pasta views e deverá possuir a extensão ejs, pois, como já mencionado, estamos empregando o framework Embedded JavaScript templating (EJS) neste projeto.



```
<!DOCTYPE html>
<html>
  <head>
```

```

<title>Produtos</title>
<meta charset="UTF-8">
<meta name="viewport"
content="width=device-width, initial-
scale=1.0">
  <!-- Bootstrap -->
  <!-- Versão minimizada do CSS -->
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/boo
tstrap/4.3.1/css/bootstrap.min.css">

  <!-- Versão minimizada do JQuery -->
  <script
src="https://code.jquery.com/jquery-
3.3.1.slim.min.js"></script>

  <!--AJAX-jQuery -->
  <script
src="https://ajax.googleapis.com/ajax/libs/j
query/3.3.1/jquery.min.js"></script>

  <!-- Versão minimizada do Popper -->
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/
popper.js/1.14.7/umd/popper.min.js"></script>
>

  <!-- Versão minimizada do Bootstrap -->
  <script
src="https://stackpath.bootstrapcdn.com/boot
strap/4.3.1/js/bootstrap.min.js"></script>

  <script src="scripts/frontend.js"
type="application/javascript"></script>

```

```
</head>
<body>
  <div class="container">
    <h1>Produtos</h1>
    <input type="button" class="btn btn-
primary" value="Novo..."
onclick="incluir()">
    <table class="table table-hover">
      <tr class="table-primary">
        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>Código</td>
        <td>Descrição</td>
        <td>Preço</td>
      </tr>
      <% dados.forEach(function(item) { %>
      <tr>
        <td>
          )" alt="Editar"/>
        </td>
        <td>
          )" alt="Deletar"/>
        </td>
        <td><%= item.codigo %></td>
        <td><%= item.descricao %></td>
        <td align="right"><%=
item.preco.toFixed(2) %></td>
      </tr>
      <% }) %>
    </table>
```



```

    <!-- Modal: Formulário -->
    <%- include('editar-produto.ejs'); -%>
    <!-- Modal: Confirmação -->
    <%- include('excluir-produto.ejs'); -
%>
  </div>
</body>
</html>

```

index.ejs

Basicamente a página carrega scripts e folhas de estilo que são necessárias ao funcionamento do jQuery e do Bootstrap. Em seguida, temos a tabela que será responsável pela exibição dos dados da coleção produtos que está armazenada no MongoDB. Por fim, usamos a instrução include do EJS para obtermos duas caixas de diálogo, uma para a edição e inclusão de produtos e outra para realizar a confirmação da exclusão de um item da coleção. Estas caixas de diálogo serão criadas posteriormente em arquivos separados.

Conforme destaca o trecho de programa a seguir, note o uso do EJS que irá receber um conjunto de dados que serão enviados pelo servidor (Node.js) e os renderizar no cliente. Os elementos do EJS são delimitados por <% %> e apresentam, como vantagem, o fato de serem especificados em JavaScript, o que facilita muito a criação das páginas, como exemplo, observe o uso do método toFixed() para realizar a formatação do preço.



```

<% dados.forEach(function(item) { %>
  <tr>
    <td>

```

Cláudio Luís V. Oliveira

```
        )" alt="Editar"/>
    </td>
    <td>
        )" alt="Deletar"/>
    </td>
    <td><%= item.codigo %></td>
    <td><%= item.descricao %></td>
    <td align="right"><%=
item.preco.toFixed(2) %></td>
</tr>
<% }> %>
```

Em seguida, na pasta views do projeto vamos implementar a página editar-produto.ejs. Ela irá conter a caixa de diálogo que irá possibilitar a edição dos dados do produto e será usada durante o processo de inclusão e alteração de produtos.

```
<div class="modal" id="caixa-dialogo">
    <div class="modal-dialog modal-sm">
        <div class="modal-content">
            <div class="modal-header">
                <h4 class="modal-title">Cadastro de
Produto</h4>
                <button type="button" class="close"
data-dismiss="modal">&times;</button>
            </div>
            <form id="form-produto">
                <input type="hidden" id="oper"
value="i" />
```

```

<div id="mensagem" class="modal-
body">
    <div class="form-group">
        Código:
        <input type="text" id="cod"
class="form-control" maxlength="3" />
    </div>
    <div class="form-group">
        Descrição:
        <input type="text" id="desc"
class="form-control" maxlength="20" />
    </div>
    <div class="form-group">
        Preço Unitário:
        <input type="text" id="preco"
class="form-control" maxlength="10" />
    </div>
</div>
<div class="modal-footer">
    <button type="submit" class="btn
btn-success">Salvar</button>
    <button type="button" class="btn
btn-danger" data-
dismiss="modal">Cancelar</button>
</div>
</form>
</div>
</div>
</div>

```

editar-produto.ejs

Também na pasta views vamos criar o arquivo excluir-produto.ejs. Ele será responsável pela criação de

Cláudio Luís V. Oliveira

uma caixa de mensagem que permitirá ao usuário confirmar ou não a exclusão de determinado produto.

```
.....
<div class="modal" id="caixa-confirmacao">
  <div class="modal-dialog modal-sm">
    <div class="modal-content">
      <div class="modal-header">
        <h4 class="modal-title">Atenção</h4>
        <button type="button" class="close"
data-dismiss="modal">&times;</button>
      </div>
      <div id="mensagem" class="modal-body">
        Você tem certeza que deseja apagar
este produto?
      </div>
      <div class="modal-footer">
        <button type="button" class="btn
btn-success" id="sim-deletar">Sim</button>
        <button type="button" class="btn
btn-danger" data-
dismiss="modal">Não</button>
      </div>
    </div>
  </div>
</div>
</div>
.....
```

excluir-produto.ejs

Desenvolvimento do servidor

Prosseguindo com o desenvolvimento da aplicação, vamos criar o servidor em Node.js que irá realizar o processamento das requisições, o acesso e a manipulação da coleção armazenada no MongoDB.

{Node.js}

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();

const MongoClient =
  require('mongodb').MongoClient;
const assert = require('assert');
const url = 'mongodb://localhost:27017';
const cliente = new MongoClient(url, {
  useNewUrlParser: true, useUnifiedTopology:
  true });

cliente.connect(function(erro) {
  assert.equal(null, erro);
  const banco = cliente.db('loja');

  var servidor = app.listen(8080, function()
  {
    var porta = servidor.address().port;
    console.log("Servidor executando na
    porta %s", porta);
  });

  app.use(express.static("public"));
  app.use(bodyParser.json());
```

Cláudio Luís V. Oliveira

```
app.use(bodyParser.urlencoded({ extended:
true }));
app.set('view engine', 'ejs');

app.get('/', (req, res) => {
  banco.collection('produtos').find().toArray(
    (erro, resultado) => {
      assert.equal(null, erro);
      res.render('index.ejs', { dados:
resultado });
    });
  });

  app.post("/produtos/incluir",
function(req, res){
  var produto = req.body;
  console.log('Incluir produto: ' +
JSON.stringify(produto));

  banco.collection('produtos').insertOne(produ
to, function(erro, res) {
    assert.equal(null, erro);
    console.log("1 documento inserido.");
  });
  return res.send(produto);
});

  app.post("/produtos/editar", function(req,
res) {
    var produto = req.body;
    console.log('Alterar produto: ' +
JSON.stringify(produto));
```

```
banco.collection('produtos').updateOne({codigo: produto.codigo}, {$set: produto},  
function(erro, res) {  
    assert.equal(null, erro);  
    console.log("1 documento alterado.");  
});  
return res.send(produto);  
});
```

```
app.post("/produtos/deletar",  
function(req, res){  
    var produto = req.body;  
    console.log('Excluir produto: ' +  
JSON.stringify(produto));
```

```
banco.collection('produtos').deleteOne(produto,  
function(erro, res) {  
    assert.equal(null, erro);  
    console.log("1 documento excluído.");  
});  
return res.send(produto);  
});
```

```
app.post("/produtos/getproduto",  
function(req, res){  
    var produto = req.body;  
    console.log('Produto: ' +  
JSON.stringify(produto));
```

```
banco.collection('produtos').findOne(produto,  
function(erro, item) {  
    assert.equal(null, erro);  
    return res.send(item);
```

```
    });  
  });  
});
```

web-mongo.js

Analisando o programa, observe, no trecho de programa a seguir, que vamos definir a configuração do servidor. Observe que a troca que dados entre as páginas será realizada usando JSON e também iremos utilizar o EJS como view engine.

{Node.js}

```
app.use(express.static("public"));  
app.use(bodyParser.json());  
app.use(bodyParser.urlencoded({ extended:  
true }));  
app.set('view engine', 'ejs');
```

Na sequência do programa, mostrada a seguir, definimos a ação que será realizada quando o cliente (navegador) solicitar a página raiz do servidor. O método find irá retornar os documentos cadastrados na coleção produtos e o resultado obtido, será renderizado pelo EJS juntamente com a página index.ejs.

{Node.js}

```
app.get('/', (req, res) => {  
  banco.collection('produtos').find().toArray(  
    (erro, resultado) => {  
      assert.equal(null, erro);  
      res.render('index.ejs', { dados:  
resultado });  
    }  
  )  
})
```



```
});  
});
```

Na Figura 15 temos um exemplo da página exibindo os documentos carregados de uma coleção implementada no MongoDB. Observe o botão “Novo...” o qual será responsável pela inclusão de produtos e os ícones para edição e exclusão de determinado produto.

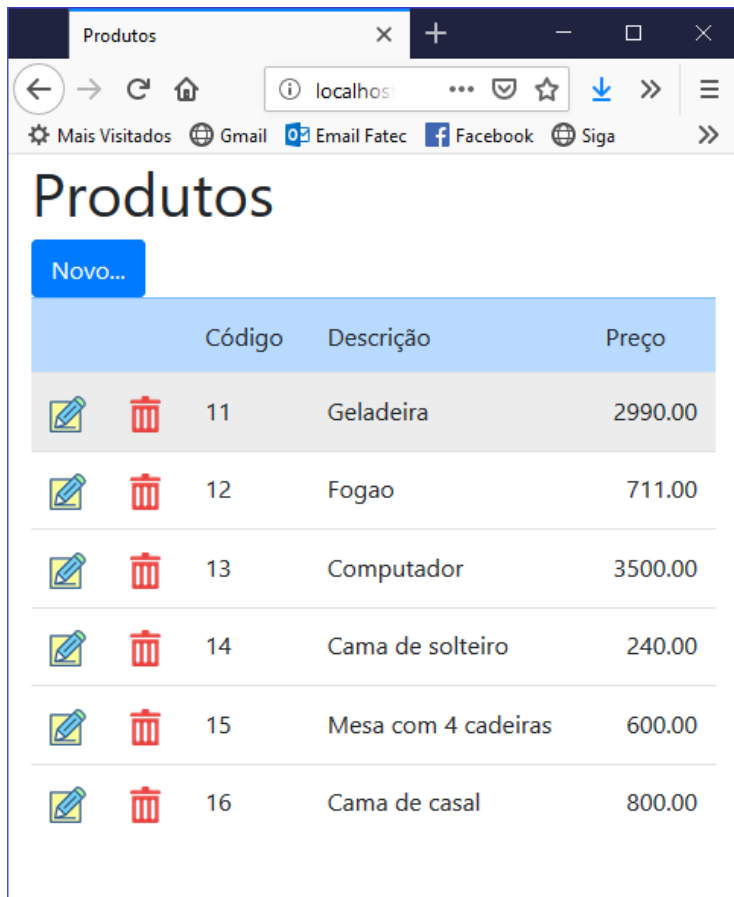


Figura 15: Exibição do conteúdo da coleção produtos

No trecho de programa a seguir é apresentada, em detalhes, a rotina que será responsável pela inclusão de um documento à coleção produto. Quando o servidor receber a requisição (`/produtos/incluir`), obtemos os dados enviados pelo cliente através da propriedade `req.body`, que está especificada em JSON. Na sequência, o método `insertOne` irá inserir os dados recebidos na coleção produtos.

{Node.js}

```
app.post("/produtos/incluir", function(req,
res){
    var produto = req.body;
    console.log('Incluir produto: ' +
JSON.stringify(produto));

    banco.collection('produtos').insertOne(produ
to, function(erro, res) {
        assert.equal(null, erro);
        console.log("1 documento inserido.");
    });
    return res.send(produto);
});
```

Em linhas gerais, as rotinas para atualização dos dados de um documento (`/produtos/editar`) e exclusão de um documento (`/produtos/deletar`), irão funcionar de maneira similar à inclusão, apenas alterando os respectivos métodos para `updateOne` e `deleteOne`.

Também temos a rotina, mostrada a seguir, que será responsável por obter um documento e enviá-lo para a página HTML, pois na rotina de alteração, precisamos mostrar os dados para que o usuário possa editá-los.

{Node.js}

```

app.post("/produtos/getproduto",
function(req, res){
    var produto = req.body;
    console.log('Produto: ' +
JSON.stringify(produto));

    banco.collection('produtos').findOne(produto
, function(erro, item) {
    assert.equal(null, erro);
    return res.send(item);
    });
});

```

AJAX (Asynchronous JavaScript and XML)

Concluindo a aplicação, passamos para a implementação das funções JavaScript que irão atuar no lado do cliente (navegador). O arquivo, com estas funções, deverá ser colocado na pasta public/scripts, pois, caso contrário, não terá permissão do servidor para aplicação para ser executado. Em resumo, elas serão responsáveis por abrir as caixas de diálogo e também realizar a realizar as requisições em AJAX (Asynchronous JavaScript and XML) para o servidor de aplicação.

{JS}

```

var codigo;

function incluir() {

```

Cláudio Luís V. Oliveira

```
    $("#oper").val('i');
    $("#cod").val('');
    $("#desc").val('');
    $("#preco").val('');
    $('#caixa-dialogo').modal('show');
}

function editar(codigo) {
    var dados = {
        "codigo": parseInt(codigo)
    };
    $("#oper").val('e');
    $("#cod").val(codigo);
    $.ajax({
        type: "POST",
        contentType: "application/json",
        url: window.location +
"produtos/getproduto",
        data: JSON.stringify(dados),
        dataType: 'json',
        success: function(produto) {
            $("#desc").val(produto.descricao);

$("#preco").val(produto.preco.toFixed(2));
        },
        error: function(erro) {
            alert("ERRO: " + erro);
        }
    });

    $('#caixa-dialogo').modal('show');
}

function deletar(cod) {
```

```
    codigo = cod;
    $('#caixa-confirmacao').modal('show');
}

$(document).ready(function() {
    $("#sim-deletar").click(function(evento) {
        evento.preventDefault();
        var dados = {
            "codigo": parseInt(codigo)
        };

        $.ajax({
            type: "POST",
            contentType: "application/json",
            url: window.location +
"produtos/deletar",
            data: JSON.stringify(dados),
            dataType: 'json',
            success: function(produto) {
                $('#caixa-
confirmacao').modal('hide');
                window.location.reload();
            },
            error: function(erro) {
                alert("ERRO: " + erro);
            }
        });
    });
});

$("#form-produto").submit(function(evento)
{
    evento.preventDefault();
    var dados = {
        codigo: parseInt($("#cod").val()),
```

```
        descricao: $("#desc").val(),
        preco: parseFloat($("#preco").val())
    };
    var destino = window.location;
    if ($("#oper").val() === 'i')
        destino += "produtos/incluir";
    else if ($("#oper").val() === 'e')
        destino += "produtos/editar";

    $.ajax({
        type: "POST",
        contentType: "application/json",
        url: destino,
        data: JSON.stringify(dados),
        dataType: 'json',
        success: function(produto) {
            $('#caixa-dialogo').modal('hide');
            window.location.reload();
        },
        error: function(erro) {
            alert("ERRO: " + erro);
        }
    });
});
});
```

frontend.js

Detalhando o funcionamento do arquivo com as funções do JavaScript que irão ser executadas no cliente. Apresentamos, inicialmente, a função incluir que irá colocar o campo escondido (hidden) a operação que estão sendo realizada, ou seja, 'i' para indicar uma inclusão. Esta função também limpa os demais campos (código, descrição e preço)

carregando uma string vazia ("). Por fim, mostra a caixa de diálogo.

{JS}

```
function incluir() {
    $("#oper").val('i');
    $("#cod").val('');
    $("#desc").val('');
    $("#preco").val('');
    $('#caixa-dialogo').modal('show');
}
```

Na Figura 16, mostramos a caixa de diálogo que irá possibilitar a inclusão de um produto.

A imagem mostra uma interface web em um navegador. No topo, há uma barra de endereço com o endereço 'localhost' e ícones para Gmail, Email Fatec, Facebook e Siga. Abaixo, há uma caixa de diálogo intitulada 'Cadastro de Produto' com um botão de fechar no canto superior direito. Dentro da caixa, há três campos de entrada rotulados 'Código:', 'Descrição:' e 'Preço Unitário:'. Na base da caixa, há dois botões: 'Salvar' (verde) e 'Cancelar' (vermelho). Abaixo da caixa de diálogo, há uma barra de status com ícones de edição e exclusão, o número '16', o texto 'Cama de casal' e o valor '800.00'.

Figura 16: Caixa de diálogo para inclusão

No trecho de código-fonte mostrado a seguir temos a função que será executada no momento da submissão do formulário, ou seja, quando o botão “Salvar” da caixa de diálogo for clicado. A função em primeiro lugar obtém dos dados digitados e coloca em formato JSON, observe a variável `dados`. Na sequência, com base na operação (definida no campo escondido do formulário, determina se ocorrerá uma inclusão ou alteração. Depois a ocorre a requisição AJAX para o servidor Node.js que fará a inclusão ou alteração. A última etapa consiste em fechar a caixa de diálogo e carregar a página principal, através do método `window.location.reload()`, para que os dados incluídos ou alterados sejam mostrados na tabela.

{JS}

```
$("#form-produto").submit(function(evento) {  
    evento.preventDefault();  
    var dados = {  
        codigo: parseInt($("#cod").val()),  
        descricao: $("#desc").val(),  
        preco: parseFloat($("#preco").val())  
    };  
    var destino = window.location;  
    if ($("#oper").val() === 'i')  
        destino += "produtos/incluir";  
    else if ($("#oper").val() === 'e')  
        destino += "produtos/editar";  
  
    $.ajax({  
        type: "POST",  
        contentType: "application/json",  
        url: destino,  
        data: JSON.stringify(dados),
```



```

    dataType: 'json',
    success: function(produto) {
        $('#caixa-dialogo').modal('hide');
        window.location.reload();
    },
    error: function(erro) {
        alert("ERRO: " + erro);
    }
  });
});

```

A rotina de alteração, mostrada em detalhes a seguir, recebe como parâmetro o código do produto, realizar uma requisição AJAX que irá consultar o banco de dados e retornar os demais dados do produto que serão carregados no formulário. O campo escondido irá receber o valor 'e' para indicar que ocorrerá uma edição (alteração) dos dados.

{JS}

```

function editar(codigo) {
    var dados = {
        "codigo": parseInt(codigo)
    };
    $("#oper").val('e');
    $("#cod").val(codigo);
    $.ajax({
        type: "POST",
        contentType: "application/json",
        url: window.location +
"produtos/getproduto",
        data: JSON.stringify(dados),
        dataType: 'json',
        success: function(produto) {

```

Cláudio Luís V. Oliveira

```
$("#desc").val(produto.descricao);

$("#preco").val(produto.preco.toFixed(2));
},
error: function(erro) {
    alert("ERRO: " + erro);
}
});

$('#caixa-dialogo').modal('show');
}
```

Na Figura 17 temos a caixa de diálogo exibindo os dados que serão editados. Ao clicar no botão “Salvar”, ocorrerá a submissão do formulário, que fará a requisição ao servidor Node.js que fará a alteração dos dados.

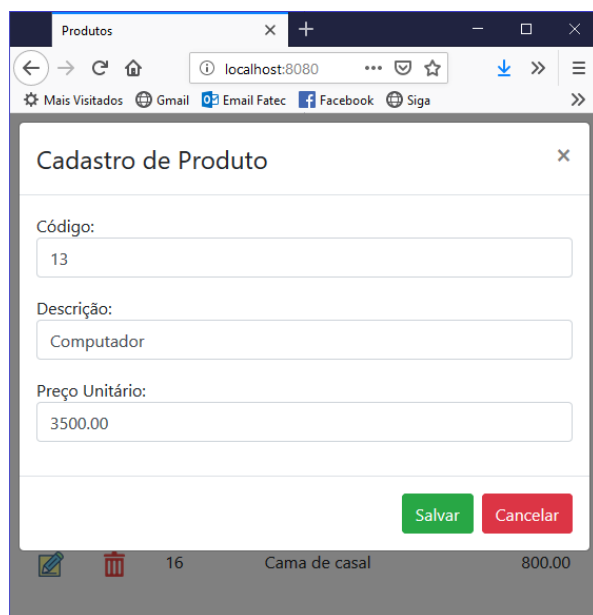


Figura 17: Caixa de diálogo para alteração

Ao clicar no ícone da lata de lixo, que é mostrado antes de cada produto exibido na página principal, devemos realizar a exclusão do respectivo documento armazenado no MongoDB. Porém, para evitar uma exclusão acidental, antes mostramos uma caixa de diálogo onde o usuário poderá confirmar a operação (Figura 18).

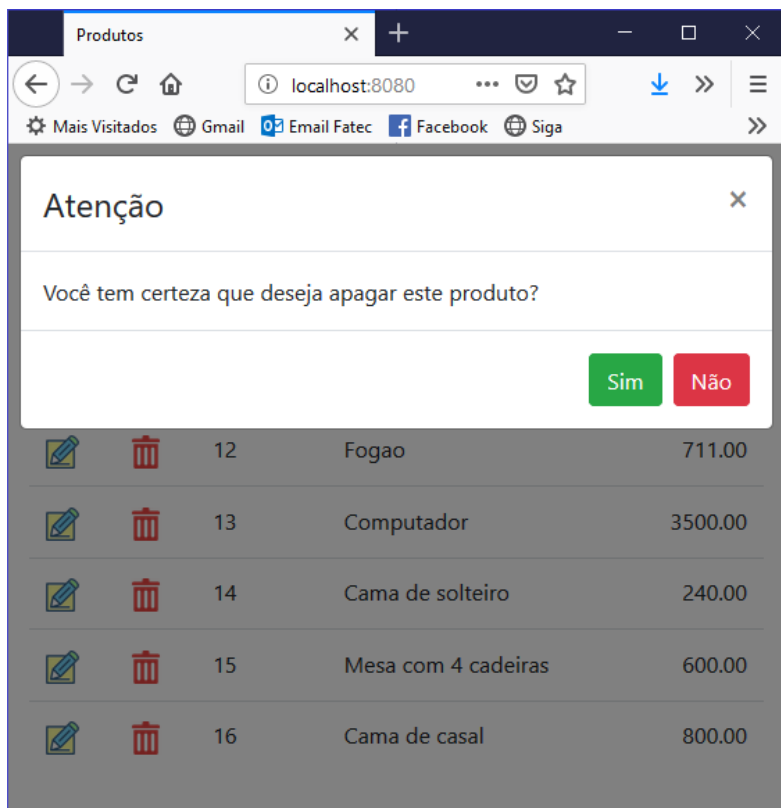


Figura 18: Caixa de diálogo para confirmar uma exclusão

Ao clicar no botão “Sim” executamos a função, mostrada a seguir, que realiza a requisição AJAX para que o documento que contém o referido produto seja apagado do

Cláudio Luís V. Oliveira

MongoDB e a tabela, que mostra os produtos, seja atualizada na página principal.

```
.....  
$("#sim-deletar").click(function(evento) {  
    evento.preventDefault();  
    var dados = {  
        "codigo": parseInt(codigo)  
    };  
  
    $.ajax({  
        type: "POST",  
        contentType: "application/json",  
        url: window.location +  
"produtos/deletar",  
        data: JSON.stringify(dados),  
        dataType: 'json',  
        success: function(produto) {  
            $('#caixa-confirmacao').modal('hide');  
            window.location.reload();  
        },  
        error: function(erro) {  
            alert("ERRO: " + erro);  
        }  
    });  
});  
.....
```

Exercícios

- 1) Descrever, utilizando JSON, uma estrutura que possibilite representar o CNPJ de uma determinada

empresa, sua razão social, nome fantasia, endereço e patrimônio líquido.

- 2) Considerando a estrutura desenvolvida no exercício anterior utilize o console do MongoDB para realizar a inclusão de um documento na coleção “empresas”.
- 3) Considerando a estrutura desenvolvida no Exercício 1, crie uma aplicação Node.js para possibilitar a inclusão de um documento no MongoDB na coleção “empresas”. Os dados deverão ser digitados através do console do Node.js.
- 4) Considerando a estrutura desenvolvida no Exercício 1, crie um servidor Node.js que mostre uma página, contendo um formulário HTML, que através de uma requisição AJAX realize a inclusão de um documento no MongoDB na coleção “empresas”.
- 5) A partir da estrutura desenvolvida no Exercício 1, crie um servidor Node.js que mostre uma página HTML que mostre todos os documentos existentes na coleção “empresas” armazenada no MongoDB.

Referências

Oliveira, C. L. V.; Zanetti, H. A. P.. **JavaScript Descomplicado: Programação para a Web, IoT e Dispositivos Móveis**. São Paulo: Érica, 2020. 216p.

Mozilla Developer Network. Disponível em

<https://developer.mozilla.org/pt-BR/>

Uma reintrodução ao JavaScript (Tutorial de JS).

Disponível em [https://developer.mozilla.org/pt-](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/A_re-introduction_to_JavaScript)

[BR/docs/Web/JavaScript/A re-introduction to JavaScript](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/A_re-introduction_to_JavaScript)

Node.js - About Docs. Disponível em

<https://nodejs.org/en/docs/>

Introdução Express/Node. Disponível em

[https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express Nodejs/Introduction](https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs/Introduction)