

CLÁUDIO LUÍS VIEIRA OLIVEIRA  
HUMBERTO AUGUSTO PIOVESANA ZANETTI



érica

# Dedicatória

À minha esposa Claudia, por tantos e tantos anos de companheirismo, comunhão e amor.

À minha filha Franciele, com muito amor.

Aos meus pais Maria Creyde e Manoel, pela dedicação e amor à família.

Ao grande amigo Humberto pelos inúmeros projetos compartilhados.

À minha esposa Flavia, por seu incentivo e dedicação.

Ao meus pais, Alberto e Célia. Aos meus irmãos, Luis Gustavo e Pedro, por sempre acreditarem em mim.

Ao meu grande amigo Cláudio, pela parceria nos projetos em todos esses anos.

À minha princesa Betina, por tornar meus dias cada vez mais felizes.

## **Requisitos mínimos de hardware e software**

- Computador Dual Core, memória RAM de 2 GB e espaço livre de 100 MB no disco rígido.
- Windows 7 (ou superior), Linux (qualquer distribuição) ou MacOS X 10 (ou superior).
- Navegador de internet: Firefox, Chrome, Internet Explorer ou Edge (qualquer versão).
- NetBeans (versão 8.1 ou superior).
- XAMPP (versão 7 ou superior).
- Node.js (versão 10 ou superior).
- MongoDB (versão 4 ou superior).
- MariaDB (versão 10 ou superior) ou MySQL Community Server (versão 8 ou superior).

*Cláudio Luís Vieira Oliveira*

*Humberto Augusto Piovesana Zanetti*

## **Sobre os autores**

**Claudio Luis Vieira Oliveira** é mestre em Sistemas de Computação pela Pontifícia Universidade Católica de Campinas (PUC-Campinas) e bacharel em Análise de Sistemas pela Universidade Metodista de Piracicaba (Unimep). Tem ampla experiência na área de Ciéncia da Computação. É professor das Faculdades de Tecnologia (Fatec) de Jundiaí, Bragança Paulista e Campinas.

**Humberto Augusto Piovesana Zanetti** é doutorando em Tecnologia pela Faculdade de Tecnologia da Universidade Estadual de Campinas (FT-Unicamp) e mestre em Ciéncia da Computação do Centro Universitário Campo Limpo Paulista (UNIFACCAMP). Há 15 anos trabalha no Ensino Técnico e Superior. Atualmente, é professor na Escola Técnica Rosa Perrone Scavone (Itanhaém/SP) e na Faculdade de Tecnologia (Fatec) de Jundiaí.

## **Apresentação**

A internet é cada vez mais utilizada no mundo todo, e está presente nos mais diversos aspectos do cotidiano das pessoas. Redes sociais, comércio eletrônico, cotação de produtos e serviços, serviços bancários, agendamento de serviços públicos e consultas dos mais diversos tipos são alguns poucos exemplos diários de utilização da internet. Dessa forma, há uma gigantesca demanda para o desenvolvimento desse tipo de aplicação, e a linguagem JavaScript tem sido amplamente utilizada, tanto para o desenvolvimento do front-end, em que está fortemente integrada ao HTML e ao CSS, como para back-end, por meio do Node.js.

Inúmeras tecnologias e ferramentas estão disponíveis para os desenvolvedores de sites e aplicações para a internet. Mas há um consenso de que o JavaScript é uma linguagem que deve ser aprendida e dominada, pois é simples, fácil, além de apresentar uma infinidade de bibliotecas e frameworks que aumentam significativamente a produtividade e a qualidade final das soluções para a web.

Neste livro, você estudará os conceitos fundamentais que possibilitam aplicações para a web, além de ferramentas e frameworks mais utilizados, incluindo o uso de sistemas de bancos de dados para realizar o armazenamento das informações.

Segundo as mudanças tecnológicas atuais, dedicamos um capítulo para o desenvolvimento de soluções para a Internet das Coisas (IoT), apresentando ferramentas e recursos disponíveis e sua integração com o JavaScript. Há também outro capítulo que aborda do uso do JavaScript no desenvolvimento de aplicativos para dispositivos móveis, isto é, celulares e tablets.

Explore este livro e as inúmeras possibilidades e tecnologias que descrevemos aqui, e deixe sua criatividade fluir!

# Sumário

Requisitos mínimos de hardware e software

Dedicatória

Sobre os autores

Apresentação

1. HTML

  1.1 Cabeçalhos

  1.2 Parágrafos

  1.3 Fontes

  1.4 Imagens

  1.5 Links

  1.6 Listas

  1.7 Tabelas

2. Cascading Style Sheets

  2.1 CSS Inline

  2.2 CSS Interno

  2.3 CSS Externo

  2.4 Seletores e Classes

3. Programação para a Web

4. JavaScript

  4.1 Primeiro Script

4.2 Variáveis em JavaScript

4.3 Operadores

4.4 Depuração de Scripts

4.5 Caixas de Mensagem e Diálogo

4.6 Containers

4.7 Estruturas de Controle

4.8 Vetores

4.9 Exibição de Imagens

4.10 Interação com Formulários

4.11 Melhorias na Página da Calculadora Básica

4.12 Botões de Rádio

4.13 Caixas de Seleção

4.14 Validação dos Campos do Formulário

4.15 JavaScript Externo

4.16 Interação entre Páginas

4.17 Requisição de Dados

## 5. jQuery

5.1 Olá jQuery

5.2 AJAX (Asynchronous Javascript and XML)

## 6. Bootstrap

6.1 Olá Bootstrap

6.2 Formulários com Bootstrap

## 7. Node.js

7.1 Olá Node.js

7.2 Entrada de Dados por meio do Console

7.3 Implementando um Servidor

7.4 Construção de uma Aplicação

## 8. Interação com Banco de Dados

8.1 MongoDB

8.2 JavaScript Object Notation (JSON)

8.3 Inserção

- 8.4 Consulta
- 8.5 Alteração
- 8.6 Exclusão
- 8.7 Aplicação Node.js com Acesso ao MongoDB
- 8.8 Manipulação de Documentos por meio do Node.js
- 8.9 Aplicação Web com Node.js e MongoDB

## 9. Internet das Coisas (IoT)

- 9.1 Arduino
- 9.2 Carga do Firmata no Arduino
- 9.3 Interação com o Arduino
- 9.4 Controle de um Dispositivo por meio da Internet
- 9.5 Leitura de Sensores
- 9.6 Exibição dos Dados de Sensores em uma Página Web
- 9.7 Armazenamento dos Dados de Sensores
- 9.8 Consulta ao Banco de Dados
- 9.9 Chart.js
- 9.10 ThingSpeak
- 9.11 Envio de Dados para a Nuvem
- 9.12 Visualização dos Dados

## 10. Desenvolvimento para Dispositivos Móveis

- 10.1 Framework Ionic
- 10.2 Nosso Primeiro Aplicativo
- 10.3 Desenvolvendo uma Calculadora Básica
- 10.4 Aplicativo para Cálculo de Índice de Massa Corporal (IMC)
- 10.5 Aplicativo de Contatos Telefônicos

## Referências Bibliográficas

# HTML

## INICIANDO OS ESTUDOS

---

Neste capítulo, serão apresentados os conceitos da linguagem HTML, que é o elemento fundamental de qualquer página desenvolvida para a web. Também serão abordados os principais elementos utilizados para realizar a definição do aspecto visual e do conteúdo de uma página HTML.

A linguagem HTML é a base de criação de qualquer página para a web, porém se trata de uma linguagem composta por elementos estáticos e está fortemente focada na composição e na formatação de documentos.

HTML é a sigla para *HyperText Markup Language* ou Linguagem de Formatação de Hipertexto. De modo geral, são documentos de texto escritos em códigos que podem ser interpretados pelos navegadores que, por sua vez, exibem as páginas devidamente formatadas.

Todo documento HTML apresenta os comandos delimitados pelos sinais de maior (>) e menor (<), sendo esses elementos denominados *tags* ou etiquetas. Eles correspondem aos comandos de formatação da linguagem. A maioria das etiquetas tem sua correspondente de fechamento, que é identificado pelo uso do caractere barra (/). Veja o exemplo a seguir:



```
<h1>Olá</h1>
```

Uma *tag* também pode ser formada por um conjunto de atributos e valores. Os atributos modificam os resultados padrões dos comandos, e os valores caracterizam essa mudança. Por exemplo, temos o atributo color que possibilita mudar a cor da letra, conforme ilustra o trecho de código-fonte a seguir:



```
<font color='blue'>Olá</font>
```

De maneira geral, HTML é uma linguagem muito simples e acessível para a produção e o compartilhamento de documentos.



### ATENÇÃO!

Recomendamos o uso do NetBeans para o desenvolvimento dos exemplos apresentados neste livro. O NetBeans pode ser gratuitamente obtido no link <<https://netbeans.org/>>.

O exemplo a seguir mostra uma página que contém apenas elementos do HTML, também conhecidos como *tags* ou etiquetas.



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

```
<head>
<title>Primeira página</title>
<meta charset="UTF-8">
</head>
<body>
<h1>Olá pessoal</h1>
</body>
</html>
```

ola.html

Note que as etiquetas devem ser usadas de modo hierárquico, isto é, a etiqueta html foi a primeira a ser aberta, então, deverá ser a última a ser fechada. A etiqueta head, por sua vez, deve estar dentro da etiqueta html e deverá ser fechada antes da etiqueta html e, assim, sucessivamente.

Ao executarmos essa página, o navegador conectado à internet é ativado e solicita ao servidor web o seu envio. O resultado desse processo deve ser similar ao mostrado pela Figura 1.1.



Figura 1.1 – Exemplo de página em HTML.

Como mencionado anteriormente, a base para a construção de aplicações para a web é a linguagem HTML. Ela, porém, basicamente realiza a formatação de documentos, mas não apresenta os conceitos das linguagens de

programação, como variáveis e estruturas de controle, entre outros. Dessa forma, para se criarem aplicações mais complexas e dinâmicas, é necessário o uso de outra linguagem de programação que complemente o HTML.

A seguir detalharemos a aplicação das etiquetas mais comumente utilizadas em páginas HTML. Lembramos que muitas outras estão disponíveis e podem ser facilmente encontradas em buscas na internet.



DICA

Uma sugestão é a Mozilla Developer Network. Acesse o link <<https://developer.mozilla.org/pt-BR/docs/Web/HTML>>.

## 1.1 Cabeçalhos

As etiquetas de cabeçalho são usadas para destacar determinados títulos e subtítulos. São identificados pelas etiquetas <h1> até <h6>.



```
<html>
<head>
<title>Cabeçalhos</title>
<meta charset="UTF-8">
</head>
<body>
<h1>Cabeçalho 1</h1>
<h2>Cabeçalho 2</h2>
<h3>Cabeçalho 3</h3>
<h4>Cabeçalho 4</h4>
<h5>Cabeçalho 5</h5>
<h6>Cabeçalho 6</h6>
Texto com a formatação padrão (sem destaque).
</body>
</html>
```

[cabecalho.html](#)

Na Figura 1.2, é possível observar a aparência da página após ser carregada pelo navegador.



Figura 1.2 – Cabeçalhos.

## 1.2 Parágrafos

A etiqueta de parágrafo permite agrupar e realizar o alinhamento do texto.



```
<!DOCTYPE html>
<html>
<head>
<title>Parágrafos</title>
<meta charset="UTF-8">
```

```
</head>
<body>
<h1>Parágrafos</h1>
<p>
A ligeira raposa marrom salta sobre o cão preguiçoso. A ligeira raposa marrom
salta sobre o cão preguiçoso. A ligeira raposa marrom salta sobre o cão
preguiçoso.
</p>
<p align="right">
A ligeira raposa marrom salta sobre o cão preguiçoso. A ligeira raposa marrom
salta sobre o cão preguiçoso. A ligeira raposa marrom salta sobre o cão
preguiçoso.
</p>
<p align="center">
A ligeira raposa marrom salta sobre o cão preguiçoso. A ligeira raposa marrom
salta sobre o cão preguiçoso. A ligeira raposa marrom salta sobre o cão
preguiçoso.
</p>
<p align="justify">
A ligeira raposa marrom salta sobre o cão preguiçoso. A ligeira raposa marrom
salta sobre o cão preguiçoso. A ligeira raposa marrom salta sobre o cão
preguiçoso.
</p>
</body>
</html>
```

---

paragrafo.html

Na Figura 1.3, mostramos os parágrafos alinhados à esquerda (padrão), direita, centro e justificado.

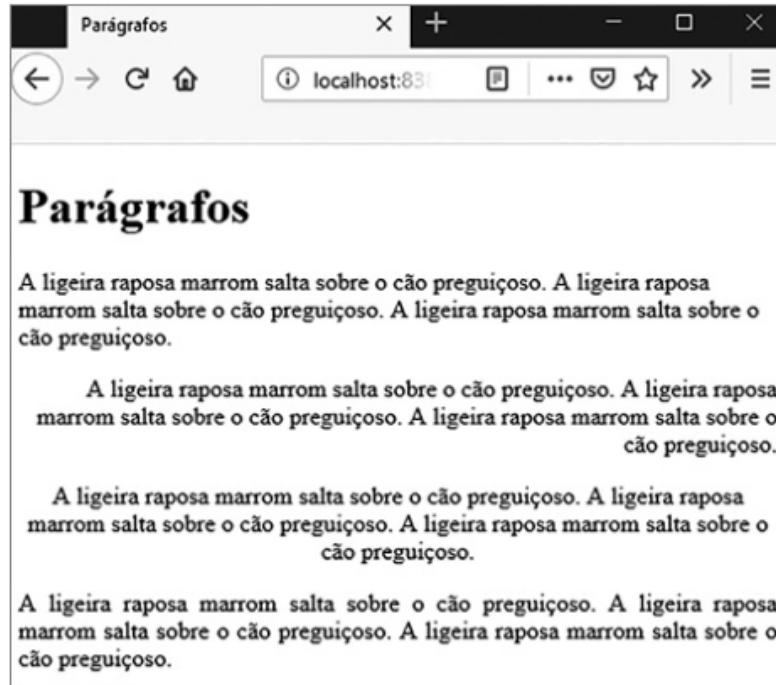


Figura 1.3 – Parágrafos.

### 1.3 Fontes

É possível mudar a fonte de barra, tamanho e cor de um texto por meio da utilização da etiqueta font. Também existem as etiquetas para itálico (i), negrito (b ou strong) e sublinhado (u).



```
<!DOCTYPE html>
<html>
<head>
<title>Fontes</title>
<meta charset="UTF-8">
</head>
<body>
<h1>Fontes</h1>
```

HTML possui um conjunto de etiquetas (tags) que possibilitam a formatação do texto de uma página. As etiquetas mais usadas para realizar a formatação são:<br />

```
<b>b - Negrito</b><br />
<strong>strong - Reforçado</strong><br />
<i>i - Itálico</i><br />
<u>u - Sublinhado</u><br /><br />
```

A etiqueta font permite a alteração do tipo da fonte de letra, tamanho e cor do texto, por exemplo:<br />

```
<font size="+2" color="red">
  Texto com tamanho ampliado e cor vermelha</font>
<br /><br />
<font size="-1" color="#FF8000">
  Texto com tamanho reduzido e cor laranja</font>
</body>
</html>
```

---

fonte.html

A Figura 1.4 mostra as diferentes opções disponíveis para formatação dos textos de uma página HTML.



Figura 1.4 – Fontes.

## 1.4 Imagens

A etiqueta img permite trabalhar com imagens dentro de uma página web. Isso pode ser feito por meio dos atributos inserir (src), dimensionar (width e height) e alinhar (align).



```
<!DOCTYPE html>
<html>
<head>
<title>Imagens</title>
<meta charset="UTF-8">
</head>
<body>
<h1>Imagens</h1>
A etiqueta img permite exibir imagens em uma página HTML.

```

```
</body>
```

```
</html>
```

---

imagem.html

Na Figura 1.5, podemos observar a exibição da imagem devidamente alinhada à esquerda.



Figura 1.5 – Imagem.

## 1.5 Links

Os links, também conhecidos como hyperlinks, permitem realizar a ligação entre as páginas HTML e os diferentes tipos de conteúdo (por exemplo, imagens, documentos ou programas para download). Também é possível realizar a ligação entre conteúdos que estão em uma mesma página HTML. A etiqueta a (âncora) deverá ser usada para definir o link.



---

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Sistema Solar</title>
```

```
<meta charset="UTF-8">
</head>
<body>
<h1>Sistema Solar</h1>
<p align="justify">O Sistema Solar consiste no conjunto constituído pelo Sol e todos os corpos celestes que estão sob sua ação gravitacional.</p>
<a href="#sol">Sol</a><br>
<a href="mercurio.html">Mercúrio</a><br>
Vênus<br>
Terra<br>
Marte<br>
Júpiter<br>
Saturno<br>
Urano<br>
Netuno<br>
<h1><a id="sol">Sol</a></h1>
<p align="justify">O Sol é a estrela central do Sistema Solar. Seu volume é composto por 92% hidrogênio, 7% de hélio e 1% de outros elementos.</p>
<br>
<a target="_blank"
href="https://pt.wikipedia.org/wiki/Sol">
Mais informações</a>
</body>
</html>
```

---

sistema-solar.html

Observe que o primeiro link criado na página, destacado a seguir, faz referência a uma âncora criada na mesma página. Nesse caso, devemos começar com o caractere # e colocar o nome.



```
<a href="#sol">Sol</a><br>
```

O local da página que será mostrado deve ser definido usando o atributo id da âncora, conforme destacado a seguir.

</>

```
<a id="sol">Sol</a>
```

Quando desejamos referenciar outra página, devemos colocar o endereço completo da localização da página. Outra possibilidade, destacada a seguir, é indicar somente o nome da página. Nesse caso, ela deve estar hospedada exatamente na mesma pasta em que a página que a chamou está localizada.

</>

```
<a href="mercurio.html">Mercúrio</a><br>
```

Na Figura 1.6 temos a página criada, em que é possível observar os links que foram criados para Sol e Mercúrio.



Figura 1.6 – Links.

Ao clicar no link "Sol", a página terá seu conteúdo exibido a partir da âncora sol, conforme apresenta a Figura 1.7.

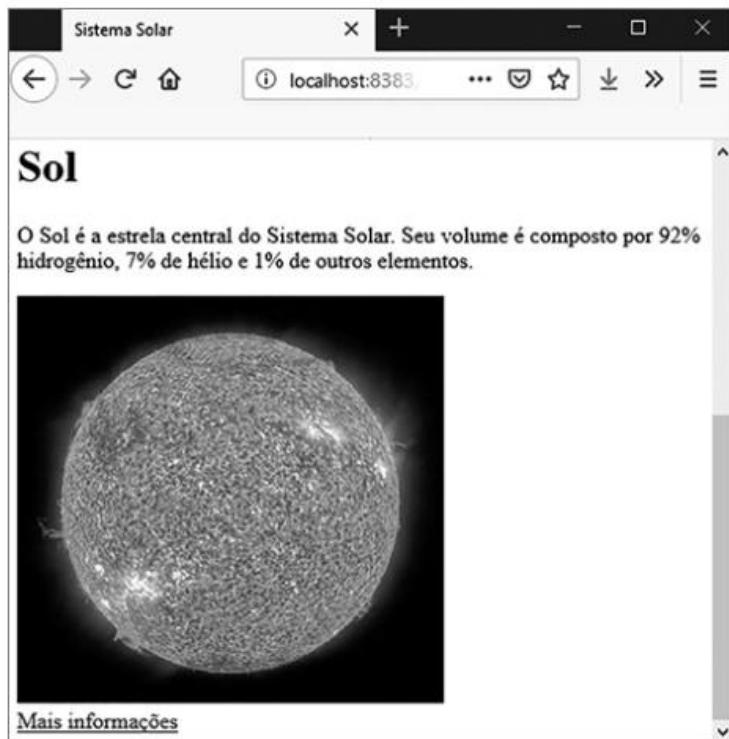


Figura 1.7 – Página mostrada a partir da âncora.

No trecho da página destacado a seguir, vamos criar um link para um endereço (URL) localizado em outro servidor. Note, nesse caso, que o endereço completo foi utilizado. Também observe o uso do atributo `target`, com o valor “`_blank`”, que determina que a página, indicada no link, deve ser aberta em uma nova aba do navegador.



```
<a target="_blank"  
    href="https://pt.wikipedia.org/wiki/Sol">  
    Mais informações</a>
```

A página sobre o planeta Mercúrio apresenta o conteúdo mostrado a seguir.



```
-----  
<!DOCTYPE html>  
  
<html>  
  <head>  
    <title>Mercúrio</title>  
    <meta charset="UTF-8">  
  </head>  
  <body>  
    <h1><a id="sol">Mercúrio</a></h1>  
    <p align="justify">Mercúrio é o menor planeta do Sistema Solar, também é o mais  
    próximo do Sol. Visualmente é parecido com a Lua, apresentando crateras de  
    impacto e planícies lisas, não possui satélites naturais</p>  
    <a target="_blank"  
       href="https://pt.wikipedia.org/wiki/Merc%C3%BArio_(planeta)">Mais  
    informações</a>  
  </body>  
</html>
```

mercurio.html

## 1.6 Listas

Uma página HTML pode ter, em seu conteúdo, listas ordenadas (numeradas) e listas não ordenadas. O primeiro exemplo implementa uma lista não ordenada. Observe o uso das etiquetas ul para definir a lista e li para identificar cada item que compõe a lista.



```
-----  
<!DOCTYPE html>  
  
<html>  
  <head>  
    <title>Lista de Compras</title>  
    <meta charset="UTF-8">  
  </head>
```

```
<body>
<h1>Lista de Compras</h1>
<ul>
<li>Banana</li>
<li>Café</li>
<li>Batata</li>
<li>Frango</li>
<li>Pão de Forma</li>
<li>Manteiga</li>
</ul>
</body>
</html>
```

[lista-nao-ordenada.html](#)

Na Figura 1.8 temos a exibição da página criada com a respectiva lista não ordenada, que é caracterizada pelo uso de marcadores iniciando cada item da lista.



Figura 1.8 – Lista não ordenada.

No exemplo a seguir, criaremos uma lista ordenada (numerada). Note que, nesse caso, devemos usar a etiqueta ol para delimitar a lista e li para identificar cada item.

</>

```
-----  
<!DOCTYPE html>  
<html>  
<head>  
<title>Como trocar uma lâmpada?</title>  
<meta charset="UTF-8">  
</head>  
<body>  
<h1>Como trocar uma lâmpada?</h1>  
<ol>  
<li>Desligar a energia elétrica</li>  
<li>Pegar uma escada</li>  
<li>Pegar uma lâmpada nova</li>  
<li>Colocar a escada sob a lâmpada queimada</li>  
<li>Subir na escada</li>  
<li>Desrosquear a lâmpada queimada</li>  
<li>Rosquear a nova lâmpada</li>  
<li>Descer da escada</li>  
<li>Ligar a energia elétrica</li>  
</ol>  
</body>  
</html>
```

lista-ordenada.html

A lista ordenada será exibida na página, conforme ilustra a Figura 1.9.

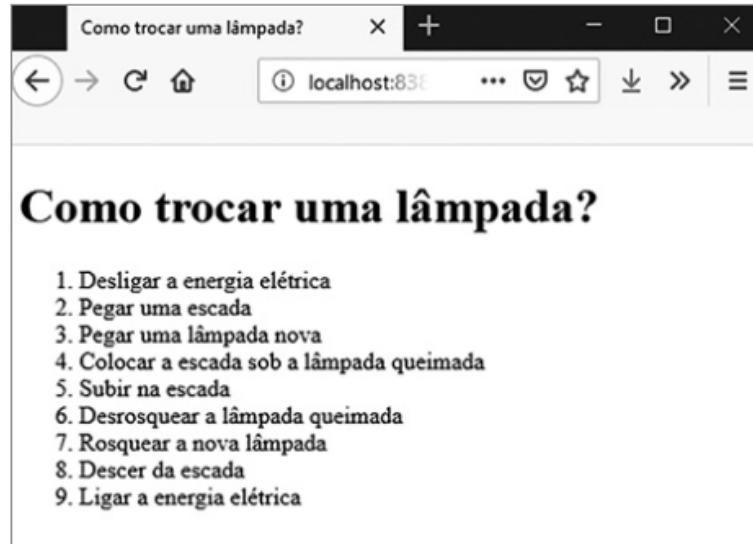


Figura 1.9 – Lista ordenada.

## 1.7 Tabelas

É muito simples criar tabelas usando HTML. Um conjunto de etiquetas permite especificar a tabela com suas respectivas linhas e colunas.



```
<!DOCTYPE html>
<html>
<head>
<title>Relatório de Vendas</title>
<meta charset="UTF-8">
</head>
<body>
<h1>Relatório de Vendas</h1>
<table border="1" cellspacing="0" cellpadding="2">
<tr>
<td align="center">Produto</td>
```

```
<td align="center">Quantidade</td>
<td align="center">Preço (R$)</td>
</tr>
<tr>
<td>Televisor</td>
<td align="center">2</td>
<td align="right">3.000,00</td>
</tr>
<tr>
<td>Geladeira</td>
<td align="center">4</td>
<td align="right">1.990,00</td>
</tr>
<tr>
<td>Fogão</td>
<td align="center">3</td>
<td align="right">600,00</td>
</tr>
</table>
</body>
</html>
```

---

tabela.html

Analisando a página, inicialmente devemos identificar a etiqueta `table` e os atributos que permitem definir a borda e o espaçamento entre as células. Em seguida, note o uso da etiqueta `tr` para especificar uma linha da tabela e a etiqueta `td` para determinar uma coluna. É importante salientar que se trata de uma estrutura hierárquica, ou seja, as colunas devem estar dentro de uma linha que, por sua vez, devem estar dentro da tabela. A Figura 1.10 mostra a exibição da página.

The screenshot shows a web browser window with the title 'Relatório de Vendas'. The address bar displays 'localhost'. The main content area contains the title 'Relatório de Vendas' and a table with three rows of data:

| Produto   | Quantidade | Preço (R\$) |
|-----------|------------|-------------|
| Televisor | 2          | 3.000,00    |
| Geladeira | 4          | 1.990,00    |
| Fogão     | 3          | 600,00      |

Figura 1.10 – Tabela.

### VAMOS PRATICAR!

1. Crie uma página HTML com conteúdo definido a seu critério, mostrando um título formatado com a etiqueta <H2>, um parágrafo com alinhamento justificado e fonte tamanho 7.
2. Desenvolva uma página HTML com conteúdo definido a seu critério, que apresente o título formatado com a etiqueta <h1>, dois subtítulos formatados com a etiqueta <h2>. O conteúdo do primeiro subtítulo deve ser apresentado na cor azul, fonte 4 px e alinhado à direita. O conteúdo do segundo subtítulo deve estar alinhado à esquerda.
3. Elabore uma página HTML que mostre em uma tabela de duas colunas e três linhas imagens de aparelhos domésticos comuns (por exemplo, fogão, geladeira e televisor). Cada imagem deve ser colocada em uma célula, juntamente de sua respectiva descrição formatada na cor verde, 6 px e centralizada.
4. Crie uma página HTML que mostre o título "Notícias" formatado com a etiqueta <H1> e apresente um conjunto de links para sites de notícias disponíveis na internet.
5. Elabore um site com três páginas HTML, conteúdo definido a seu critério, sendo que a primeira página deve possuir links para as outras duas páginas criadas.

# Cascading Style Sheets

## INICIANDO OS ESTUDOS

---

Neste capítulo, mostraremos como utilizar as Folhas de Estilo em Cascata (CSS ou *Cascading Style Sheets*) para expandir as possibilidades de formatação do conteúdo de páginas HTML. Além disso, o seu uso é fundamental para a padronização e a definição de uma identidade visual das diversas páginas de um website.

As Folhas de Estilo em Cascata (CSS ou *Cascading Style Sheets*) permitem customizar a formatação dos elementos HTML. Dessa forma, ampliam as possibilidades de exibição das páginas e oferecem bastante versatilidade para a diagramação das páginas e a construção da identidade visual de um site. Outra vantagem de adotarmos CSS em nossos sites é a possibilidade de separar o conteúdo das páginas de sua formatação, o que facilita o processo de manutenção e atualização das páginas.

Vamos abordar as três formas de utilização do CSS em nossas páginas HTML.

## 2.1 CSS Inline

A primeira opção mostrada consiste em utilizar o atributo `style` das etiquetas para inserir diretamente o elemento CSS. Essa opção é interessante apenas quando temos pouco conteúdo no site ou para fazer algum teste rápido para avaliar alterações na diagramação da página. Mas, de forma geral, deve ser evitada nos sites que já estão publicados.



```
<!DOCTYPE html>
<html>
<head>
<title>Introdução ao CSS</title>
<meta charset="UTF-8">
</head>
<body style="background-color:navy;">
<h1 style="font-family:sans-serif; color:yellow; padding:20px;">
    Introdução ao CSS
</h1>
<p style="color:white; padding:20px;">
    CSS Inline.
</p>
</body>
</html>
```

[css-inline.html](#)

Conforme mostra o exemplo anterior, o CSS trabalha sempre por meio de pares de propriedades e valores, sendo que os pares devem ser separados por ponto e vírgula. Propriedades e valores estão unidos pelo caractere dois-pontos (por exemplo: `color:white`, em que `color` é a propriedade e `white` é o valor que será atribuído à propriedade). Na Figura 2.1, temos a página HTML customizada com as propriedades do CSS.



Figura 2.1 – Exemplo de CSS Inline.

## 2.2 CSS Interno

Essa opção permite separar o conteúdo da página dos elementos de formatação por meio do emprego da etiqueta `style`, em que será definido o conjunto de propriedades customizadas pela folha de estilos em cascata.



```
<!DOCTYPE html>
<html>
<head>
<title>Introdução ao CSS</title>
<meta charset="UTF-8">
<style>
body {
```

```
background-color: black;  
}  
  
h1 {  
color: cyan;  
text-align: center;  
padding: 30px;  
}  
  
p {  
color: white;  
padding: 30px;  
}  
  
</style>  
</head>  
<body>  
<h1>Introdução ao CSS</h1>  
<p>CSS Interno</p>  
</body>  
</html>
```

---

css-interno.html

Observe que, dentro da etiqueta `style`, são definidas as propriedades aplicáveis ao elemento HTML, conforme destaca o trecho de código-fonte a seguir. No exemplo, a etiqueta `h1` terá a sua cor alterada para ciano, o texto será exibido alinhado ao centro da página e terá uma margem de 30 pixels em relação aos demais elementos HTML. Também é importante salientar que o grupo de propriedades para determinado elemento HTML deve estar delimitado por chaves (Figura 2.2).

CSS

---

```
h1 {  
color: cyan;  
text-align: center;  
padding: 30px;
```

}

---

Na Figura 2.2 temos a exibição da página criada.



Figura 2.2 – Exemplo de CSS Interno.

## 2.3 CSS Externo

Nesse item, mostraremos como separar o conteúdo das páginas de formatação, além de como possibilitar que uma única folha de estilos seja utilizada por diversas páginas do site. Dessa maneira, é possível criar uma identidade visual para todo o site que é simples de manter, pois qualquer alteração dentro da folha de estilos afeta automaticamente todas as páginas do site. Observe que será criado um arquivo (estilo.css) que contém todos os elementos CSS. A página HTML apenas deve referenciar esse arquivo por meio da etiqueta `link`.



---

```
<!DOCTYPE html>

<html>
<head>
<title>Introdução ao CSS</title>
<meta charset="UTF-8">
<link rel="stylesheet" type="text/css" href="estilo.css" />
</head>
<body>
```

```
<h1>Introdução ao CSS</h1>
<p>CSS Externo</p>
</body>
</html>
```

---

css-externo.html

Como mencionado, o arquivo CSS contém apenas os elementos de formatação da página.

CSS

---

```
body {
background-color: navy;
}
h1 {
color: yellow;
text-align: center;
}
p {
color: white;
}
```

---

estilo.css

Na Figura 2.3, você vê a página devidamente customizada pela folha de estilos.



Figura 2.3 – Exemplo de CSS Externo.

## 2.4 Seletores e Classes

No exemplo anterior notamos que a formatação sobre determinado elemento HTML afeta todos os elementos da página. Por exemplo, a formatação da etiqueta de parágrafo (p), mostrada em detalhes no trecho de código-fonte a seguir, afeta todos os parágrafos da página.

CSS

```
p {  
color: white;  
}
```

No projeto de um website, existem situações em que precisamos tratar determinados elementos de forma individual ou por meio de grupos. Nesses casos, quando for preciso atuar sobre um elemento individualmente, é indicado aplicar os conceitos de seletores, ou, no caso das classes, utilizamos quando desejamos trabalhar com um grupo de elementos.

</>

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Introdução ao CSS</title>
```

```
<meta charset="UTF-8">
<link rel="stylesheet" type="text/css" href="estilo.css" />
</head>
<body id="fundo">
<h1 id="titulo">Introdução ao CSS</h1>
<p class="texto">Seletores</p>
<p class="texto">Classes</p>
</body>
</html>
```

---

seletores-classes.html

Observe no código-fonte apresentado a seguir que um seletor será especificado usando o caractere `#`, sempre precedendo o identificador (`id`) do elemento afetado.

Nesse caso, o identificador deve ser único, ou seja, não podemos definir mais do que um elemento HTML com o mesmo `id` em uma mesma página. Por exemplo, o elemento `body` que possui a propriedade `id` definida como `fundo` será referenciada no CSS como `#fundo`.

Uma classe, por sua vez, permite referenciar um conjunto de elementos por meio do atributo HTML `class`, que será referenciado no CSS com o caractere de ponto final antecedendo o nome da classe (por exemplo, `.texto`), que será usado pelos parágrafos que pertencem à classe `texto`.

**CSS**

---

```
#fundo {
background-color: #808080;
padding: 20px;
}
#titulo {
color: lime;
}
.texto {
color: lightcyan;
font-weight: bold;
```

{

estilo.css

Na Figura 2.4, apresentamos a página.



Figura 2.4 – Seletores e classes.



Uma ótima referência é a [Mozilla Developer Network](https://developer.mozilla.org/pt-BR/docs/Web/CSS). Para visualizar, acesse o link: <https://developer.mozilla.org/pt-BR/docs/Web/CSS>.

### VAMOS PRATICAR!

1. Crie uma página HTML com conteúdo definido a seu critério. Deve mostrar um título formatado com a etiqueta <H2> e, por meio de CSS inline, um parágrafo com alinhamento justificado e fonte 14 px.
2. Desenvolva uma página HTML, com conteúdo definido a seu critério, que apresente título formatado com a etiqueta <h1>, dois subtitulos formatados com a etiqueta <h2>. O texto relacionado ao primeiro subtítulo deve ser apresentado na cor azul, fonte tamanho 12 px e alinhado à direita. O texto relacionado ao segundo subtítulo deve estar alinhado à esquerda. Utilize CSS interno para realizar as formatações solicitadas.
3. Elabore uma página HTML que mostre uma tabela com seis imagens de aparelhos domésticos comuns (por exemplo, fogão, geladeira e televisor). Cada imagem deve ser apresentada centralizada em uma célula, juntamente de sua respectiva descrição formatada na cor verde, tamanho 24 px e centralizada. Adote CSS interno para realizar as formatações solicitadas.
4. Elabore um site com três páginas HTML, com conteúdo definido a seu critério, em que a primeira página

deve possuir links para as outras duas páginas criadas. Utilize um CSS externo de modo que todas as páginas apresentem um fundo preto; os títulos devem ser mostrados em amarelo, textos na cor branca, fonte tamanho 14 px e links na cor ciano.

5. Aplicando o conceito de CSS externo, seletores e classes, crie uma página que exiba no nome das cores vermelho, verde, amarelo, azul, ciano, rosa, roxo e marrom. Cada item deve ser mostrado formatado na cor que representa e em tamanho 28 px.

# Programação para a Web

## INICIANDO OS ESTUDOS

---

Neste capítulo, apresentaremos, de forma suscinta, a linguagem HTML, considerada base para o desenvolvimento de páginas web, bem como sua classificação.

A linguagem HTML é a base de criação de qualquer página para a web. Mas se trata de uma linguagem composta por elementos estáticos, sendo fortemente focada na composição e na formatação de documentos.

Com a expansão do uso da internet nas últimas décadas, houve a necessidade de criar linguagens que conseguissem agregar conteúdo dinâmico às páginas, além de recursos, como a validação de dados e o acesso a banco de dados, entre outras funcionalidades.

Nesse contexto, surgem as linguagens como PHP, ASP.net, JSP, JavaScript, entre outras. Essas linguagens não substituem o HTML, que é a base de qualquer documento para a web, mas atuam em conjunto, suprindo os recursos que não estão disponíveis no HTML.

Essas linguagens são classificadas em:

- **Server Side**, ou seja, são executadas no servidor web.
- **Client Side**, que são as linguagens executadas no cliente (navegador web).

Nas linguagens Server Side, os programas criados são executados no servidor e os resultados são enviados em HTML para o cliente que está acessando o servidor por meio de um programa navegador (Figura 3.1). Como principais linguagens Server Side, podemos citar PHP, ASP, ASP.Net e JSP (Java).

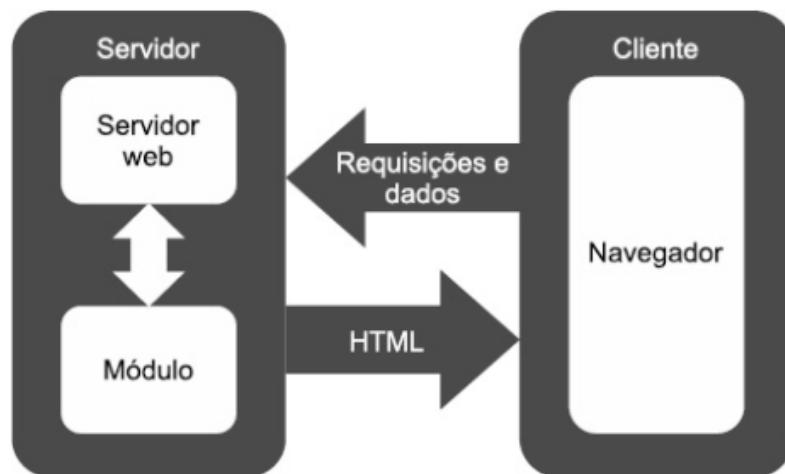


Figura 3.1 – Estrutura básica de uma aplicação web.

Já nas linguagens Client Side, os scripts são enviados juntamente do HTML para o navegador web, o qual será responsável pela execução. A principal linguagem Client Side é o JavaScript, que hoje é padrão no mercado.

# JavaScript

## INICIANDO OS ESTUDOS

---

Neste capítulo, apresentaremos a linguagem de programação JavaScript. Ela é utilizada em páginas web para acrescentar características dinâmicas. Com isso, há, de maneira simples e eficiente, mais capacidade de interação com o usuário. O JavaScript acrescenta inúmeras possibilidades quando aplicado em conjunto com HTML e CSS na construção de sites para a internet.

O JavaScript é uma linguagem de programação orientada a objetos, sendo interpretada e executada pelo navegador web (server-side script). Apresenta uma sintaxe similar à linguagem Java e tem como objetivo principal oferecer melhor interatividade às páginas.

Uma característica importante da linguagem JavaScript é que ela não apresenta tipos de dados, isto é, qualquer variável definida é do tipo variante. Isso quer dizer que o tipo de dados é definido de acordo com a informação armazenada naquele momento. O tipo de dados de determinada variável também pode ser modificado ao longo da execução da aplicação, conforme seu conteúdo é alterado.

Como mostra a Figura 4.1, o JavaScript, como toda linguagem orientada a objetos, está fundamentado nos conceitos de funções, objetos, eventos, propriedades e métodos.



Figura 4.1 – Funções, objetos, eventos, atributos e métodos.

## 4.1 Primeiro Script

A seguir, observe o código-fonte do primeiro script que será criado. Esse script é parte de uma página com conteúdo HTML que deve ser gravada com a respectiva extensão.

</>

```
<!DOCTYPE html>

<html>
  <head>
    <title>Introdução ao JavaScript</title>
```

```
<meta charset="UTF-8">
</head>
<body onload="iniciar()">
<script type="text/javascript">
function iniciar() {
document.write("<h1>" + document.title
+ "</h1>");
document.write("Olá pessoal!<br>");
}
</script>
</body>
</html>
```

ola.html

Esse exemplo consiste basicamente em utilizar o evento onload da página HTML para realizar a chamada da função iniciar(). A função, por sua vez, utiliza o método write do objeto document para escrever um determinado conteúdo na área de trabalho do navegador. O resultado obtido após a execução da página no navegador é mostrado na Figura 4.2.



Figura 4.2 – Utilizando o JavaScript.



Figura 4.3 – Mostrando propriedades de um objeto.

## 4.2 Variáveis em JavaScript

As variáveis em JavaScript não possuem um tipo de dados no momento de sua declaração. Elas assumem 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, naquele momento, ela é do tipo de dados String.

Em seguida, as variáveis recebem um valor numérico assumindo, isto é, nesse momento, são do tipo de dados inteiro. Ainda no exemplo, as variáveis finalizam como sendo do tipo de dados float.



---

```
<!DOCTYPE html>
<html>
<head>
<title>Bem-vindo ao JavaScript</title>
<meta charset="UTF-8">
</head>
<body onload="iniciar()">
<script type="text/javascript">
function iniciar() {
```

```

var x = "Olá!";
document.write("<h1>Variáveis em JavaScript</h1>");
document.write("A variável agora é String: " + x + "<br>");
x = 12 + 13;
document.write("Agora é do tipo de dados inteiro: " + x + "<br>");
x = 4.50 + 5.25;
document.write("Neste momento é float: " + x + "<br>");
}
</script>
</body>
</html>

```

---

variaveis.html

A Figura 4.4 mostra a página criada.



Figura 4.4 – Variáveis.

Observe no exemplo de código-fonte a seguir que os valores para o tipo de dados String devem ser especificados entre aspas simples ('') ou duplas (""). Por exemplo:



```
var x = "Olá";  
var y = 'amigo! ';
```

Valores numéricos, por sua vez, não possuem delimitadores. Como separador decimal, deve-se utilizar o caractere de ponto final (.), conforme mostrado no código-fonte a seguir.



```
var x = 12 + 13;  
var y = 4.50 + 5.25;
```

### 4.3 Operadores

Assim como outras linguagens de programação, o JavaScript implementa um grande conjunto de operadores. Veja no Quadro 4.1 os mais utilizados.

Quadro 4.1 – Conjunto de operadores

| 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                                 |

|             |        |                  |
|-------------|--------|------------------|
| Aritméticos | >      | Maior            |
|             | $\geq$ | Maior ou igual   |
|             | +      | Adição           |
|             | -      | Subtração        |
|             | *      | Multiplicação    |
|             | /      | Divisão          |
|             | %      | Resto da divisão |
|             | **     | Exponenciação    |
|             | ++     | Incremento       |
|             | --     | Decremento       |
| Lógicos     | $\&\&$ | E (AND)          |
|             | $\ $   | OU (OR)          |
|             | !      | NÃO (NOT)        |

No trecho de programa a seguir, ilustramos o uso de alguns operadores.



```
var x = 10;
x -= 4;
var y = x ** 2;
```

Na primeira linha, temos o operador de atribuição ( $=$ ), isto é, a variável `x` receberá o valor 10. Na linha seguinte, usamos o operador de atribuição de subtração ( $-=$ ); nesse caso, `x` recebe 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 (36). Dessa forma, `y` armazenará 36.



A Mozilla Developer Network apresenta a relação completa dos operadores. Acesse o link: <<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>>.

## 4.4 Depuração de Scripts

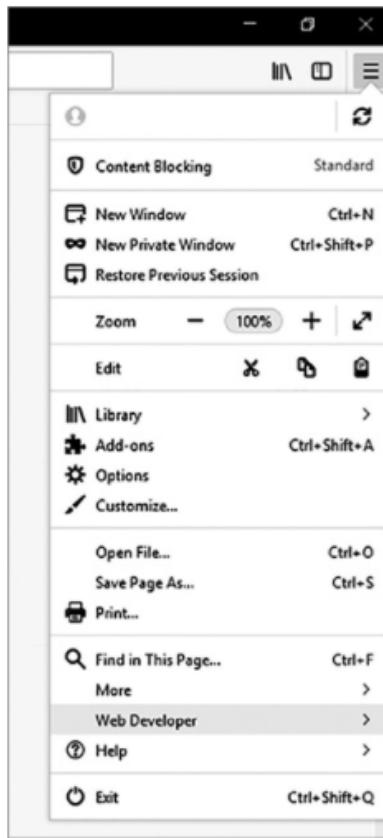


Figura 4.5 – Menu.

Os navegadores implementam ferramentas que permitem monitorar a execução das instruções JavaScript e emitir diretamente comandos para avaliar o seu funcionamento. Nesse tópico, abordaremos o uso dessas ferramentas no navegador Firefox.

Para começar, abra o navegador Firefox. Em seguida, clique na barra de ferramentas no botão "Open menu", localizado no canto superior direito da janela do navegador (Figura 4.5). Selecione a opção "Web Developer" e, em seguida, escolha "Web Console".

Na parte inferior da janela do navegador, você verá o "Web Console" (Figura 4.6). Nele é possível enviar comandos. Além disso, o "Web Console" também aponta as linhas de um script que apresentaram erro durante a execução do script.

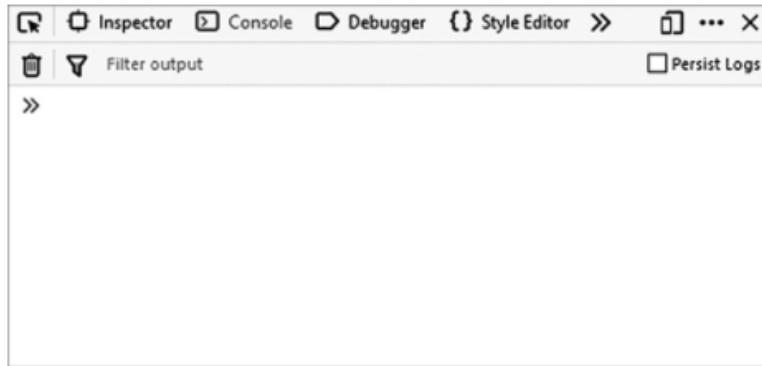


Figura 4.6 – Console.

Na Figura 4.7, observe que vamos utilizar o console para verificar o funcionamento de alguns operadores apresentados anteriormente.

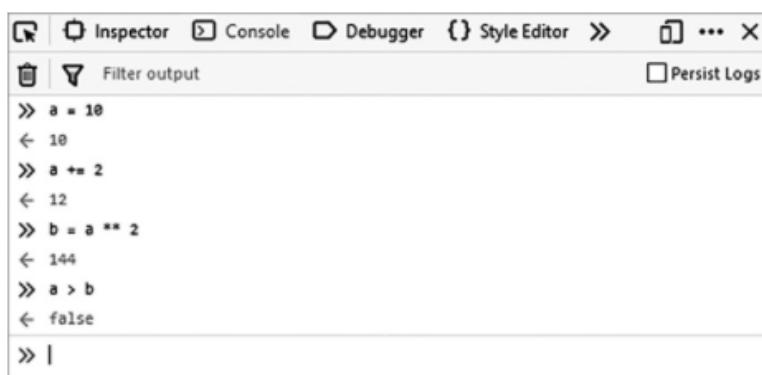


Figura 4.7 – Avaliando o uso dos operadores.

Essa é uma maneira muito rápida e fácil de verificar como determinado comando ou operação se comportará. O exemplo a seguir mostra como identificar erros nos scripts de uma página HTML. Deixamos intencionalmente um erro no código-fonte, pois a string atribuída à variável nome não foi adequadamente delimitada por aspas.



---

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

```
<title>Caixas de Diálogo e Mensagem</title>
<meta charset="UTF-8">
</head>
<body onload="iniciar()">
<script type="text/javascript">
function iniciar() {
var nome = "Zé Ninguém;
}
</script>
<h1>Erro</h1>
</body>
</html>
```

erro.html

Ao carregar a página no navegador, observe que o console indica que houve um erro e aponta a linha do arquivo (Figura 4.8). No exemplo, trata-se da linha 10.



Figura 4.8 – Identificação de erros.

Ao clicar na linha indicada, o navegador exibe o código-fonte da página, que destaca a linha com erro (Figura 4.9).

A screenshot of a web browser window titled "Erro". The address bar shows "http://localhost:8080/". The page content is an HTML document with the following code:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Erro</title>
5     <meta charset="UTF-8">
6   </head>
7   <body onload="iniciar()">
8     <script type="text/javascript">
9       function iniciar() {
10         var nome = "Zé Ninguém";
11       }
12     </script>
13     <h1>Erro</h1>
14   </body>
15 </html>
16
```

The line "var nome = "Zé Ninguém;" is highlighted with a dark gray background.

Figura 4.9 – Código-fonte da página com o erro em destaque.

Os ambientes integrados de desenvolvimento (IDEs) também auxiliam na tarefa de identificar erros nos scripts. Na Figura 4.10, o erro é indicado no NetBeans. Ao posicionar o ponteiro do mouse sobre o ícone que aparece no lugar da linha 10, a IDE mostrará o que causou o erro (Missing close quote). Nesse caso da Figura 4.10, faltou fechar as aspas duplas.

A screenshot of the NetBeans IDE showing a code editor with the following HTML and JavaScript code:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Erro</title>
5     <meta charset="UTF-8">
6     var nome = "Zé Ninguém";
7   </head>
8   <body>
9     <script type="text/javascript">
10    function iniciar() {
11      var nome = "Zé Ninguém";
12    }
13  </script>
14  <h1>Erro</h1>
15 </body>
16 </html>
```

A tooltip box appears over the line "var nome = "Zé Ninguém";" at line 6, containing the text "Missing close quote" with an arrow pointing to the closing double quote at the end of the line.

Figura 4.10 – Erro identificado por meio da IDE NetBeans.

## 4.5 Caixas de Mensagem e Diálogo

No exemplo a seguir, são mostradas as funções `prompt` e `alert` que permitem, respectivamente, receber e exibir informações em janelas (caixas de diálogo e mensagem) destacadas da página principal. Também é mostrado o método `toUpperCase()`, que permite converter determinada cadeia de caracteres em letras maiúsculas.



```
<!DOCTYPE html>
<html>
<head>
<title>Caixas de Diálogo e Mensagem</title>
<meta charset="UTF-8">
</head>
<body>
<script type="text/javascript">
function login() {
var nome = prompt("Qual o seu nome?",
"Zé Ninguém");
alert("Olá, " + nome.toUpperCase() +
".");
}
</script>
<h1>Identificação</h1>
<input type="button" value="Login"
onclick="javascript:login()">
</body>
</html>
```

caixas.html

Veja na Figura 4.11 um exemplo da execução da página quando o botão Login é pressionado.

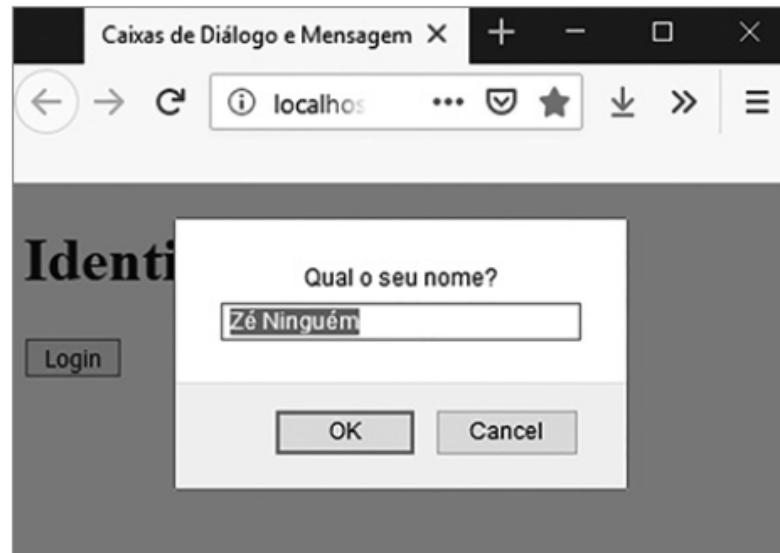


Figura 4.11 – Uso da função prompt.

Depois de preencher o campo nome e pressionar o botão OK, a função alert exibe o nome da pessoa (Figura 4.12).

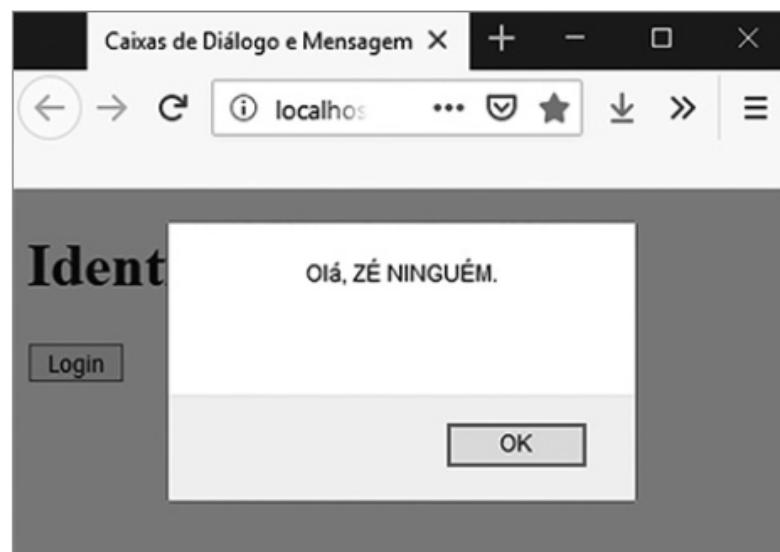


Figura 4.12 – Uso da função Alert.

## 4.6 Containers

Apesar de a linguagem HTML não permitir a inserção (ou modificação) do conteúdo após o carregamento da

página, ela oferece alguns elementos (*tags*) que atuam como container de dados.

Os elementos containers são aqueles que permitem marcar uma determinada porção do texto. Posteriormente, por meio de uma linguagem script, esse texto pode ter o seu conteúdo modificado após a página ser carregada.



```
<!DOCTYPE html>

<html>
<head>
<title> Elementos Containers</title>
<meta charset="UTF-8">
</head>
<body>
<script type="text/javascript">
function login() {
var nome = prompt("Qual o seu nome?",
"Zé Ninguém");
resultado.innerHTML = "Olá, " +
nome.toUpperCase() + ".";
}
</script>
<h1>Identificação</h1>
<input type="button" value="Login"
onclick="login()">
<hr>
<div id="resultado">Olá.</div>
</body>
</html>
```

[containers.html](#)

A Figura 4.13 mostra um exemplo de como a página poderá ser exibida após a execução do script.



Figura 4.13 – Texto exibido no elemento container.

## 4.7 Estruturas de Controle

A linguagem JavaScript implementa as tradicionais estruturas para controle de fluxo do programa. Veja cada uma delas a seguir.

- **if:** permite definir a execução ou não de determinado bloco de código, de acordo com a condição especificada. No exemplo a seguir, o valor da variável x é testado. De acordo com o seu valor, o comando if determina qual bloco de código deve ser executado.



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

```
<title>Estruturas de Controle</title>
<meta charset="UTF-8">
</head>
<body onload="iniciar()">
<script type="text/javascript">
function iniciar() {
var x = prompt("Digite um número?");
document.write("<h1>Estruturas de Controle</h1>");
if (x > 5) {
document.write("O valor de X é maior que cinco.<br>");
}
else if (x < 5) {
document.write("O valor de X é menor que cinco.<br>");
}
else {
document.write("O valor de X é igual a cinco.<br>");
}
}
</script>
</body>
</html>
```

---

estrutura-if.html

- **switch:** de acordo com o valor da variável testada pelo comando, é determinado o bloco de código que será executado. No exemplo a seguir, é possível obter o número que representa o dia da semana. Com base em seu valor, o nome que representa o dia é armazenado em uma variável, sendo posteriormente exibido na área de trabalho do navegador.



---

```
<!DOCTYPE html>
```

```
<html>
<head>
<title>Estruturas de Controle</title>
<meta charset="UTF-8">
</head>
<body onload="iniciar()">
<script type="text/javascript">
function iniciar() {
var hoje = new Date();
var diaSemana = hoje.getDay();
var nomeDiaSemana = "";
document.write("<h1>Estruturas de Controle</h1>");
switch(diaSemana) {
case 0:
nomeDiaSemana = "Domingo";
break;
case 1:
nomeDiaSemana = "Segunda-feira";
break;
case 2:
nomeDiaSemana = "Terça-feira";
break;
case 3:
nomeDiaSemana = "Quarta-feira";
break;
case 4:
nomeDiaSemana = "Quinta-feira";
break;
case 5:
nomeDiaSemana = "Sexta-feira";
```

```
break;  
case 6:  
    nomeDiaSemana = "Sábado";  
}  
document.write ("Hoje é " + nomeDiaSemana);  
}  
</script>  
</body>  
</html>
```

estrutura-switch.html

- **for:** o comando permite a repetição, caso a condição especificada seja verdadeira. No exemplo a seguir são mostrados os números inteiros de 0 a 4.



```
<!DOCTYPE html>  
  
<html>  
  <head>  
    <title>Estruturas de Controle</title>  
    <meta charset="UTF-8">  
  </head>  
  <body onload="iniciar()">  
    <script type="text/javascript">  
      function iniciar() {  
        document.write("<h1>Estruturas de Controle</h1>");  
        for (var i = 0; i < 5; i++) {  
          document.write("Contando ... " + i + "<br>");  
        }  
      }  
    </script>
```

```
</body>
```

```
</html>
```

---

estrutura-for.html

- **while:** similar ao comando for, executa a repetição de um bloco de programa. No exemplo a seguir, são mostrados os números entre 1 e 5.



```
<!DOCTYPE html>

<html>
<head>
<title>Estruturas de Controle</title>
<meta charset="UTF-8">
</head>
<body onload="iniciar()">
<script type="text/javascript">
function iniciar() {
var i = 1;
document.write("<h1>Estruturas de controle</h1>"); 
document.write("Contando ...");
while (i <= 5) {
document.write(" " + i);
i++;
}
}
</script>
</body>
</html>
```

---

estrutura-while.html

- **do-while**: é similar aos comandos for e while apresentados anteriormente. A diferença fundamental entre eles é que a verificação da condição ocorre apenas após a execução do bloco de programa a ser repetido, que sempre será executado pelo menos uma vez. Quando se utilizam for e while, se a condição inicial resultar em falso, o bloco a ser repetido nunca será executado. No exemplo a seguir são mostrados os números pares entre 0 e 10.



```
<!DOCTYPE html>
<html>
<head>
<title>Estruturas de Controle</title>
<meta charset="UTF-8">
</head>
<body onload="iniciar()">
<script type="text/javascript">
function iniciar() {
var i = 0;
document.write("<h1>Estruturas de controle</h1>");
document.write("Contando ...");
do {
document.write(" " + i);
i = i + 2;
}
while (i <= 10);
}
</script>
</body>
</html>
```

estrutura-do.html

## 4.8 Vetores

O objeto Array permite a definição de vetores cujos elementos podem ser acessados posteriormente por meio do respectivo índice.



```
<!DOCTYPE html>
<html>
<head>
<title>Montadoras</title>
<meta charset="UTF-8">
</head>
<body onload="exibir()">
<script type="text/javascript">
function exibir() {
var montadora = new Array("Fiat",
"Ford", "General Motors",
"Honda", "Nissan", "Renault",
"Volkswagen");
document.write("<h1>Montadoras</h1>");
for (i = 0; i < montadora.length; i++)
document.write (montadora[i] +
"<br>");
}
</script>
</body>
</html>
```

vetores.html

No exemplo a seguir, um vetor é declarado e inicializado com alguns itens. Em seguida, a rotina JavaScript acessa os elementos do vetor e os imprime na página HTML (Figura 4.14).

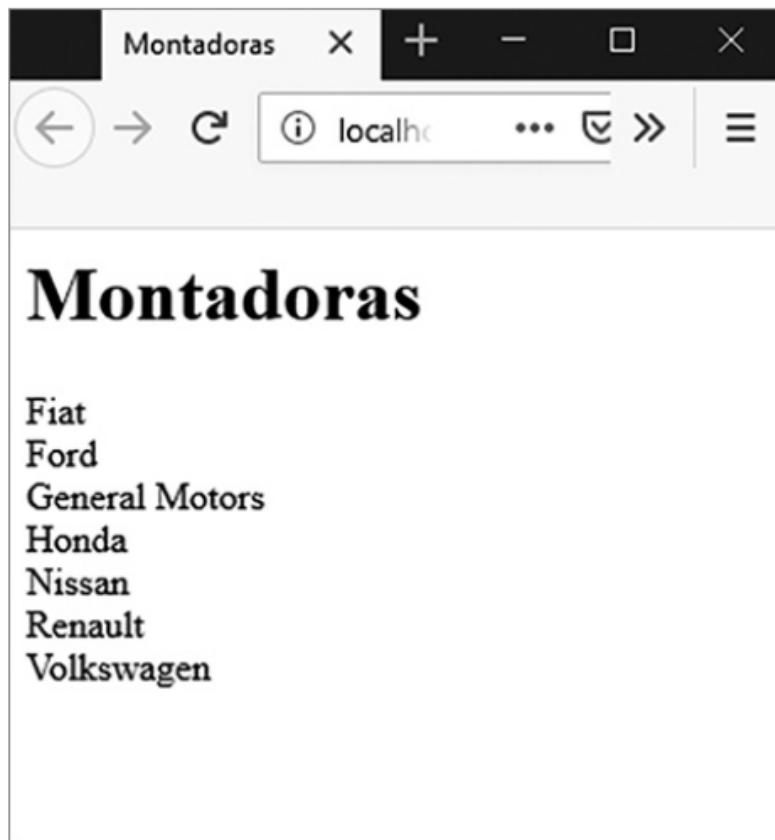


Figura 4.14 – Exemplo de uso de vetores.

## 4.9 Exibição de Imagens

A linguagem JavaScript é bastante versátil. Ela possibilita acessar os arquivos com imagens e permite que uma imagem seja trocada por outra. Nesse caso, não é necessário que a página HTML original seja recarregada. A partir dessa característica, é possível dinamizar as páginas, sem a necessidade de usar scripts com código complexo.



```
<!DOCTYPE html>
<html>
<head>
<title>Imagens</title>
<meta charset="UTF-8">
```

```
</head>
<body>
<script type="text/javascript">
function mudar(objeto, imagem) {
lampada.src = imagem;
}
</script>
<h1>Imagens</h1>
<a onmouseover="mudar('lampada',
'lamp-lig.png')"
onmouseout="mudar('lampada',
'lamp-desl.png')">

```

-----  
imagens.html

Na Figura 4.15, a imagem exibida na página é trocada sempre que o ponteiro do mouse entra ou sai da área ocupada pela imagem.

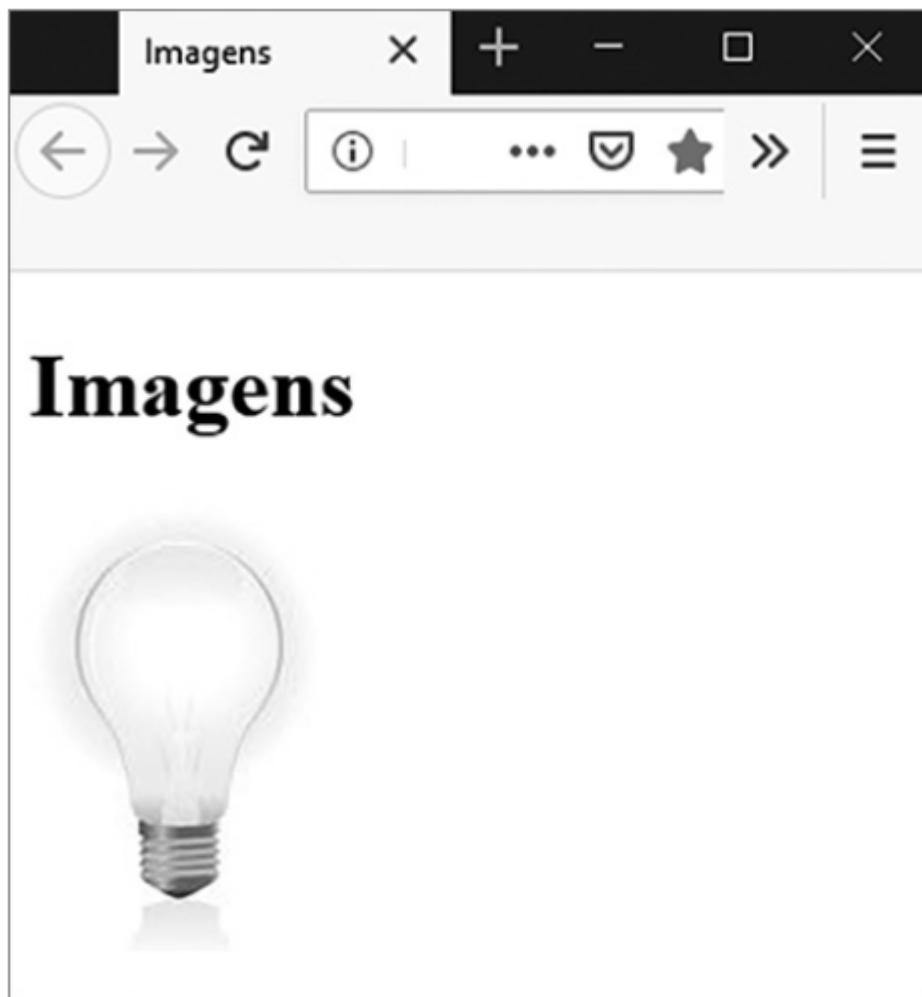


Figura 4.15 – Substituindo uma figura por meio do JavaScript.

## 4.10 Interação com Formulários

Com JavaScript, é possível acessar os elementos de formulários definidos em HTML. Dessa forma, é possível realizar verificações e alterações no conteúdo dos elementos, o que oferece mais possibilidades de interação com o usuário final da aplicação.



```
<!DOCTYPE html>

<html>
<head>
<title>Calculadora Simples</title>
<meta charset="UTF-8">
</head>
<body>
<script type="text/javascript">
function somar() {
res.value = parseInt(v1.value) + parseInt(v2.value);
}
</script>
<h1>Calculadora Simples</h1>
Valor 1: <input type="text" id="v1" value="0"><br>
Valor 2: <input type="text" id="v2" value="0"><br>
<input type="button" value="Somar"
onclick="somar()"><br>
Resultado: <input type="text" id="res" value="0" readonly>
</body>
</html>
```

somar.html

Na Figura 4.16 é possível observar o resultado da página após a interação do usuário.

The screenshot shows a mobile browser window titled "Calculadora Simples". The address bar displays "localho...". The main content area contains the title "Calculadora Simples" in large bold letters. Below it, there are three input fields: "Valor 1: 23", "Valor 2: 12", and a button labeled "Somar". Underneath these, there is another input field labeled "Resultado: 35". The browser interface includes standard navigation buttons (back, forward, search) and a menu icon.

Figura 4.16 – Exemplo de formulário.

#### 4.11 Melhorias na Página da Calculadora Básica

Após abordarmos os conceitos básicos de interação entre formulários em HTML e linguagem JavaScript, podemos realizar melhorias na aplicação mostrada na Figura 4.16.

O primeiro passo consiste em implementar as outras três operações aritméticas básicas. O modo mais fácil é criar um botão para cada operação juntamente da respectiva função. As outras alterações realizadas serão o suporte aos números reais (float), além de evitar a divisão de determinado número por zero.



```
<!DOCTYPE html>
```

```
<html>
<head>
<title>Calculadora Simples</title>
<meta charset="UTF-8">
</head>
<body>
<script type="text/javascript">
function somar() {
res.value = parseFloat(v1.value) + parseFloat(v2.value);
}
function subtrair () {
res.value = parseFloat(v1.value) -parseFloat(v2.value);
}
function multiplicar () {
res.value = parseFloat(v1.value) * parseFloat(v2.value);
}
function dividir () {
if (parseFloat(v2.value) === 0)
res.value = "Erro!";
else
res.value = parseFloat(v1.value) / parseFloat(v2.value);
}
</script>
<h1>Calculadora Simples</h1>
Valor 1: <input type="text" id="v1" value="0"><br>
Valor 2: <input type="text" id="v2" value="0"><br>
<input type="button" value="Somar"
onclick="somar()">
<input type="button" value="Subtrair"
onclick="subtrair()">
```

```
<input type="button" value="Multiplicar"
       onclick="multiplicar()">
<input type="button" value="Dividir"
       onclick="dividir()"><br>
Resultado: <input type="text" id="res" value="0" readonly>
</body>
</html>
```

calculadora.html

Observe na Figura 4.17 o resultado da subtração de dois valores digitados pelo usuário.

Calculadora Simples

Valor 1: 23

Valor 2: 12

Somar Subtrair Multiplicar Dividir

Resultado: 11

Figura 4.17 – Exemplo de formulário com subtração.

Outra implementação possível é criar uma única função que realiza todas as operações. Para isso, a função deve

receber um parâmetro que indique qual operação será executada pelo botão Somar. Observe o trecho de código-fonte a seguir.



```
<input type="button" value="Somar"  
onclick="calcular('+')">
```

Com base no valor do parâmetro, a função deve executar a operação solicitada. Observe o código-fonte a seguir.



```
<!DOCTYPE html>  
  
<html>  
<head>  
<title>Calculadora Simples</title>  
<meta charset="UTF-8">  
</head>  
<body>  
<script type="text/javascript">  
function calcular(op) {  
if (op === '+')  
res.value = parseFloat(v1.value) + parseFloat(v2.value);  
else if (op === '-')  
res.value = parseFloat(v1.value) - parseFloat(v2.value);  
else if (op === '*')  
res.value = parseFloat(v1.value) * parseFloat(v2.value);  
else {  
if (parseFloat(v2.value) === 0)  
res.value = 'Erro!';  
else  
res.value = parseFloat(v1.value) / parseFloat(v2.value);
```

```
}

}

</script>

<h1>Calculadora Simples</h1>

Valor 1: <input type="text" id="v1" value="0"><br>
Valor 2: <input type="text" id="v2" value="0"><br>

<input type="button" value="Somar"
      onclick="calcular('+')">
<input type="button" value="Subtrair"
      onclick="calcular('-')">
<input type="button" value="Multiplicar"
      onclick="calcular('*')">
<input type="button" value="Dividir"
      onclick="calcular('/')"><br>

Resultado: <input type="text" id="res" value="0" readonly>

</body>
</html>
```

calculadora-parametro.html

## 4.12 Botões de Rádio

Os botões de rádio são amplamente utilizados em formulários. Eles são indicados quando é necessário restringir os valores de uma determinada entrada de dados a um conjunto de opções possíveis. Com base no exemplo da calculadora, utilizaremos um conjunto de botões de rádio para que o usuário possa selecionar a operação a ser realizada.

</>

```
<!DOCTYPE html>

<html>
<head>
```

```
<title>Calculadora Simples</title>
<meta charset="UTF-8">
</head>
<body>
<script type="text/javascript">
function calcular() {
if (somar.checked === true)
res.value = parseFloat(v1.value) + parseFloat(v2.value);
else if (subtrair.checked === true)
res.value = parseFloat(v1.value) - parseFloat(v2.value);
else if (multiplicar.checked === true)
res.value = parseFloat(v1.value) * parseFloat(v2.value);
else {
if (parseFloat(v2.value) === 0)
res.value = "Erro!";
else
res.value = parseFloat(v1.value) / parseFloat(v2.value);
}
}
</script>
<h1>Calculadora Simples</h1>
Valor 1: <input type="text" id="v1" value="0"><br>
Valor 2: <input type="text" id="v2" value="0"><br>
Operação:
<input type="radio" id="somar" name="op" checked> Somar
<input type="radio" id="subtrair" name="op"> Subtrair
<input type="radio" id="multiplicar" name="op"> Multiplicar
<input type="radio" id="dividir" name="op"> Dividir<br>
<input type="button" value="Executar"
onclick="calcular()"><br>
```

```
Resultado: <input type="text" id="res" value="0" readonly>
</body>
</html>
```

---

calculadora-radio.html

A Figura 4.18 mostra a implementação da página da calculadora básica usando os botões de rádio.

The screenshot shows a web browser window with the title "Calculadora Simples". The address bar displays "localhost:8383/". The main content area contains the following elements:

- Two input fields labeled "Valor 1:" and "Valor 2:", both containing the value "12".
- A label "Operação:" followed by four radio buttons:
  - radio button for "Somar" (Add)
  - radio button for "Subtrair" (Subtract)
  - radio button for "Multiplicar" (Multiply) (selected)
  - radio button for "Dividir" (Divide)
- A button labeled "Executar" (Execute).
- An output field labeled "Resultado:" containing the value "408".

Figura 4.18 – Botões de rádio.

#### 4.13 Caixas de Seleção

A caixa de seleção é outro elemento disponível em formulários, que possibilita limitar a entrada de dados a um conjunto predeterminado de valores. Observe que, no formulário HTML, devemos utilizar a etiqueta `select` e acrescentar também as respectivas opções. Veja o trecho de código-fonte a seguir.



---

```
Operação: <select id="op">
<option value="">(Escolha uma opção)</option>
```

---

```
<option value="+>Somar</option>
<option value="->Subtrair</option>
<option value="*>Multiplicar</option>
<option value="/">Dividir</option>
</select>
```

---

Agora, observe que a função calcular deve determinar o valor escolhido pelo usuário na caixa de seleção. Isso é feito por meio da propriedade `value`.



---

```
function calcular() {
  if (op.value === '+')
    res.value = parseFloat(v1.value) + parseFloat(v2.value);
  else if (op.value === '-')
    res.value = parseFloat(v1.value) - parseFloat(v2.value);
  ...
}
```

---

Observe a implementação completa da página.



---

```
<!DOCTYPE html>

<html>
  <head>
    <title>Calculadora Simples</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <script type="text/javascript">
      function calcular() {
```

```
if (op.value === '+')
res.value = parseFloat(v1.value) + parseFloat(v2.value);
else if (op.value === '-')
res.value = parseFloat(v1.value) - parseFloat(v2.value);
else if (op.value === '*')
res.value = parseFloat(v1.value) * parseFloat(v2.value);
else if (op.value === '/') {
if (parseFloat(v2.value) === 0)
res.value = 'Erro!';
else
res.value = parseFloat(v1.value) / parseFloat(v2.value);
}
else
alert('Escolha a operação desejada!');
}

</script>
<h1>Calculadora Simples</h1>
Valor 1: <input type="text" id="v1" value="0"><br>
Valor 2: <input type="text" id="v2" value="0"><br>
Operação:
<select id="op">
<option value="">(Escolha uma opção)</option>
<option value="+>Somar</option>
<option value="->Subtrair</option>
<option value="*>Multiplicar</option>
<option value="/">Dividir</option>
</select><br>
<input type="button" value="Executar"
onclick="calcular() "><br>
Resultado: <input type="text" id="res" value="0" readonly>
```

```
</body>
```

```
</html>
```

calculadora-selecao.html

O resultado da execução do script gera um resultado semelhante ao mostrado na Figura 4.19.

The screenshot shows a browser window with the title "Calculadora Simples". The address bar displays "localhost:8383/". The main content area has the heading "Calculadora Simples". Below it, there are four input fields: "Valor 1" containing "45", "Valor 2" containing "15", "Operação" dropdown set to "Dividir", and a "Resultado" field containing "3". A "Executar" button is located between the operation dropdown and the result field.

Figura 4.19 – Utilização da caixa de seleção.

#### 4.14 Validação dos Campos do Formulário

Um aspecto importante do JavaScript é permitir a verificação e a validação local dos dados de um formulário. Dessa forma, não é necessário realizar requisições ao servidor web, o que aumenta significativamente a capacidade de interação com o usuário e melhora o desempenho da página.

A página que será criada deve mostrar descrição de um produto, preço unitário e quantidades para, posteriormente, calcular o preço total. Os dados que o usuário digitará no formulário devem ser validados com base nas seguintes regras:

- a) Todos os campos devem ser obrigatoriamente preenchidos.
- b) O campo quantidade deverá aceitar um número inteiro entre 1 e 999.
- c) O campo preço unitário deverá aceitar apenas números reais positivos.

Para a realizar a validação, existe o evento `onblur`, o qual permite executar uma função quando o campo de texto perde o foco. Isto é, o cursor deixa o campo quando o botão esquerdo do mouse é pressionado em outra área da tela ou

quando se pressiona a tecla de tabulação. Veja a seguir o exemplo para validação do campo descrição.



```
<input type="text" id="desc" value="" size="20" maxlength="20" onblur="validarDescricao()" />
```

Em JavaScript, a função que seria chamada pelo evento onblur pode ser escrita da maneira mostrada a seguir. Inicialmente, verificamos se o valor do campo desc (descrição do produto) está vazio. Na comparação, o método `trim` desconsidera os espaços em branco que, eventualmente, possam ocorrer no início e/ou no fim do valor digitado no campo.



```
function validarDescricao() {  
    if (desc.value.trim() === "") {  
        desc.style.background = "yellow";  
        alert("Preencha a descrição do produto!");  
        return false;  
    }  
    else {  
        desc.style.background = "white";  
        return true;  
    }  
}
```

Quando o campo estiver vazio, ele será destacado em amarelo e uma mensagem será mostrada (Figura 4.20). Além disso, quando a validação falhar, a função retornará falso; caso contrário, o retorno será verdadeiro.

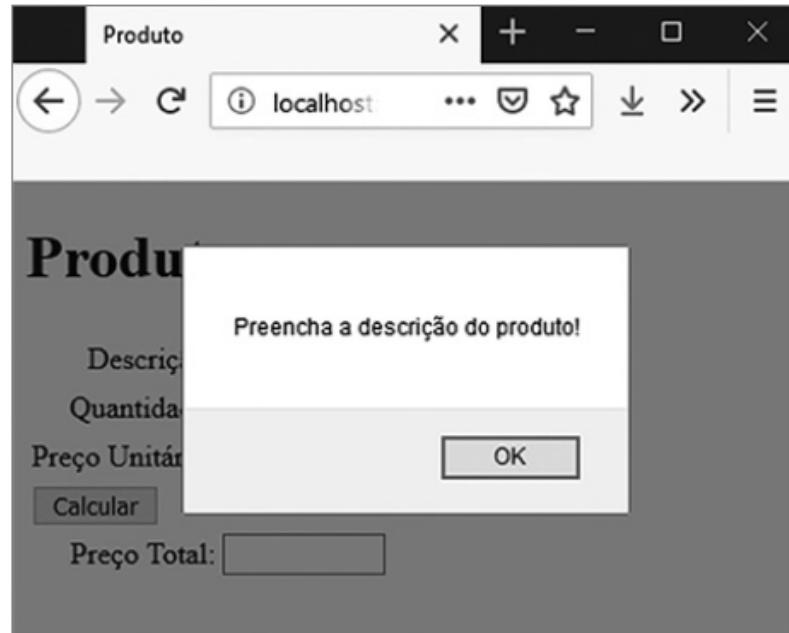


Figura 4.20 – Alerta que o campo não foi preenchido.

A seguir, apresentamos a implementação completa.



```
<!DOCTYPE html>

<html>
<head>
<title>Produto</title>
<meta charset="UTF-8">
</head>
<body>
<script type="text/javascript">
function validarDescricao() {
if (desc.value.trim() === "") {
desc.style.background = "yellow";
alert("Preencha a descrição do produto!");
}
}
```

```
return false;
}
else {
desc.style.background = "white";
return true;
}
}

function validarQuantidade() {
var erro = false;
if (qtd.value.trim() === "") {
erro = true;
}
else {
if (isNaN(qtd.value) === true) {
erro = true;
}
else {
var nQtd = parseInt(qtd.value);
if (nQtd < 1 || nQtd > 999) {
erro = true;
}
}
}
}

if (erro === true) {
qtd.style.background = "yellow";
alert("A quantidade deve ser um número entre 1 e 999!");
}
else {
qtd.style.background = "white";
}
```

```
return (!erro);
}

function validarPreco() {
var erro = false;
if (unit.value.trim() === "") {
erro = true;
}
else {
if (isNaN(unit.value) === true) {
erro = true;
}
else {
var nUnit = parseFloat(unit.value);
if (nUnit <= 0.0) {
erro = true;
}
else {
unit.value = nUnit.toFixed(2);
}
}
}
}

if (erro === true) {
unit.style.background = "yellow";
alert("O preço unitário deve ser um número maior que zero!");
}
else {
unit.style.background = "white";
}
return (!erro);
}
```

```
function calcular() {
if (validarDescricao() && validarQuantidade() && validarPreco()) {
var nTotal = parseFloat(unit.value) * parseInt(qtd.value);
total.value = nTotal.toFixed(2);
}
}

</script>
<h1>Produto</h1>
<table border="0">
<tr>
<td align="right">Descrição:</td>
<td><input type="text" id="desc" value="" size="20" maxlength="20"
onblur="validarDescricao()" /></td>
</tr>
<tr>
<td align="right">Quantidade:</td>
<td><input type="text" id="qtd" value="" size="3" maxlength="3"
onblur="validarQuantidade()" /></td>
</tr>
<tr>
<td align="right">Preço Unitário:</td>
<td><input type="text" id="unit" value="" size="10" maxlength="10"
onblur="validarPreco()" /></td>
</tr>
<tr>
<td colspan="2"><input type="submit" value="Calcular" onclick="calcular()"/>
</td>
</tr>
<tr>
<td align="right">Preço Total:</td>
<td><input type="text" id="total" value="" size="10" readonly /></td>
```

```
</tr>
</table>
</body>
</html>
```

produto.html

A Figura 4.21 mostra o resultado devidamente calculado e formatado após a validação dos dados digitados.



Figura 4.21 – Cálculo do preço total.

#### 4.15 JavaScript Externo

Muitas vezes, a solução de um determinado problema computacional gera longos arquivos de código-fonte. A modularização é utilizada para permitir agrupar melhor o código-fonte, além de facilitar o entendimento e as tarefas de manutenção. Dessa forma, é possível criar um arquivo externo apenas com o código-fonte em JavaScript, mantendo-o separado do HTML, como mostra o exemplo a seguir. Desenvolveremos uma calculadora com uma interface gráfica "estilo" Windows.

Para começar, criaremos apenas a página com o conteúdo HTML.



```
<!DOCTYPE html>

<html>
<head>
<title>Calculadora</title>
<meta charset="UTF-8">
</head>
<body>
<script type="text/javascript" src="calculadora.js"></script>
<h2>Calculadora</h2>
<table bgcolor="#BBBBBB" border="0" cellpadding="3" cellspacing="0" summary="">
<tr>
<td colspan="4">
<input type="text" id="res" value="0" size="12" readonly>
<input type="hidden" id="v1" value="0">
<input type="hidden" id="op" value="">
</td>
</tr>
<tr>
<td><input type="button" id="off" value="Off" onclick='retornar()'></td>
<td><input type="button" id="lim" value=" C " onclick='limpar()'></td>
<td>&ampnbsp</td>
<td>&ampnbsp</td>
</tr>
<tr>
<td><input type="button" id="d7" value=" 7 " onclick='digito("7")'></td>
<td><input type="button" id="d8" value=" 8 " onclick='digito("8")'></td>
<td><input type="button" id="d9" value=" 9 " onclick='digito("9")'></td>
<td><input type="button" id="div" value=" / " onclick='operacao("/")'></td>
</tr>
<tr>
```

```
<td><input type="button" id="d4" value=" 4 " onclick='digito("4")'></td>
<td><input type="button" id="d5" value=" 5 " onclick='digito("5")'></td>
<td><input type="button" id="d6" value=" 6 " onclick='digito("6")'></td>
<td><input type="button" id="mul" value=" X " onclick='operacao("X")'></td>
</tr>

<tr>
<td><input type="button" id="d1" value=" 1 " onclick='digito("1")'></td>
<td><input type="button" id="d2" value=" 2 " onclick='digito("2")'></td>
<td><input type="button" id="d3" value=" 3 " onclick='digito("3")'></td>
<td><input type="button" id="sub" value=" - " onclick='operacao("-")'></td>
</tr>

<tr>
<td><input type="button" id="d0" value=" 0 " onclick='digito("0")'></td>
<td><input type="button" id="sep" value=" , " onclick='separador()'></td>
<td><input type="button" id="tot" value=" = " onclick='total()'></td>
<td><input type="button" id="adi" value=" + " onclick='operacao("+")'></td>
</tr>
</table>
</body>
</html>
```

calculadora-completa.html

Note, no trecho de código-fonte a seguir, que a chamada do arquivo externo será realizada por meio da própria etiqueta script, a partir do atributo **src**.

</>

```
<script type="text/javascript" src="calculadora.js"></script>
```

Na sequência, devemos criar o arquivo calculadora.js que contenha as funções em JavaScript.



```
function String2Number(valor) {  
    valor = valor.replace(",",".");  
    return(parseFloat(valor));  
}  
  
function digito(dig) {  
    if (res.value.length < 12) {  
        if (res.value != "0")  
            res.value = res.value + dig;  
        else  
            res.value = dig;  
    }  
}  
  
function total() {  
    if (op.value === "+")  
        res.value = parseFloat(v1.value) + parseFloat(res.value);  
    else if (op.value === "-")  
        res.value = parseFloat(v1.value) - parseFloat(res.value);  
    else if (op.value === "X")  
        res.value = parseFloat(v1.value) * parseFloat(res.value);  
    else if (op.value === "/") {  
        if (res.value !== 0)  
            res.value = parseFloat(v1.value) / parseFloat(res.value);  
        else  
            res.value = "Erro!";  
    }  
}  
  
function operacao(ope) {  
    v1.value = res.value;
```

```
op.value = ope;
res.value = "0";
}
function separador() {
if (res.value.indexOf(",") < 0)
res.value = res.value + ",";
}
function limpar() {
res.value = "0";
}
```

calculadora.js

Na Figura 4.22 observamos a aparência da página criada.



Figura 4.22 – Exemplo completo da calculadora.

## 4.16 Interação entre Páginas

A linguagem JavaScript possui objetos que permitem realizar a interação entre diferentes páginas HTML. No exemplo de código-fonte a seguir, detalharemos o código necessário para que a calculadora, desenvolvida no exemplo anterior, preencha um campo do tipo texto existente em uma outra página HTML.

Na página principal há uma função em JavaScript que abre a página HTML que contém o programa da calculadora. Essa função será chamada quando o usuário clicar no botão calculadora. Também implementaremos o campo texto, que deverá receber o valor disponível na calculadora.

Nessa página, será utilizado o método `open` do objeto `window`, que permite abrir uma nova instância do navegador, além de controlar parâmetros como tamanho, localização, barras de menu.



```
<!DOCTYPE html>

<html>
<head>
<title>Interação entre páginas</title>
<meta charset="UTF-8">
</head>
<body>
<script type="text/javascript">
function abrirCalc() {
window.open('calculadora-completa.html','Calculadora','toolbar=no, \
location=no,directories=no,status=no,menubar=no, \
scrollbars=no,resizable=no,menubar=no,top=250,left=450, \
width=200,height=300');
}
</script>
<h1> Interação entre páginas</h1>
Digite um valor ou utilize a
<input type="button" onclick="abrirCalc()" value="calculadora">;
<input type="text" id="valor" value="0">
</body>
</html>
```

A página da calculadora-completa.html deverá ser alterada para receber um botão que transfere o resultado para a página que a chamou e, posteriormente, fechar a janela. Veja o exemplo a seguir.



```
<input type="button" id="off" value="Off"  
onclick='retornar()'>
```

Em seguida, a função retornar deve ser acrescentada ao arquivo calculadora.js.



```
function retornar() {  
    window.opener.valor.value = res.value;  
    window.close();  
}
```

O funcionamento é simples: basta carregar a página principal e, ao clicar no link da calculadora, outra página contendo a aplicação é exibida. Ao pressionar o botão Off, o conteúdo que aparece no visor é transferido para o campo texto da página anterior. Ainda nesse exemplo, a janela que contém a calculadora é fechada (Figura 4.23).



Figura 4.23 – Interação entre as páginas.

#### 4.17 Requisição de Dados

Os valores obtidos por meio de formulários podem ser enviados para outra página. Nesse caso, devemos usar a propriedade `location` do objeto `window` para criar o caminho (URL) da página que deve ser carregada quando o usuário pressionar o botão Enviar. Juntamente do endereço da página, devemos acrescentar os parâmetros que serão enviados e os respectivos valores.



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

```
<title>Envio de Dados</title>
<meta charset="UTF-8">
<script type="text/javascript">
function gerarURL() {
window.location ="receber.html?nome=" + nome.value + "&email=" + email.value;
}
</script>
</head>
<body>
<h1>Envio de Dados</h1>
Nome: <input type="text" id="nome" size="20"><br>
Email: <input type="text" id="email" size="20"><br>
<input type="button" value="Enviar" onclick="gerarURL () ;">
</body>
</html>
```

enviar.html

A página que receberá os dados deve usar a propriedade `window.location.href` para obter a cadeia de caracteres (string) que contém o endereço (URL) da página e todos os valores que foram enviados. Em seguida, é necessário realizar a manipulação da cadeia de caracteres para separar e obter os valores desejados de cada um dos parâmetros.



```
<!DOCTYPE html>
<html>
<head>
<title>Requisição de Dados</title>
<meta charset="UTF-8">
<script type="text/javascript">
function obterValor(chave) {
url = window.location.href;
```

```
chave = chave.replace(/[\[\]]/g, '\\$&');
var regex = new RegExp('[?&]' + chave + '(=([^&#]*|&|#|$)');
var res = regex.exec(url);
if (!res)
return null;
if (!res[2])
return '';
return decodeURIComponent(res[2].replace(/\+/g, ' '));
}

function lerFormulario() {
var nome = obterValor("nome");
var email = obterValor("email");
var texto = "Nome: <b>" + nome + "</b><br>" +
"E-mail: <b>" + email + "</b><br>";
resultado.innerHTML = texto;
}

</script>
</head>
<body onload="lerFormulario()">
<h1>Requisição de Dados</h1>
<div id="resultado"></div>
</body>
</html>
```

---

receber.html

A Figura 4.24 mostra o resultado do processamento dos parâmetros recebidos pela página.



Figura 4.24 – Dados recebidos.

 **DICA**

Uma ótima referência para ampliar os conhecimentos em JavaScript é a Mozilla Developer Network. Para isso, acesse o link <<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>>.

#### VAMOS PRATICAR!

1. Desenvolva uma página web que receba quatro números reais digitados pelo usuário. Em seguida, utilizando JavaScript, calcule e exiba o valor da média desses números.
2. Desenvolva uma página web para uma determinada loja que precisa calcular o preço de venda de um produto. O cálculo deve ser efetuado por meio da multiplicação do preço unitário × quantidade vendida e, posteriormente, subtraído o valor do desconto. Considere todas as variáveis do tipo de dado real, que serão digitadas pelo usuário, e um arquivo externo de JavaScript que contenha as funções necessárias para resolver o problema.
3. A Lei de Ohm define que a resistência ( $R$ ) de um condutor é obtida por meio da divisão da tensão aplicada ( $V$ ) dividida pela intensidade de corrente elétrica ( $A$ ). Dessa forma, a partir de uma tensão e corrente digitadas pelo usuário em um formulário HTML, utilize o JavaScript para calcular e mostrar o valor da

resistência na mesma página web que contém o formulário.

4. Adotando um arquivo externo de JavaScript que contém as funções necessárias para resolver o problema, elabore uma página web que, a partir de um valor digitado pelo usuário e o respectivo prefixo, mostre a representação do valor nos demais prefixos. Veja um exemplo:

Digite o valor: **10.000**

Digite o prefixo: **M**

**Converter**

Resultado:

10.000.000 k

10 G

0,01 T

Adote como referência a Tabela 4.1.

Tabela 4.1 – Tabela de referência

| Prefixo  | Valor (decimal)                  |
|----------|----------------------------------|
| k (kilo) | $10^3$<br>(1000)                 |
| M (mega) | $10^6$<br>(1,000,000)            |
| G (Giga) | $10^9$<br>(1,000,000,000)        |
| T (Tera) | $10^{12}$<br>(1,000,000,000,000) |

5. Considere que a aprovação de um aluno em determinada disciplina requer uma média final maior ou igual a 6,0. Elabore uma página web que receba duas notas, realize o cálculo da média, exiba o valor calculado e uma imagem indicando se o aluno está aprovado ou reprovado.
6. A partir dos lados de um retângulo ou quadrado digitados pelo usuário, elabore uma página web que calcule e exiba o valor da área da figura, informe se é um retângulo ou um quadrado e exiba uma imagem que represente a respectiva figura geométrica. Lembre-se de que a área é obtida pela multiplicação da base (L) pela altura (A).
7. Considerando um número inteiro digitado pelo usuário, calcule e exiba o valor da sua fatorial em uma página web. 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 em uma página web, determine e exiba o maior número.

9. Desenvolva uma página HTML que receberá as informações por meio de um formulário e realizará a validação dos dados digitados. Para isso, considere 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 no formulário utilizando, para isso, um arquivo JavaScript externo:
- O campo sexo deverá aceitar apenas F (Feminino) ou M (Masculino).
  - O campo ano de nascimento deve aceitar um valor entre 1901 e 2001.
  - O campo ano de fabricação deverá ser um valor inteiro positivo.
  - O campo valor do veículo deve ser um número real positivo.
  - O campo porcentagem do bônus deverá ser um número real entre 0 e 25.
10. Adotando como referência o formulário desenvolvido no exercício anterior, crie uma função JavaScript para determinar o valor da apólice. Utilize os seguintes critérios para cálculo:
- 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%; e, para veículos de fabricação anterior a 1980, devemos utilizar 2,5% como base de cálculo.
  - Caso o segurado seja do sexo feminino, aplique um desconto 10% sobre o valor calculado no item a; caso contrário, acrescente 5% ao valor calculado no item a.
  - Se o segurado tiver menos de 30 anos ou mais de 60 anos, acrescente 20% ao valor da apólice após os cálculos realizados nos itens a e b.
  - A partir do valor apurado nos itens a, b e c, aplique o desconto com base na porcentagem de bônus informada pelo usuário.

# 5

## jQuery

### INICIANDO OS ESTUDOS

Neste capítulo, apresentaremos uma breve introdução à biblioteca jQuery. Essa é uma das bibliotecas mais populares e amplamente utilizadas para JavaScript. Seu uso facilita a implementação de muitos aspectos do JavaScript, possibilitando melhor produtividade no desenvolvimento para a internet.

Uma biblioteca de funções JavaScript muito utilizada e, provavelmente, a mais popular é a jQuery. Foi criada com o objetivo de simplificar a seleção de elementos, a criação de efeitos visuais e a manipulação de eventos. É uma ferramenta bastante versátil, que possibilita inclusive a criação de plugins.

Os arquivos necessários para o funcionamento do jQuery estão disponíveis no link <[jquery.com](http://jquery.com)> e podem ser instalados no seu servidor. Uma página que usa o Bootstrap deve carregar os seguintes arquivos:



```
<!--jQuery: Carregar por meio do servidor local ou usar CDN -->
<!-- Versão minimizada do jQuery -->
<script src="js/jquery.min.js"></script>
```

Os arquivos também estão disponíveis por meio de CDN (Content Delivery Network). Dessa forma, eles não serão instalados no seu servidor e podem ser acessados diretamente on-line.

## 5.1 Olá jQuery

O exemplo de código-fonte a seguir mostra alguns conceitos básicos do jQuery. Para começar, observe o uso do `$()`, que é utilizado para localizar um elemento dentro do HTML. Por exemplo, para identificar a caixa de texto que possui o `id` definido como `nome`, devemos usar `$("#nome")`; para obter o que o usuário digitou na caixa de texto, usamos o método `val`, ou seja, `$("#nome").val()`.



```
<html>
<head>
<title>Olá jQuery</title>
<meta charset="UTF-8">
<script src="https://code.jquery.com/jquery-3.4.1.min.js">
</script>
<script type="text/javascript">
$(document).ready(function() {
```

```
$("#aqui").click(function() {  
    alert ("Olá " + $("#nome").val());  
});  
});  
</script>  
</head>  
<body>  
<h1>Olá jQuery</h1>  
Qual o seu nome?  
<input type="text" id="nome" /><br />  
<input type="button" id="aqui" value="Clique aqui!" />  
</body>  
</html>
```

---

ola-jquery.html

Note que, no exemplo anterior, o uso do evento `$(document).ready` é gerado após todos os elementos HTML da página serem carregados. Dessa forma, todas as demais funções do jQuery devem ser executadas a partir dele. Assim, o evento associado ao clique do botão "aqui" está inserido dentro de `$(document).ready`. A Figura 5.1 exemplifica uma página após o usuário preencher o campo nome e pressionar o botão "Clique aqui!".

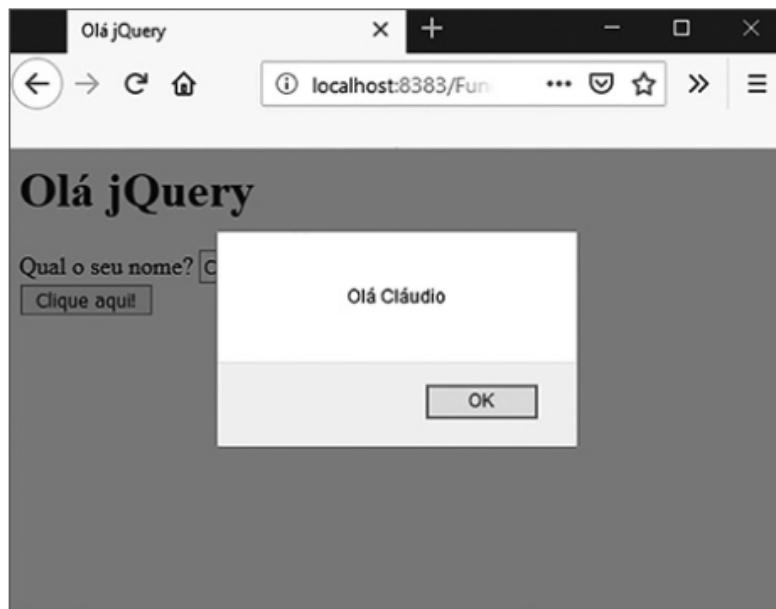


Figura 5.1 – Olá jQuery.

## 5.2 AJAX (Asynchronous Javascript and XML)

AJAX é uma forma de realizar uma requisição ao servidor. Nesse caso, o servidor retornará apenas os dados, e não uma página HTML completa com marcadores e conteúdo. Dessa forma, é possível carregar apenas uma parte da página, o que melhora o desempenho e possibilita melhor interação. O jQuery facilita a implementação de AJAX, oferecendo um método específico para isso, conforme mostra o código-fonte a seguir.



```
<!DOCTYPE html>

<html>
<head>
<title>A ligeira raposa</title>
<meta charset="UTF-8">
<script src="https://code.jquery.com/jquery-3.4.1.min.js">
</script>
<script type="text/javascript">
$(document).ready(function() {
```

```
$("#mais").click(function() {
$.ajax({url: "conteudo.txt", success: function(resultado) {
$("#info").html(resultado);
}});
});
});
});

</script>
</head>
<body>
<h1>A ligeira raposa</h1>
A ligeira raposa marrom salta sobre o cão preguiçoso. <input type="button" id="mais" value="Mais informações" />
<div id="info"></div>
</body>
</html>
```

ajax-jquery.html

O arquivo `conteudo.txt`, mostrado a seguir, contém o texto que será acrescentado à página HTML quando o usuário clicar no botão "Mais informações".



```
A ligeira raposa marrom salta sobre o cão
preguiçoso. A ligeira raposa marrom salta
sobre o cão preguiçoso. A ligeira raposa
marrom salta sobre o cão preguiçoso. A
ligeira raposa marrom salta sobre o cão
preguiçoso. A ligeira raposa marrom salta
sobre o cão preguiçoso. A ligeira raposa
marrom salta sobre o cão preguiçoso.
```

conteudo.txt

Na Figura 5.2 observamos a página carregada com o conteúdo inicial, definido no HTML.



Figura 5.2 – Página com o conteúdo inicial.

A Figura 5.3 mostra a página após a requisição AJAX, que é responsável por carregar o conteúdo do arquivo texto na div identificada com o id "info".



Figura 5.3 – Página após a requisição AJAX.

VAMOS PRATICAR!

1. Desenvolva uma página web com JavaScript e jQuery. A página deve receber quatro números reais digitados pelo usuário e, em seguida, calcular e exibir o valor da média dos números digitados.

2. Crie uma página web com JavaScript e jQuery para uma determinada loja que precisa calcular o preço de venda de um produto. O cálculo deve ser efetuado por meio da multiplicação do preço unitário × quantidade vendida e, posteriormente, subtrair o valor do desconto. Considere todas as variáveis do tipo de dado real que serão digitadas pelo usuário.
3. Desenvolva uma página HTML, com conteúdo definido a seu critério, que apresente o título formatado com a etiqueta <h1> e dois subtítulos formatados com a etiqueta <h2>. O conteúdo associado ao primeiro subtítulo deve ser apresentado na cor azul, fonte tamanho 12 px, alinhado à direita. O conteúdo relacionado ao segundo subtítulo deve estar alinhado à esquerda. Utilize jQuery e AJAX para que o conteúdo associado a cada subtítulo possa ser mostrado ou escondido quando se clicar no subtítulo. Esse conteúdo deve ser obtido a partir de dois arquivos texto relacionados aos respectivos subtítulos.
4. Coloque em uma página HTML um botão com o texto "Obter Data e Hora Atuais". Esse botão, quando pressionado, deve realizar uma requisição AJAX a um servidor da internet. Por exemplo: <http://worldclockapi.com/> e exibir em uma <DIV> a data e a hora atuais.
5. Altere o exercício anterior de modo que seja possível exibir na <DIV>, além da data e hora atuais, uma imagem que represente o dia ou a noite, conforme o horário.

# Bootstrap

## INICIANDO OS ESTUDOS

---

Neste capítulo abordaremos o Bootstrap, um dos frameworks mais populares para desenvolvimento do front-end de sites e aplicações para a web. Fácil de usar, oferece muitas opções de componentes visuais e possibilidade de layouts. Além disso, com pouco esforço, o Bootstrap permite a construção de interfaces sofisticadas para o usuário, minimizando o tempo necessário para o desenvolvimento.

A adoção de frameworks facilita o desenvolvimento de aplicações, pois implementa as funções mais comumente utilizadas em determinada parte do projeto. Por exemplo, podemos citar o acesso a banco de dados ou à interface com o usuário.

O Bootstrap é um framework HTML, CSS e JavaScript para desenvolvimento de interfaces responsivas para aplicações na internet. Essas interfaces funcionam adequadamente em computadores e dispositivos móveis.

Os arquivos necessários para o funcionamento do Bootstrap estão disponíveis no site <[getbootstrap.com](http://getbootstrap.com)> e podem ser instalados no seu servidor. Uma página que utiliza o Bootstrap deve carregar os seguintes arquivos:



```
<!-Bootstrap: Carregar por meio do servidor local ou usar CDN -->
<!-- Versão minimizada do CSS -->
<link rel="stylesheet"
      href="css/bootstrap.min.css">
<!-- Tema CSS opcional -->
<link rel="stylesheet"
      href="css/bootstrap-theme.min.css">
<!-- Versão minimizada do JQuery -->
<script src="js/jquery.min.js"></script>
<!-- Versão minimizada do Boostrap -->
<script src="js/bootstrap.min.js"></script>
```

Os arquivos também estão disponíveis por meio de CDN (Content Delivery Network). Dessa forma, não serão instalados no seu servidor, mas acessados diretamente por meio da conexão com a internet.

## 6.1 Olá Bootstrap

Conforme o exemplo de código-fonte a seguir, a estrutura básica de uma página com Bootstrap consiste em realizar algumas definições, além de carregar as Folhas de Estilo (CSS) e as bibliotecas em JavaScript necessárias.



```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Olá Bootstrap</title>
<!--Bootstrap: Carregar por meio do servidor local ou usar CDN -->
<!-- Versão minimizada do CSS -->
<link rel="stylesheet"
href="css/bootstrap.min.css">
<!-- Tema CSS opcional -->
<link rel="stylesheet"
href="css/bootstrap-theme.min.css">
<!-- Versão minimizada do JQuery -->
<script src="js/jquery.min.js"></script>
<!-- Versão minimizada do Boostrap -->
<script src="js/bootstrap.min.js"></script>
</head>
<body>
<h1>Olá Pessoal!</h1>
</body>
</html>
```

ola-bootstrap.html

Ao analisar a página criada, observe a seguir o uso da etiqueta **meta** para definir a compatibilidade do Bootstrap com o Internet Explorer.



```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
```

A etiqueta `meta` a seguir define a `viewport`, ou seja, a área para a exibição da página. Nesse caso, a propriedade `device-width` define que toda a área disponível deve ser usada para smartphones, tablets ou computadores. Essa etiqueta é fundamental para que a página se comporte de maneira responsiva.



```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Na sequência, são carregadas as folhas de estilo "bootstrap.min.css" e "bootstrap-theme.min.css". Também devem ser carregadas as bibliotecas JavaScript JQuery (jquery.min.js) e Bootstrap (bootstrap.min.js).

Na Figura 6.1, observe a página que foi criada após seu carregamento no navegador.



Figura 6.1 – "Olá Bootstrap".

## 6.2 Formulários com Bootstrap

Agora, vamos reescrever a página criada anteriormente, que realizava a validação dos campos do formulário e o cálculo do preço total de produto, só que usando o framework Bootstrap. Também retiraremos o código JavaScript da página HTML e o colocaremos em um arquivo à parte.



```
<!DOCTYPE html>

<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Produto</title>
    <!-- Bootstrap -->
    <!-- Versão minimizada do CSS -->
    <link rel="stylesheet"
          href="https://stackpath.bootstrapcdn.com/bootstrap/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>
<!-- 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 Boostrap -->
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js">
</script>
</head>
<body>
<script src="produto.js" type="text/javascript"></script>
<div class="container">
<h1>Produto</h1>
<div class="form-group">
Descrição:
<input type="text" id="desc" class="form-control" maxlength="20" />
</div>
<div class="form-group">
Quantidade:
<input type="text" id="qtd" class="form-control" maxlength="3" />
</div>
<div class="form-group">
Preço Unitário:
<input type="text" id="unit" class="form-control" maxlength="10" />
</div>
<div class="form-group">
<input type="submit" value="Calcular" class="btn btn-primary"
onclick="calcular()"/>
</div>
```

```
<div class="form-group">
Preço Total:
<input type="text" id="total" class="form-control" readonly />
</div>
<!-- Modal -->
<div class="modal" id="alerta">
<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">
Preencha o campo.
</div>
<div class="modal-footer">
<button type="button" class="btn btn-danger" data-dismiss="modal">Ok</button>
</div>
</div>
</div>
</div>
</body>
</html>
```

produto-bootstrap.html

Agora, exploraremos os recursos do Bootstrap utilizando a classe `form-group` para deixar o formulário responsivo; e a classe `modal` para criar uma caixa de mensagem que substituirá a função `alert` do JavaScript.

A seguir, temos o arquivo que implementará as funções JavaScript.

{JS}

```
function validarDescricao() {
    if (desc.value.trim() === "") {
        desc.style.background = "yellow";
        mensagem.innerHTML = "Preencha a descrição do produto!";
        $('#alerta').modal('show');
        return false;
    }
    else {
        desc.style.background = "white";
        return true;
    }
}

function validarQuantidade() {
    var erro = false;
    if (qtd.value.trim() === "") {
        erro = true;
    }
    else {
        if (isNaN(qtd.value) === true) {
            erro = true;
        }
        else {
            var nQtd = parseInt(qtd.value);
            if (nQtd < 1 || nQtd > 999) {
                erro = true;
            }
        }
    }
}
```

```
if (erro === true) {
    qtd.style.background = "yellow";
    mensagem.innerHTML = "A quantidade deve ser um número entre 1 e 999!";
    $('#alerta').modal('show');
}
else {
    qtd.style.background = "white";
}
return (!erro);
}

function validarPreco() {
var erro = false;
if (unit.value.trim() === "") {
    erro = true;
}
else {
    if (isNaN(unit.value) === true) {
        erro = true;
    }
    else {
        var nUnit = parseFloat(unit.value);
        if (nUnit <= 0.0) {
            erro = true;
        }
        else {
            unit.value = nUnit.toFixed(2);
        }
    }
}
if (erro === true) {
```

```
unit.style.background = "yellow";
mensagem.innerHTML = "O preço unitário deve ser um número maior que zero!";
$('#alerta').modal('show');
}
else {
unit.style.background = "white";
}
return (!erro);
}

function calcular() {
if (validarDescricao() && validarQuantidade() && validarPreco()) {
var nTotal = parseFloat(unit.value) * parseInt(qtd.value);
total.value = nTotal.toFixed(2);
}
}
```

---

produto.js

Observe na Figura 6.2 a página remodelada com o uso do Bootstrap. Observe a caixa de mensagem que é exibida, indicando que o campo quantidade deve ser corretamente preenchido. Além do design responsivo, é importante notar que a visualização da página fica bem mais fluida e profissional.

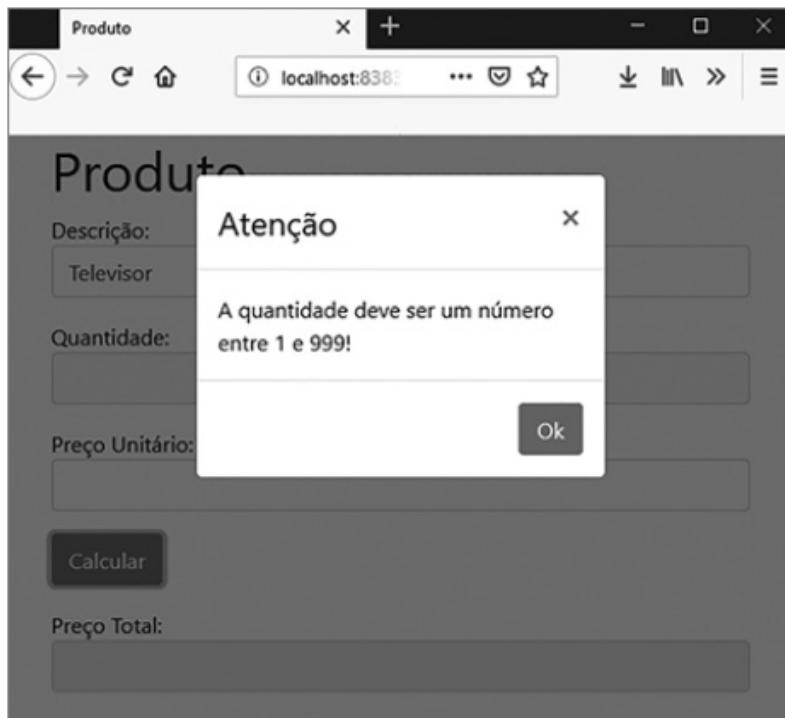


Figura 6.2 – Caixa de diálogo.

### VAMOS PRATICAR!

1. Desenvolva uma página web com JavaScript, jQuery e Bootstrap, que receba quatro números reais digitados pelo usuário e, em seguida, verifique se o usuário apenas digitou número. A página deve também calcular e exibir o valor da média dos números digitados.
2. Crie uma página web com JavaScript, jQuery e Bootstrap para uma determinada loja que precisa calcular o preço de venda de um produto. O cálculo deve ser realizado por meio da multiplicação do preço unitário × quantidade vendida e, posteriormente, subtrair o valor do desconto. Verifique se todos os campos do formulário foram preenchidos e considere todas as variáveis do tipo de dado real.
3. Considere o desenvolvimento de uma página HTML com Bootstrap e jQuery que recebe as informações por meio de um formulário e realiza a validação dos dados digitados. Para isso, use como base uma seguradora de veículos que precisa calcular o valor da apólice a partir das 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 devem ser realizadas no formulário utilizando, para isso, um arquivo JavaScript externo:
  - a. O campo sexo deverá aceitar apenas F (Feminino) ou M (Masculino).
  - b. O campo ano de nascimento deve aceitar um valor entre 1901 e 2001.

- 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.
4. Adotando como referência o formulário desenvolvido no exercício anterior, crie uma função que use jQuery para 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%; e, para os demais anos de fabricação, devemos utilizar 2,5% como base de cálculo.
  - b. Caso o segurado seja do sexo feminino, aplique um desconto de 10% sobre o valor calculado no item a; caso contrário, acrescente 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 nos itens a e b.
  - d. A partir do valor apurado nos itens a, b e c, aplique o desconto com base na porcentagem de bônus informada pelo usuário.
5. Em uma página HTML contendo jQuery e Bootstrap, crie um botão com o texto "Obter Data e Hora Atuais". Esse botão, quando pressionado, deverá realizar uma requisição AJAX e exibir na página a data e hora atuais obtidas de um servidor da internet. Por exemplo: <http://worldclockapi.com/>.

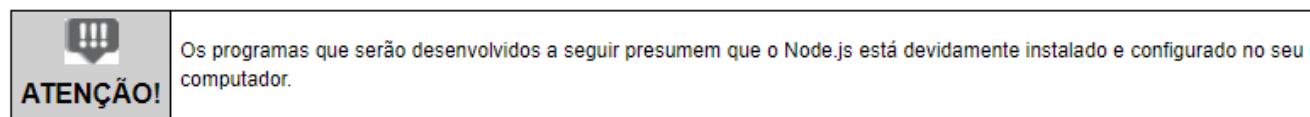
# Node.js

## INICIANDO OS ESTUDOS

---

Neste capítulo, estudaremos o interpretador JavaScript chamado Node.js, que tem como principal objetivo criar uma plataforma de execução de aplicações na linguagem JavaScript no lado servidor (e não apenas no lado cliente, executado em navegadores, como é comumente utilizado). Esse ambiente de execução no lado servidor abre a possibilidade de criação de diversos tipos de aplicações, com as facilidades e a escalabilidade que a linguagem JavaScript proporciona.

O Node.js é um ambiente de servidor de código aberto que possibilita a execução do JavaScript como uma linguagem de programação server side. Trata-se de uma multiplataforma gratuita, cujos instalador, tutoriais e documentação estão disponíveis no link <https://nodejs.org/pt-br/>.



## 7.1 Olá Node.js

Esse programa tem o intuito de oferecer 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 o código-fonte a seguir.

```
>-----  
node ola.js-----
```

Na Figura 7.1, aparece o resultado da execução do programa criado.

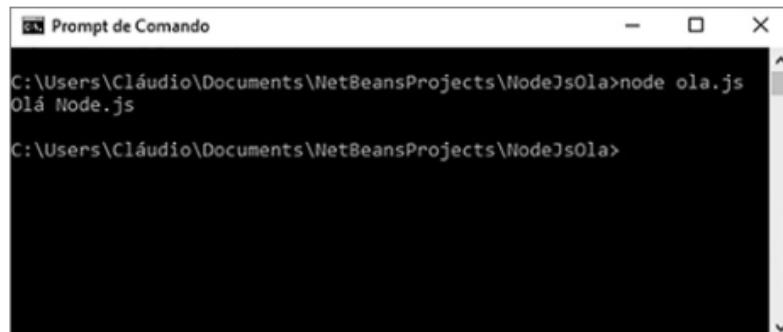


Figura 7.1 – Resultado da execução do programa.

Caso esteja usando o NetBeans, basta clicar no botão Executar o Projeto (F6), e o resultado será mostrado na janela Output (Figura 7.2).

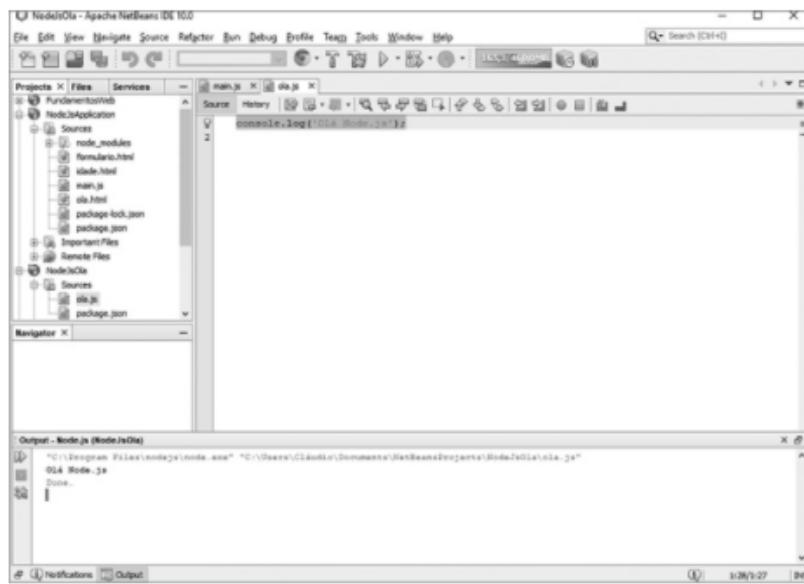


Figura 7.2 – Executando o programa no NetBeans.

## 7.2 Entrada de Dados por meio do Console

Quando há a necessidade de obter alguma entrada de dados por meio do console (teclado), deve-se utilizar a interface `readLine`, conforme mostra 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('Seu nome é ' + resposta);
  teclado.close();
});
```

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

No exemplo de código-fonte a seguir, será solicitado ao usuário que digite dois números. Em seguida, o valor da soma será calculado e exibido.

{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

### 7.3 Implementando um Servidor

A etapa seguinte é implementar um servidor de páginas web por meio do Node.js. O primeiro passo consiste em instalar o framework express, que é projetado para facilitar a criação de aplicações para a web. A instalação deve ser feita na pasta criada para essa aplicação, em que o npm deve ser executado da maneira apresentada a seguir.

&gt;-

```
npm install express
```

Em seguida, vamos elaborar o programa para criar um servidor local executando a 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.js

Ao analisar o programa, observe que o método `app.listen` cria um servidor na porta TCP indicada. Já o `app.get` determina a ação que deve ser realizada de acordo com o caminho passado. O exemplo seria a raiz do site (caractere '/'). O método `fs.readFile` abre o arquivo HTML apresentado a seguir. Em seguida, ele é enviado para o navegador por meio 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/bootstrap/4.3.1/css/bootstrap.min.css"
      integrity="sha384-  
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
      crossorigin="anonymous">

</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.js., como mostrado no código-fonte a seguir. Se estiver usando o NetBeans, apenas execute o projeto.



---

```
node servidor.js
```

---

Em seguida, abra o navegador e accese o site local na porta 8080, isto é, digite <localhost:8080/> na barra de endereço. Observe que a página HTML será exibida (Figura 7.3).

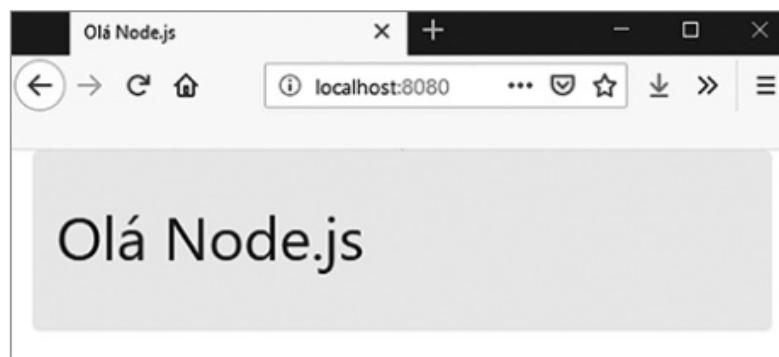


Figura 7.3 – Página carregada.

Utilize a combinação de teclas "Ctrl C" para finalizar a execução do servidor.

## 7.4 Construção de uma Aplicação

Com o Node.js é bastante simples trabalhar com os dados obtidos em formulários HTML. Essa próxima aplicação demonstra como podemos acessar os valores dos campos de formulários e processá-los no servidor.

O primeiro passo é criar a página HTML para implementar o formulário. Utilize apenas dois campos – o nome da pessoa e o ano em 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/bootstrap/4.3.1/css/bootstrap.min.css"
      integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUhcWr7x9JvoRxT2MZw1T"
      crossorigin="anonymous">
  </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" value="" size="25" class="form-control" />
</div>
<div class="form-group">
Ano de Nascimento: <input type="text" name="anonasc" value="" size="5"
class="form-control" />
</div>
<input type="submit" value="Calcular" class="btn btn-primary" />
</form>
</div>
</body>
</html>
```

---

formulario.html

Observe no trecho de código-fonte 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, será criada a página HTML para receber os dados processados pelo servidor. Observe que os dados que serão posteriormente enviados para essa página estão identificados como {{nome}}, {{anonasc}} 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/bootstrap/4.3.1/css/bootstrap.min.css"
      integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUhcWr7x9JvoRxT2M Zw1T"
      crossorigin="anonymous">

</head>
<body>
<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.html

Após a criação das páginas com o conteúdo HTML, a segunda etapa é 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('formulario.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.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();
});
});

```

---

main.js

Utilize o Node.js para executar o arquivo main.js., conforme mostrado a seguir; se estiver usando o NetBeans, apenas execute o projeto.



```
node main.js
```

No navegador, abra o site e digite <localhost:8080/> na barra de endereço. Preencha o formulário de maneira similar ao mostrado na Figura 7.4.

The screenshot shows a web browser window with the title bar 'Idade'. The address bar displays 'localhost:8080'. The main content area contains a form with the following fields:

- Nome:** José da Silva
- Ano de Nascimento:** 1990
- Calcular** (A dark grey button)

Figura 7.4 – Formulário para cálculo da idade.

Quando pressionar o botão "Calcular", o servidor processará os dados e enviará o resultado para a página "idade.html" (Figura 7.5).

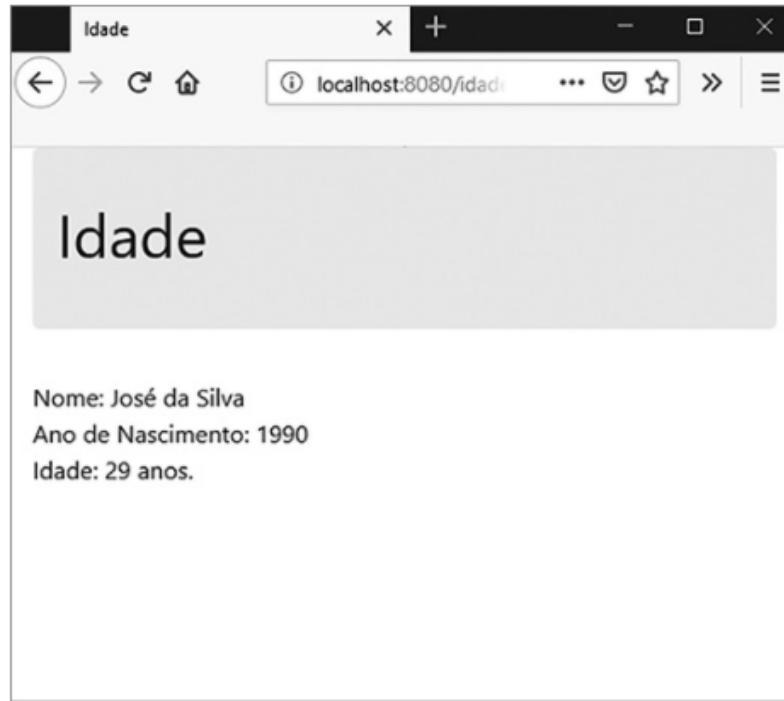


Figura 7.5 – Exibição dos dados processados.

Após a execução dessa aplicação, analisaremos, 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, você aprenderá como obter os 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();  
var bodyParser = require('body-parser');  
var urlencodedParser = bodyParser.urlencoded({ extended: true });  
-----
```

Observe no trecho a seguir que ele será o responsável pela criação do servidor que receberá as solicitações do navegador por meio 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 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();  
  });  
});
```

Veja, no trecho de código-fonte a seguir, que, quando o servidor receber a requisição "/idade", ele carregará o arquivo "idade.html" e obterá os dados que foram enviados por meio da submissão do formulário. Para receber esses dados, usamos a propriedade `req.body.nome`, em que nome corresponde ao valor da propriedade `id` do campo do formulário.

Em seguida, usaremos 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();
});

});
```

### VAMOS PRATICAR!

1. Desenvolva uma rotina em Node.js que receba quatro números reais digitados pelo usuário por meio do console e, em seguida, calcule e exiba a valor da média dos números.
2. Desenvolva um programa em Node.js para uma determinada loja que precisa calcular o preço de venda de um produto. O cálculo deverá ser efetuado por meio da multiplicação do preço unitário × quantidade vendida e, posteriormente, subtrair o valor do desconto. Considere todas as variáveis do tipo de dado real, que serão digitadas pelo usuário com uso do console.
3. A Lei de Ohm define que a resistência ( $R$ ) de um condutor é obtida por meio da divisão da tensão aplicada ( $V$ ) dividida pela intensidade de corrente elétrica ( $A$ ). Dessa forma, a partir de uma tensão e corrente digitadas pelo usuário por meio do console, calcule e mostre o valor da resistência.
4. Desenvolva um servidor usando Node.js que apresente uma página HTML contendo a data e hora atuais.
5. Considere que a aprovação de um aluno em determinada disciplina requeira uma média final maior ou igual a 6,0 (seis). Elabore uma aplicação Node.js que receba por meio 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.

# Interação com Banco de Dados

## INICIANDO OS ESTUDOS

---

Neste capítulo, abordaremos aspectos fundamentais de uma aplicação que utiliza sistemas de banco de dados. Para isso, foi escolhido o banco de dados nãorelacional MongoDB, um dos mais utilizados atualmente. Um banco de dados nãorelacional tem como principais vantagens escalabilidade, simplificação do projeto e melhor desempenho nas consultas. Também estudaremos a formatação de dados JSON (JavaScript Object Notation), adotada em larga escala para aplicações que necessitam organizar dados, realizar transferência entre plataformas e em banco de dados nãorelacionais, como o MongoDB.

Uma das principais vantagens de processar o JavaScript no lado servidor, por meio do Node.js, é a possibilidade de acesso aos Sistemas Gerenciadores de Banco de Dados (SGBDs).

## 8.1 MongoDB

A partir de agora, vamos desenvolver uma aplicação que usará como banco de dados o MongoDB, isto é, um banco de dados do tipo noSQL, que usa o JSON (JavaScript Object Notation) para realizar o armazenamento dos dados. É um banco muito utilizado em conjunto com o Node.js, pois 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 de tabelas e registros implementados nos bancos de dados relacionais.



DICA

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

Durante a instalação do MongoDB, você perceberá que pode realizar a instalação do banco de dados, como um serviço do sistema operacional, ou ser ativado manualmente, por meio da linha de comandos. Caso faça a opção por colocar o serviço no ar de forma manual, use o comando mostrado a seguir. (Observação: `dados` deve indicar o caminho completo para a pasta que armazernará 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 por meio do comando `show databases`.



```
show databases
```

A criação ou a seleção de um banco de dados é feita com o comando `use`. Por exemplo, o comando mostrado a seguir cria ou seleciona o banco chamado `loja`.



```
use loja
```

A visualização das coleções de um determinado banco de dados é realizada pelo comando `show collections`. Lembramos 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` produzirá um resultado vazio.

## 8.2 JavaScript Object Notation (JSON)

Considerando que a base para utilização do MongoDB é o JSON, apresentaremos alguns conceitos dessa notação que é amplamente utilizada. JavaScript Object Notation (JSON) ou Notação de Objetos JavaScript é uma notação leve para troca de dados. É de fáceis compreensão, leitura e escrita. Adota formato texto e é 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). Isso 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 exemplo a seguir.



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

Dessa forma, analisando a estrutura, entendemos que estamos especificando um par que apresenta o nome '`codigo`' que possui o valor 10. Em seguida, temos o par cujo nome é '`descricao`' e que tem o valor "`"Televisor"`", e o par chamado '`preco`' que tem o valor 1990.00.

A segunda estrutura do JSON é chamada de `array` e consiste em uma lista ordenada de valores que pode caracterizar

array, vetor, lista ou sequência, e coleção de valores ordenados. Um array é delimitado por colchetes, em que os valores são separados por vírgula, conforme mostrado no exemplo a seguir.

{JSON}

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

Um array pode conter objetos. Observe 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. Com o intuito de ilustrar a interação entre JSON e JavaScript, vamos implementar um exemplo que mostrará como inserir e remover itens em um array e listar o seu conteúdo. Para isso, considere um conjunto de produtos que apresenta como características código, descrição e preço.

A página HTML mostrada a seguir tem três caixas de texto para que o usuário digite as características do produto, além de um botão para inserção. A div identificada como "tabela" mostrará, posteriormente, 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>
```

json-produto.html

As funções JavaScript serão implementadas em um arquivo à parte, cujo conteúdo é mostrado a seguir. A variável `produtos` contém o array. Observe na função `inserir` que o método `push` adiciona um objeto ao array. Já o método `splice` é responsável pela remoção de objetos do array `produtos`. Esse método tem como parâmetros a posição do item no array e a quantidade de itens que deve ser removida (no exemplo, deixamos fixo o valor 1). Ou seja, apenas apagaremos o item que está na posição usada como parâmetro.



```
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 cellspacing='0' cellpadding='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 + ")'><image 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` monta uma tabela para exibir os dados que estão armazenados no array `produtos`. Essa função é chamada sempre que ocorre uma inserção ou remoção no array, de modo que as informações exibidas na página web permaneçam sempre atualizadas. A Figura 8.1 exemplifica a página criada.



Figura 8.1 – Dados do array JSON.



DICA

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

### 8.3 Inserção

No MongoDB não é necessário criar previamente a coleção. Dessa forma, basta inserir os dados que devem ser armazenados e, assim, a coleção será automaticamente criada. No exemplo a seguir, criaremos uma coleção chamada `produtos` e faremos inserção do 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. Nesse caso, usaremos o método `insertMany`, de acordo com o exemplo a seguir.



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

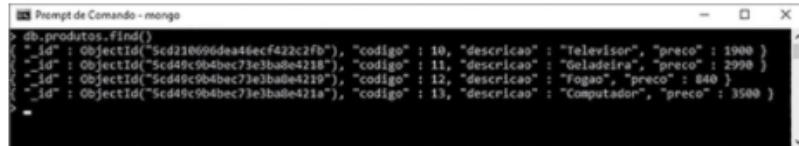
### 8.4 Consulta

O método `find` é usado para consultar os documentos que estão armazenados em determinada coleção. O exemplo a seguir mostra todos os documentos que existem na coleção `produtos`.



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

Na Figura 8.2, você observa o resultado da execução do método `find`, considerando a inserção dos dados realizada nos exemplos anteriores.



```
mongoshell> db.produtos.find()
[{"_id": ObjectId("5cd210696dea46ecf422c2fb"), "codigo": 10, "descricao": "Televisor", "preco": 1900 },
 {"_id": ObjectId("5cd49c9b4bec73e3ba8e4218"), "codigo": 11, "descricao": "Geladeira", "preco": 2998 },
 {"_id": ObjectId("5cd49c9b4bec73e3ba8e4219"), "codigo": 12, "descricao": "Fogao", "preco": 840 },
 {"_id": ObjectId("5cd49c9b4bec73e3ba8e421a"), "codigo": 13, "descricao": "Computador", "preco": 3500 }]
```

Figura 8.2 – Exibição dos dados processados.

No método `find` também podemos passar parâmetros que permitem filtrar os documentos que serão mostrados no resultado. Por exemplo, o comando a seguir exibe 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 esses conceitos, você verá no próximo exemplo os produtos com preço maior que 2000.



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

Observe a seguir que utilizaremos o operador `$in` para mostrar os documentos com código igual a 10 ou 11.



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

## 8.5 Alteração

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



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

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

## 8.6 Exclusão

Quando for necessário apagar um documento da coleção, é indicado usar o método `deleteOne`. No exemplo a seguir, removeremos o documento que possui o código igual a 12.



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

O método `deleteMany` deve ser empregado quando desejamos excluir vários documentos.

## 8.7 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 seguida, vamos mostrar uma aplicação simples que exibirá, no próprio console, os documentos armazenados na coleção `produtos`, já criada anteriormente. Observe que realizaremos a conexão ao MongoDB. Para isso, é indicado especificar o endereço (url) do servidor.

O método `connect` tentará realizar a conexão. Caso não seja possível, o método `assert.equal` encerrará imediatamente a aplicação.



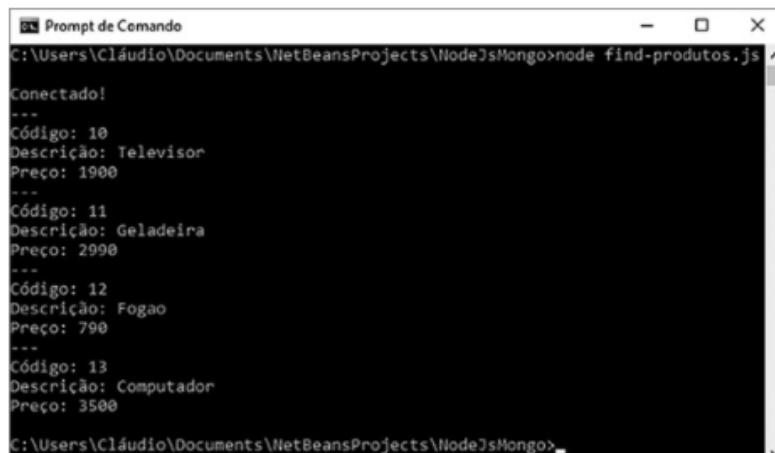
```
const MongoClient = require('mongodb').MongoClient;  
const assert = require('assert');  
const url = 'mongodb://localhost:27017';
```

```
const cliente = new MongoClient(url, { useNewUrlParser: true ,  
useUnifiedTopology: true });  
  
cliente.connect(function(erro) {  
assert.equal(null, erro);  
console.log("Conectado!");  
  
const banco = cliente.db('loja');  
  
var cursor = banco.collection('produtos').find();  
  
cursor.forEach(function(documento) {  
console.log("----");  
console.log("Código: " + documento.codigo);  
console.log("Descrição: " + documento.descricao);  
console.log("Preço: " + documento.preco);  
});  
cliente.close();  
});
```

---

find-produtos.js

Quando a conexão for estabelecida, a seleção do banco será realizada. No nosso exemplo, é loja. Utilizaremos, para isso, o método `cliente.db`. Para obter o conteúdo de uma coleção, é preciso criar um cursor para obter o retorno do método `find`. Para finalizar, o método `forEach` obtém os documentos recuperados. A execução da aplicação deverá produzir um resultado similar ao mostrado pela Figura 8.3.



```
Prompt de Comando  
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 8.3 – Exibição do conteúdo da coleção produtos.

## 8.8 Manipulação de Documentos por meio do Node.js

A aplicação a seguir insere um documento na coleção produtos. Note o uso do MongoClient para passar as informações para conexão ao banco de dados. Em seguida, no método connect do cliente, ocorre a seleção do banco de dados. A execução da inserção na coleção desejada é feita por meio do método insertOne.

{Node.js}

```
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);
console.log("Conectado!");
const banco = cliente.db('loja');
banco.collection('produtos').insertOne({
codigo: 14,
descricao: 'Cama de casal',
preco: 300
});
console.log("Inserido!");
cliente.close();
});
```

insert-produto.js

O próximo exemplo mostra um exemplo de rotina de alteração realizada a partir do Node.js. Nesse caso, é utilizado 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 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);  
  console.log("Conectado!");  
  const banco = cliente.db('loja');  
  banco.collection('produtos').updateMany(  
    { preco: { $gt: 2000 } },  
    { $mul: { preco: 0.9 } }  
  );  
  console.log("Alterado!");  
  cliente.close();  
});  
-----
```

update-produto.js

Para concluir os exemplos de operações básicas no MongoDB a partir do Node.js, veja como realizar a exclusão de um documento da coleção produtos, que possui código igual a 10. Para realizar essa operação, vamos utilizar o método **deleteOne**.

{Node.js} ~

```
-----  
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);  
  console.log("Conectado!");  
});  
-----
```

```
const banco = cliente.db('loja');
banco.collection('produtos').deleteOne(
{ codigo: 10 }
);
console.log("Excluído!");
cliente.close();
});
```

delete-produto.js

## 8.9 Aplicação Web com Node.js e MongoDB

No exemplo a seguir, uniremos os conceitos estudados até o momento para criar uma aplicação web. Essa aplicação acessa e manipula os dados armazenados em uma coleção.

Para começar, instalaremos o driver de acesso ao MongoDB e o framework Express, ambos utilizados anteriormente. Instalaremos também o Embedded JavaScript templating (EJS), que consiste em um framework para visualização. Ele oferece uma maneira fácil de transportar os dados da aplicação do lado do servidor para o cliente.



```
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`, crie as pastas `images` e `scripts` (Figura 8.4). Também escolha e copie duas imagens que indiquem as operações para editar e deletar.

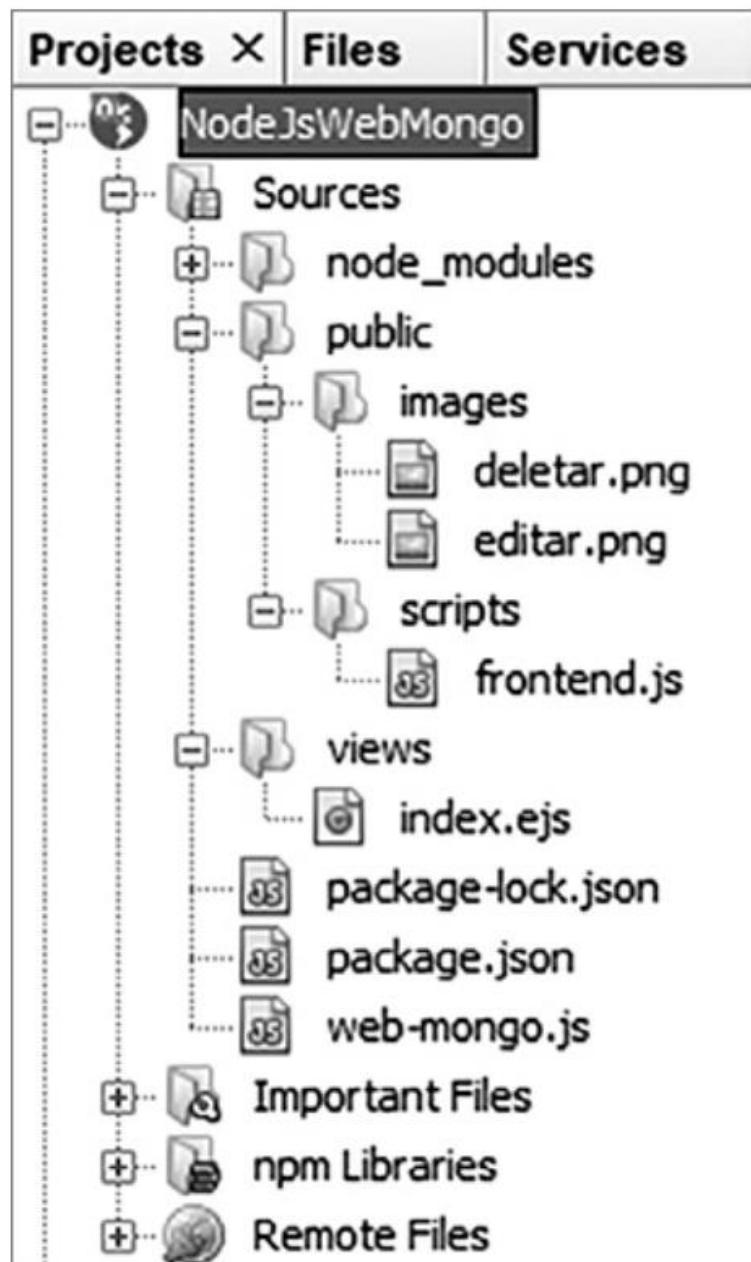


Figura 8.4 – Estrutura de pastas do projeto.

A página HTML da aplicação, apresentada a seguir, deve ser armazenada dentro da pasta `views`. Além disso, deve ter a extensão `ejs`, pois utilizaremos o framework Embedded JavaScript templating (EJS) nesse projeto.

Basicamente, essa página carrega scripts e folhas de estilo que são necessários para o funcionamento do jQuery e do BootStrap. Teremos também a tabela responsável pela exibição dos dados da coleção produtos, que está armazenada no MongoDB. Por fim, criaremos duas caixas de diálogo: uma para edição e inclusão de produtos; e outra para realizar a confirmação da exclusão de um item da coleção.

</>

```
<!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/bootstrap/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/jquery/3.3.1/jquery.min.js">
      </script>
    <!-- Versão minimizada do Popper -->
    <script
      src="https://cdn.jsdelivr.net/npm/popper.js@1.14.7/dist/umd/popper.min.js">
      </script>
    <!-- Versão minimizada do Boostrap -->
    <script
      src="https://stackpath.bootstrapcdn.com/bootstrap/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 -->
<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>
```

```
<!-- Fim do Modal: Formulário -->
<!-- Modal: Confirmação -->
<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>
<!-- Fim do Modal: Confirmação -->
</div>
</body>
</html>
```

---

index.ejs

O trecho de programa a seguir mostra o EJS que receberá um conjunto de dados a serem enviados pelo servidor (Node.js) para o cliente e, em seguida, renderizados. Os elementos do EJS são delimitados por `<% %>` e, entre suas vantagens, são especificados em JavaScript, o que facilita muito a criação das páginas. Por exemplo, podemos utilizar o método `toFixed()` para realizar a formatação do preço.



```
<% 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>
<% }) %>
```

Na próxima etapa, criaremos o servidor em Node.js. Ele realizará processamento das requisições, acesso e 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());
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(produto, 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 a seguir, observe que definiremos a configuração do servidor. A troca de dados entre as páginas será realizada usando JSON. Também utilizaremos 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, definiremos a ação que é realizada quando o cliente (navegador) solicitar a página raiz do servidor. O método `find` retornará os documentos cadastrados na coleção `produtos`. Em seguida, o resultado obtido será renderizado pelo EJS juntamente da 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 });
  });
});
```

Veja na Figura 8.5 um exemplo de página que exibe os documentos carregados de uma coleção implementada no MongoDB. Observe o botão "Novo...", que será responsável pela inclusão de produtos, e os ícones para edição e exclusão de determinado produto.

|  | Código | Descrição           | Preço   |
|--|--------|---------------------|---------|
|  | 11     | Geladeira           | 2990.00 |
|  | 12     | Fogao               | 711.00  |
|  | 13     | Computador          | 3500.00 |
|  | 14     | Cama de solteiro    | 240.00  |
|  | 15     | Mesa com 4 cadeiras | 600.00  |
|  | 16     | Cama de casal       | 800.00  |

Figura 8.5 – Exibição do conteúdo da coleção produtos.

Veja no trecho de programa a seguir a rotina detalhada para incluir um documento à coleção produto. Quando o servidor receber a requisição (/produtos/incluir), obteremos os dados enviados pelo cliente por meio da propriedade `req.body`, que está especificada em JSON. Na sequência, o método `insertOne` fará inserção dos 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(produto, 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 a exclusão de um documento (`/produtos/deletar`) funcionarão de maneira similar à inclusão. A diferença será a alteração dos respectivos métodos para `updateOne` e `deletarOne`.

A seguir, você observa a rotina utilizada para obter um documento e enviá-lo para a página HTML. 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);  
    });  
});
```

---

Para concluir a aplicação, implementaremos as funções JavaScript para atuar no lado do cliente (navegador). O arquivo com essas funções deve ser colocado na pasta `public/scripts`. Caso contrário, ele não terá permissão do servidor para a aplicação ser executada.

Em resumo, essas funções serão responsáveis por abrir as caixas de diálogo e realizar as requisições em **AJAX** (Asynchronous JavaScript and XML) para o servidor de aplicação.

{JS}

```
var codigo;
```

```
function incluir() {
    $("#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

Detalharemos agora o funcionamento do arquivo com as funções do JavaScript que serão executadas no cliente.

Inicialmente, você conhecerá a função incluir. Essa função colocará a operação que está sendo realizada no campo escondido (hidden), ou seja, 'i' para indicar uma inclusão. Essa função também limpará os demais campos (código, descrição e preço) ao carregar uma string vazia (""). Por fim, ela mostrará a caixa de diálogo.



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

Observe na Figura 8.6 a caixa de diálogo que possibilita a inclusão de um produto.

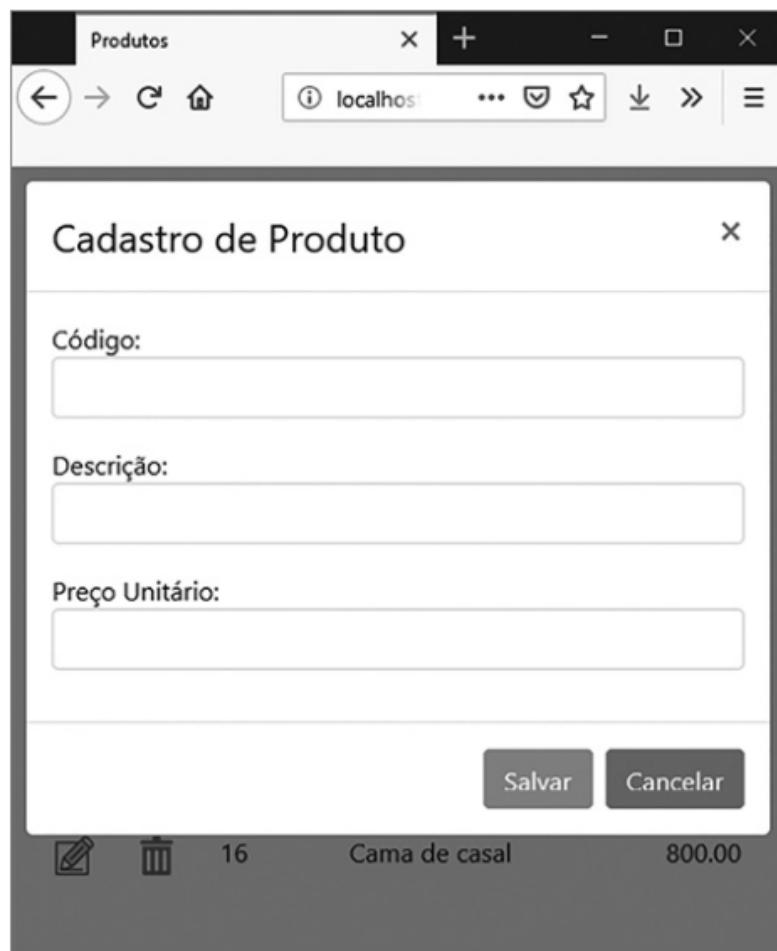


Figura 8.6 – Caixa de diálogo para inclusão.

No trecho de código-fonte a seguir, você observa a função que será executada no momento da submissão do formulário, ou seja, quando o usuário clicar no botão "Salvar" da caixa de diálogo.

Para começar, essa função obtém os dados digitados e os coloca em formato JSON. Isso ocorre por meio da variável `dados`.

Na sequência, com base na operação definida no campo escondido do formulário, determina-se se haverá inclusão ou alteração. Em seguida, ocorre a requisição AJAX para o servidor Node.js, responsável pela inclusão ou alteração.

Para finalizar, é preciso fechar a caixa de diálogo e carregar a página principal. Para isso, utiliza-se o método `window.location.reload()`, para que os dados incluídos ou alterados sejam mostrados na tabela.



```
$("#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);
}
});
});
```

---

Veja a seguir a rotina de alteração. O parâmetro é o código do produto. Será realizada uma requisição AJAX, que consulta o banco de dados e retorna os demais dados do produto que serão carregados no formulário. O campo escondido recebe o valor 'e' para indicar que ocorrerá uma edição (alteração) dos dados.



```
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');
}
```

A Figura 8.7 mostra uma caixa de diálogo exibindo os dados editáveis. Ao clicar no botão "Salvar", o formulário será enviado. Assim, o servidor Node.js, que fará a alteração dos dados, recebe a devida requisição.

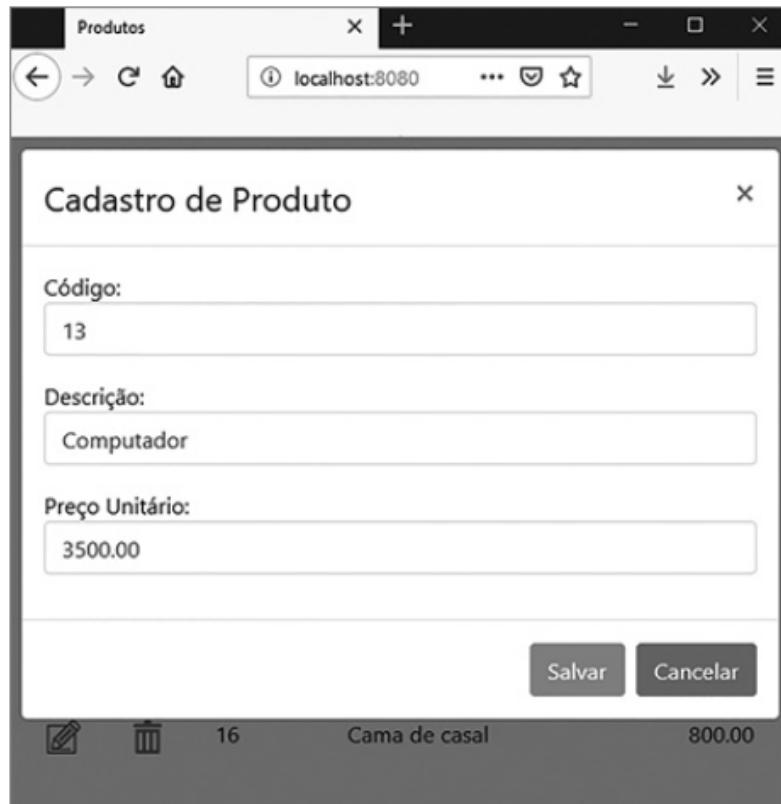


Figura 8.7 – Caixa de diálogo para alteração.

Ao clicar no ícone da lata de lixo (mostrado antes de cada produto exibido na página principal), é preciso excluir o respectivo documento armazenado no MongoDB. Porém, para evitar uma exclusão acidental, surge primeiro uma caixa de diálogo, em que o usuário pode confirmar a operação (Figura 8.8).

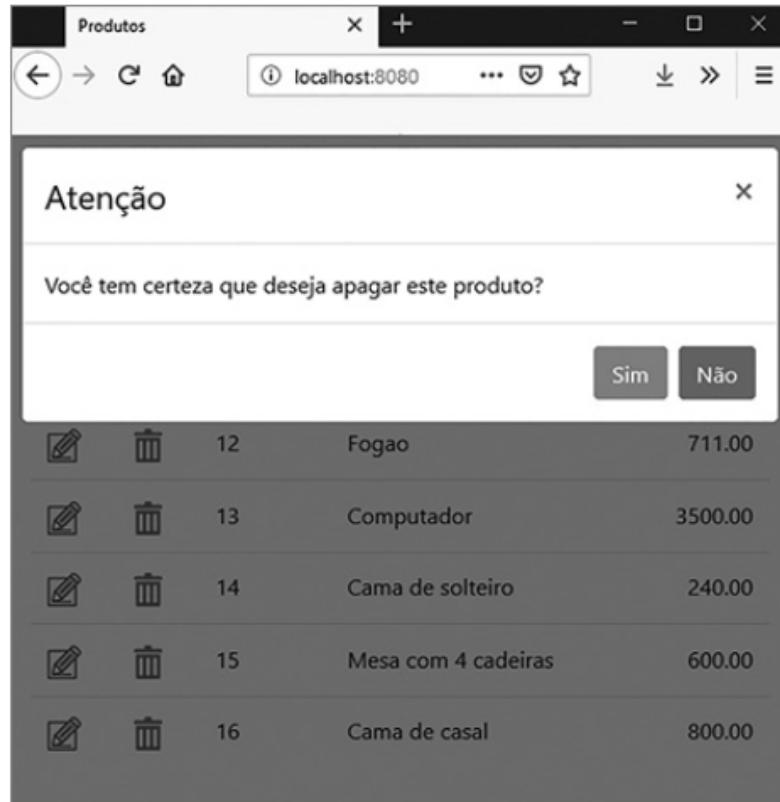


Figura 8.8 – Caixa de diálogo para confirmar a exclusão.

Ao clicar no botão "Sim", executaremos a função que realiza a requisição AJAX. Nesse caso, o documento que contém o referido produto será apagado do MongoDB; e a tabela que mostra os produtos será 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);
}
});
});
```

---

### VAMOS PRATICAR!

1. Descreva utilizando JSON uma estrutura que possibilite representar o CNPJ de uma determinada empresa, 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 que possibilite a inclusão de um documento no MongoDB, dentro da coleção "empresas". Os dados devem ser digitados por meio 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. E, em seguida, por meio de uma requisição AJAX, realize a inclusão de um documento no MongoDB, dentro da coleção "empresas".
5. A partir da estrutura desenvolvida no Exercício 1, crie um servidor Node.js que mostre uma página HTML, sendo que todos os documentos existentes na coleção "empresas" devem ser armazenados no MongoDB.

# Internet das Coisas (IoT)

## INICIANDO OS ESTUDOS

---

Neste capítulo, abordaremos a utilização da plataforma de prototipagem eletrônica Arduino para desenvolver aplicações a partir do conceito de Internet das Coisas (IoT ou *Internet of Things*). Utilizando o protocolo Firmata (criado para facilitar a comunicação entre software e hardware do Arduino) e a biblioteca Johnny-Five, desenvolveremos exemplos de aplicações que acessam sensores e atuadores por meio da internet. Para auxiliar a criação das aplicações IoT, utilizaremos o ThingSpeak, plataforma aberta de armazenamento e análise de dados na nuvem.

Atualmente, muitos dispositivos eletrônicos possuem acesso à internet (computadores, smartphones, televisores etc.). Além desses exemplos, temos também pequenos dispositivos microcontrolados que podem ser usados para controlar iluminação, portas, janelas e outros elementos de uma residência. Já outros dispositivos podem ser usados para controlar remotamente temperatura e umidade de uma estufa, por exemplo. Em linhas gerais, existe uma infinidade de possibilidades e aplicações. Esse é o conceito da Internet das Coisas (IoT, do inglês *Internet of Things*), que veremos a seguir.

## 9.1 Arduino

O Arduino é a plataforma de prototipagem eletrônica mais popular do momento. Além de um baixo custo, possui o conceito de hardware e software abertos e uma comunidade extremamente atuante. De uma maneira simples, possibilita a conexão entre sensores e atuadores, os quais permitem a interação de determinada aplicação com o mundo real.

A comunicação de uma determinada aplicação com o Arduino pode ser realizada por meio do protocolo Firmata. O Firmata é basicamente um protocolo de comunicação entre microcontroladores e um programa instalado em um computador.

Realizaremos agora uma implementação por meio de Node.js. Utilizaremos a biblioteca Johnny-Five, que é uma plataforma para computação física, robótica e IoT.

|  |   |
|--|---|
| <br><b>DICA</b> | Acesse <a href="http://johnny-five.io/">http://johnny-five.io/</a> para obter mais informações, pré-requisitos e procedimentos para instalação da plataforma Johnny-Five. |
|--|---|

## 9.2 Carga do Firmata no Arduino

O ambiente de desenvolvimento do Arduino é gratuito e pode ser baixado no link <<http://www.arduino.cc>>. Em seguida, basta executar o arquivo baixado para realizar a instalação.

Inicialmente, é preciso seguir alguns passos simples para estabelecer a comunicação entre a placa do Arduino e a linguagem de programação JavaScript.

Comece transferindo o programa específico "StandardFirmata" para a placa Arduino. O Firmata fará a ponte entre a programação em JavaScript e a placa Arduino.

Em seguida, abra o ambiente de desenvolvimento do Arduino, vá em "Arquivo"; depois, clique em "Exemplos"; e, por fim, "StandardFirmata". Observe a Figura 9.1.

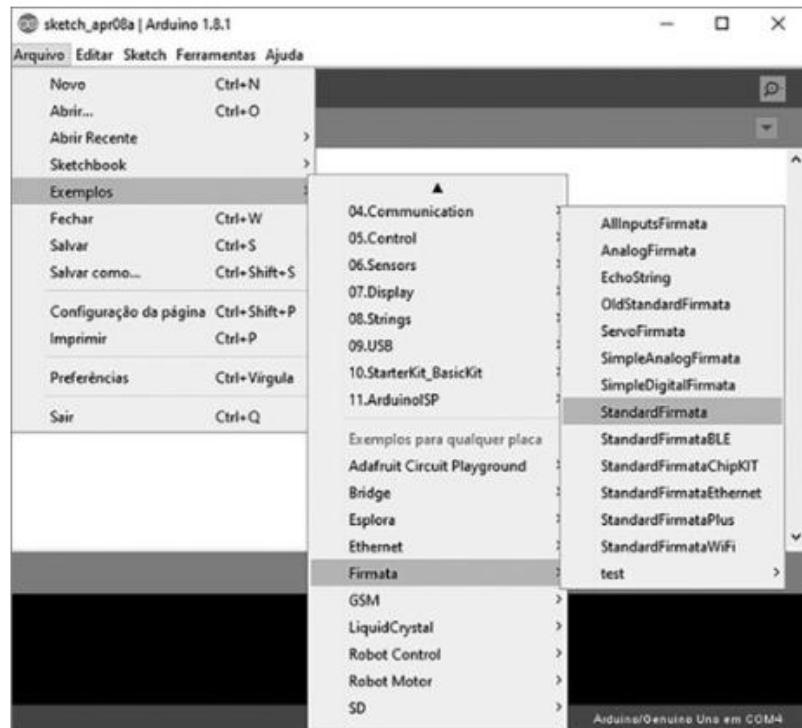


Figura 9.1 – Abrir o programa Firmata.

O código-fonte será exibido no ambiente de desenvolvimento do Arduino relativo ao programa "StandardFirmata". Para transferir o programa para a placa, vá até o menu "Ferramentas", escolha o item "Porta" e, em seguida, selecione a porta serial na qual o Arduino foi conectado. Agora, basta clicar no botão "Carregar" (Figura 9.2).

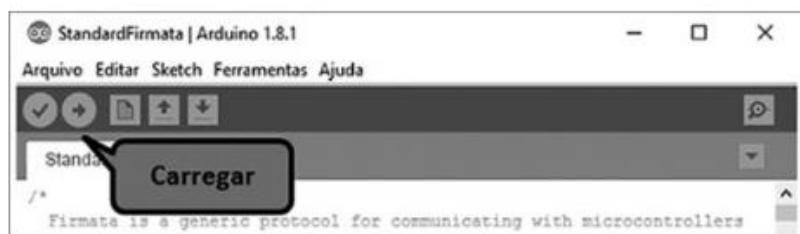


Figura 9.2 – Transferência do programa para o Arduino.

O programa transferido ficará armazenado na memória flash do Arduino até o que o outro programa seja gravado pelo ambiente de desenvolvimento do Arduino. Dessa forma, é possível desligar o Arduino e, quando ele for religado, o programa carregado será reiniciado automaticamente.

### 9.3 Interação com o Arduino

Conheceremos agora alguns detalhes sobre o funcionamento da placa Arduino. Os pinos digitais do Arduino (Figura 9.3) podem atuar como entrada ou saída. São usados para obter dados de dispositivos sensores digitais. Por exemplo, um botão para ativar ou desligar atuadores, como Diodos Emissores de Luz (LEDs), servo motores ou relês.



Figura 9.3 – Pinos digitais do Arduino Uno R3.

Por meio da biblioteca Johnny-Five, podemos utilizar um pino digital do Arduino para controlar o funcionamento de um LED. Um nível 1 (HIGH) colocado no pino acende o LED, enquanto um nível 0 (LOW) apaga o LED. Ou seja, nesse exemplo, o pino digital atua como saída. Veja a seguir o material necessário para que seja possível realizar a montagem do circuito eletrônico.

|   |  |
|---|--|
| <br>Relação<br>de<br>materiais | <ul style="list-style-type: none"><li>• 1 Arduino Uno R3 (ou qualquer outro modelo).</li><li>• 1 Resistor de 220 Ohms (vermelho, vermelho e marrom) ou 330 Ohms (laranja, laranja e marrom).</li><li>• 1 Led (qualquer cor).</li><li>• 1 Protoboard.</li><li>• Cabos de ligação.</li></ul> |
|---|--|

Após reunir o material necessário, realize a montagem de acordo com a Figura 9.4.

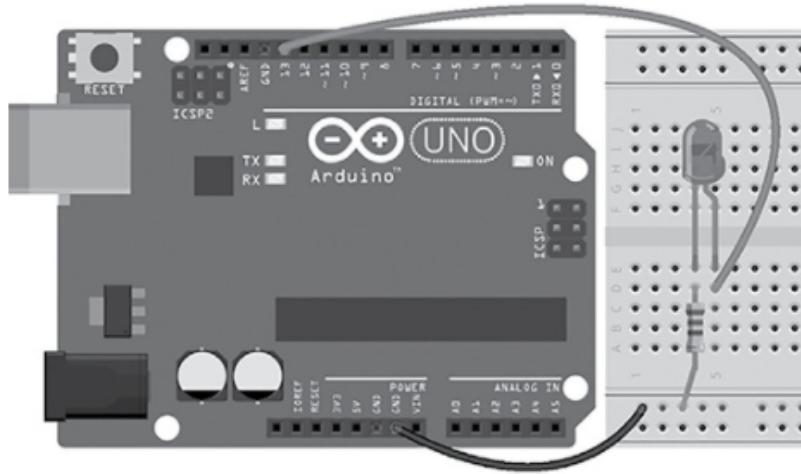


Figura 9.4 – Conexões.

Após criar a pasta em que o projeto será salvo, utilize o `npm` para realizar a instalação da biblioteca Johnny-Five.



```
npm install johnny-five
```

O programa importará a biblioteca Johnny-Five por meio da instrução `require`. A partir daí, criaremos um objeto para representar a placa Arduino. Em seguida, no evento `ready`, criaremos um objeto para o LED conectado ao pino digital 13 do Arduino. Por fim, executaremos o método `blink`, que fará com que o LED acenda e apague em intervalos de 500ms, isto é, a cada meio segundo.



```
(Node.js)  
var five = require("johnny-five");  
  
var arduino = new five.Board();  
arduino.on("ready", function() {  
  var led = new five.Led(13);  
  led.blink(500);  
});
```

pisca-pisca.js

## 9.4 Controle de um Dispositivo por meio da Internet

Nesse projeto, criaremos uma aplicação Node.js que permitirá ao usuário ligar ou desligar um LED por meio de uma página web. Utilizaremos exatamente o mesmo circuito eletrônico que foi montado para o exemplo anterior.

Após criar a pasta para salvar o projeto, utilize o `npm` para instalar a biblioteca Johnny-Five. Crie também a pasta `public`, onde ficará salvo o arquivo com as funções JavaScript executadas no lado do cliente (navegador).



```
-----  
npm install johnny-five  
-----
```

A página HTML mostrada a seguir utiliza o Bootstrap. Ela implementará apenas os botões que permitem ligar ou desligar o LED.



```
-----  
<!DOCTYPE html>  
  
<html>  
  <head>  
    <title>Controle de um dispositivo por meio da internet</title>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <link rel="stylesheet"  
          href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">  
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js">  
    </script>  
    <script  
          src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js">  
    </script>  
    <script  
          src="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">  
    </script>  
    <script type="application/javascript" src="led-frontend.js"></script>  
  </head>  
  <body>
```

```
<div class="container">
<div class="jumbotron">
<h1>Controle de um dispositivo por meio da internet</h1>
</div>
<div class="btn-group">
<button id="on" type="button" class="btn btn-success btn-lg">Ligar</button><br>
<br>
<button id="off" type="button" class="btn btn-secondary btn-
lg">Desligar</button>
</div>
</div>
</body>
</html>
```

led.html

Em seguida, criaremos o arquivo com as funções JavaScript que serão executadas no navegador. Esse arquivo deve ser salvo na pasta public. Basicamente, será usado para mudar a aparência dos botões, destacando a ação que será realizada no LED, ou seja, ligar ou desligar e realizar a respectiva requisição AJAX.



```
$(document).ready(function() {
  $("#on").click(function(){
    $.ajax({url: "/led/on", success: function(result){
      $('#on').removeClass('btn-success').addClass('btn-secondary');
      $('#off').removeClass('btn-secondary').addClass('btn-danger');
    }});
  });
  $("#off").click(function(){
    $.ajax({url: "/led/off", success: function(result){
      $('#on').removeClass('btn-secondary').addClass('btn-success');
    }});
  });
});
```

```
$('#off').removeClass('btn-danger').addClass('btn-secondary');  
});  
});  
});
```

led-frontend.js

Prosseguindo a implementação, criaremos o servidor em Node.js, que mostrará a página principal e receberá as requisições AJAX para acender ou apagar o LED.

(Node.js)

```
var five = require("johnny-five");  
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 arduino = new five.Board();  
console.log("Arduino->Pino 13 (LED)");  
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.get('/', function (req, res) {  
fs.readFile('led.html', function(erro, dado) {  
res.writeHead(200, {'Content-Type': 'text/html'});  
res.write(dado);  
res.end();  
});
```

```
});  
arduino.on("ready", function() {  
var led = new five.Led(13);  
this.repl.inject({  
led: led  
});  
app.get('/led/on', function (req, res) {  
led.on();  
res.writeHead(200, {'Content-Type': 'text/html'});  
res.write('');  
res.end();  
});  
app.get('/led/off', function (req, res) {  
led.off();  
res.writeHead(200, {'Content-Type': 'text/html'});  
res.write('');  
res.end();  
});  
});
```

led.js

Note o uso dos métodos `on` e `off` para ligar e desligar o LED a partir da requisição recebida pelo servidor, isto é, `/led/on` e `/led/off`, respectivamente. Veja na Figura 9.5 a página HTML criada, processada a partir da requisição feita à raiz do servidor da aplicação.

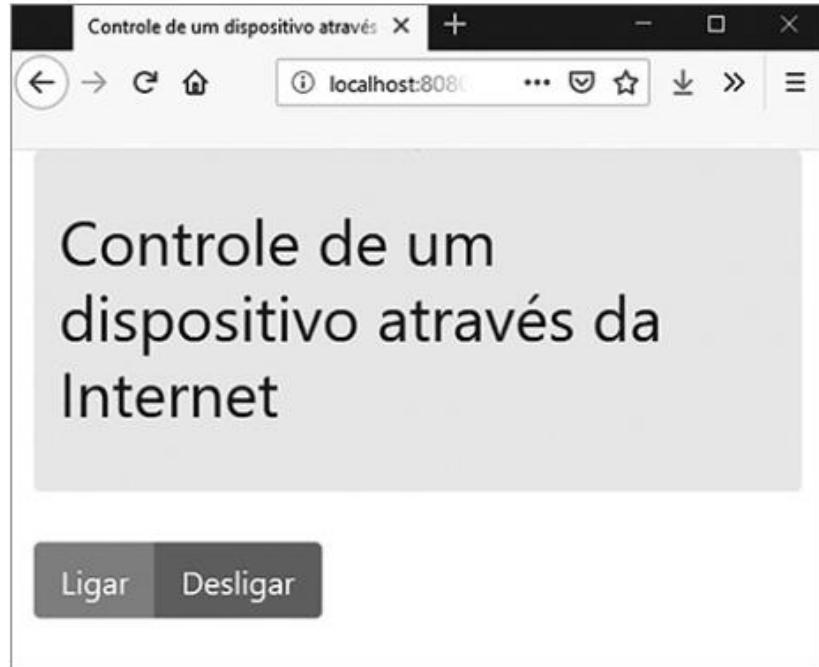


Figura 9.5 – Botões para ligar e desligar.

## 9.5 Leitura de Sensores

Além do conjunto de pinos digitais, já abordados anteriormente, o Arduino também possui pinos que funcionam como entradas analógicas (Figura 9.6). Esses pinos podem receber dados de vários tipos de sensores analógicos, como temperatura, umidade e luminosidade.

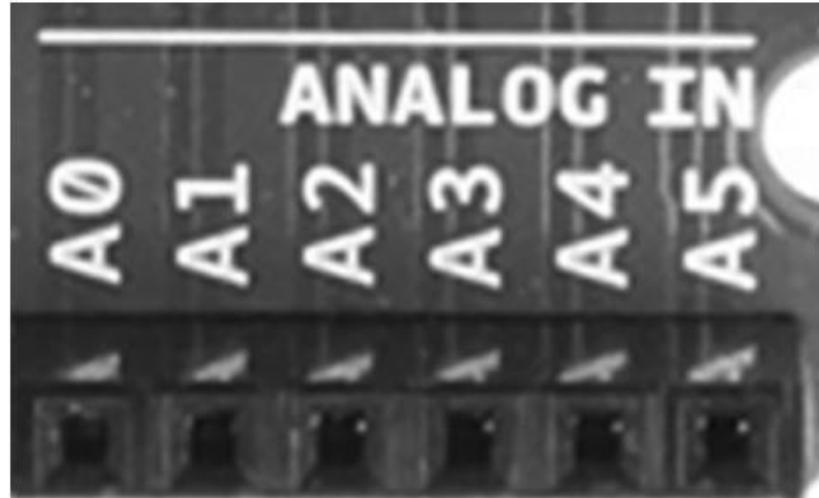


Figura 9.6 – Entradas analógicas.

O termistor, por exemplo, é um tipo de resistor que tem como característica a possibilidade de mudar o valor de sua resistência conforme a temperatura. Podemos obter esse valor por meio de um pino analógico do Arduino e convertê-lo para uma escala de temperatura, como graus Celsius, Fahrenheit ou Kelvin.

|  |   |
|--|---|
| <br><b>Relação de materiais</b> | <ul style="list-style-type: none"><li>• 1 Arduino Uno R3 (ou qualquer outro modelo).</li><li>• 1 Termistor NTC de 10 kOhms.</li><li>• 1 Resistor de 10 kOhms (marrom, preto e laranja).</li><li>• 1 Protoboard.</li><li>• Cabos de ligação.</li></ul> |
|--|---|

A Figura 9.7 apresenta a montagem do circuito eletrônico que será usado nesse projeto.

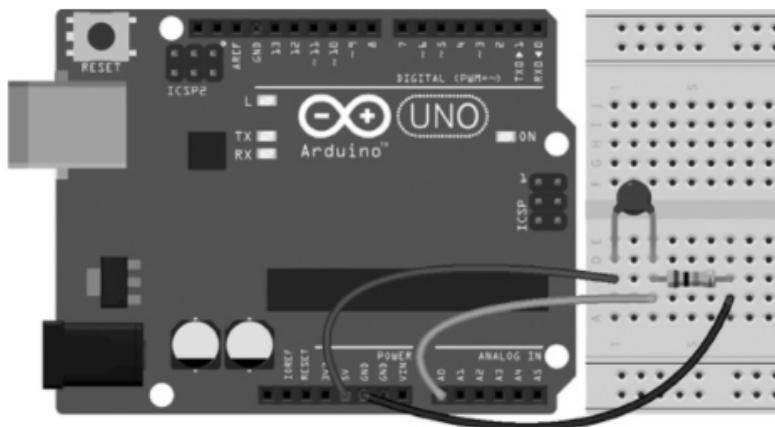


Figura 9.7 – Conexões.

Veja a seguir um programa cujo valor da tensão (Volts) presente no termistor é obtido por meio da entrada analógica 0 (zero) do Arduino, que o representa como um valor inteiro entre 0 e 1023. Esse valor é convertido para Kelvin e, depois, para graus Celsius.

{Node.js}

```
var five = require("johnny-five");

var arduino = new five.Board();
arduino.on("ready", function() {
  var A0 = new five.Sensor("A0");
```

```
A0.on("change", function() {
  var temp = getTemperatura(this.value);
  console.log(temp.toFixed(1) + "°C");
});

function getTemperatura(volts) {
  var tempK, tempC;
  tempK = Math.log(10000.0 * (1024.0 / volts - 1));
  tempK = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * tempK * tempK)) * tempK);
  tempC = tempK - 273.15;
  return tempC;
}
```

termistor.js

A fórmula e os conceitos envolvidos na conversão da tensão obtida pelo termistor para temperatura estão disponíveis em vários sites. No programa que utilizamos como referência, as informações estão disponíveis nos links <https://en.wikipedia.org/wiki/Thermistor> e <https://playground.arduino.cc/ComponentLib/Thermistor2>.

## 9.6 Exibição dos Dados de Sensores em uma Página Web

Os dados obtidos com uso de sensores podem ser exibidos em uma página web. No exemplo a seguir, criaremos um servidor Node.js que realizará essa tarefa. A montagem do circuito eletrônico é exatamente a mesma realizada no exemplo anterior.

Para começar, crie a estrutura de pastas para o projeto. Em seguida, crie a pasta public e, dentro dela, as pastas scripts e images. Dentro de pasta de imagens (images), copie a imagem de um termômetro. Instale os programas Express e Johnny-Five, conforme mostrado a seguir.



```
npm install express
npm install johnny-five
```

Em seguida, é possível implementar a página HTML que exibirá a temperatura. Observe o uso do elemento `span`.

pois ele receberá a temperatura a ser exibida.



```
<!DOCTYPE html>
<html>
<head>
<title>Temperatura</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js">
</script>
<script
src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js">
</script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
</script>
<script type="application/javascript" src="scripts/termistor-frontend.js">
</script>
</head>
<body>
<div class="container">
<div class="jumbotron">
<h1>Temperatura</h1>
</div>
<h1>

<span id="temperatura"></span>°C
</h1>
```

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

---

termistor.html

Criaremos, então, o arquivo com as funções JavaScript que serão executadas no navegador. Esse arquivo deve ser salvo na pasta scripts, que está dentro da pasta public. Ele realizará uma requisição AJAX ao servidor Node.js a cada 10 segundos, para que seja possível obter o valor da temperatura e atualizar o elemento `span` da página HTML. Esse processo evita a necessidade de carregar a página toda sempre que a temperatura for atualizada.

O método `window.setTimeout` permitirá essa atualização automática. Os parâmetros utilizados são a função e o intervalo de tempo em milissegundos (para nova execução).



---

```
function atualizar() {
    $("#temperatura").html('...');

    $.ajax({
        type: "POST",
        contentType: "text/plain",
        url: window.location + "obter/temperatura",
        success: function(dados) {
            $("#temperatura").html(dados);
            window.setTimeout(atualizar, 10000);
        }
    });
}

$(document).ready(function() {
    atualizar();
});
```

---

termistor-frontend.js

Para concluir a aplicação, temos o servidor Node.js, que será implementado usando o Express. A requisição à raiz do site mostrará a página HTML que foi criada. Além disso, a requisição '/obter/temperatura' retornará apenas a temperatura obtida pelo sensor.

{Node.js} ▾

```
var five = require("johnny-five");
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 arduino = new five.Board();
var temp = 0.0;
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.get('/', function (req, res) {
fs.readFile('termistor.html', function(erro, dado) {
res.writeHead(200, {'Content-Type': 'text/html'});
res.write(dado);
res.end();
});
});
app.post('/obter/temperatura', function(req, res){
res.writeHead(200, {'Content-Type': 'text/plain'});
res.write(temp.toFixed(1));
res.end();
});
```

```
});

arduino.on("ready", function() {
var A0 = new five.Sensor("A0");
A0.on("change", function() {
temp = getTemperatura(this.value);
});
});

function getTemperatura(volts) {
var tempK, tempC;
tempK = Math.log(10000.0 * (1024.0 / volts - 1));
tempK = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * tempK * tempK)) * tempK);
tempC = tempK - 273.15;
return tempC;
}
```

---

termistor-web.js

Veja na Figura 9.8 a página carregada no navegador, já com a temperatura obtida pelo sensor sendo atualizada em intervalos de 10 segundos.

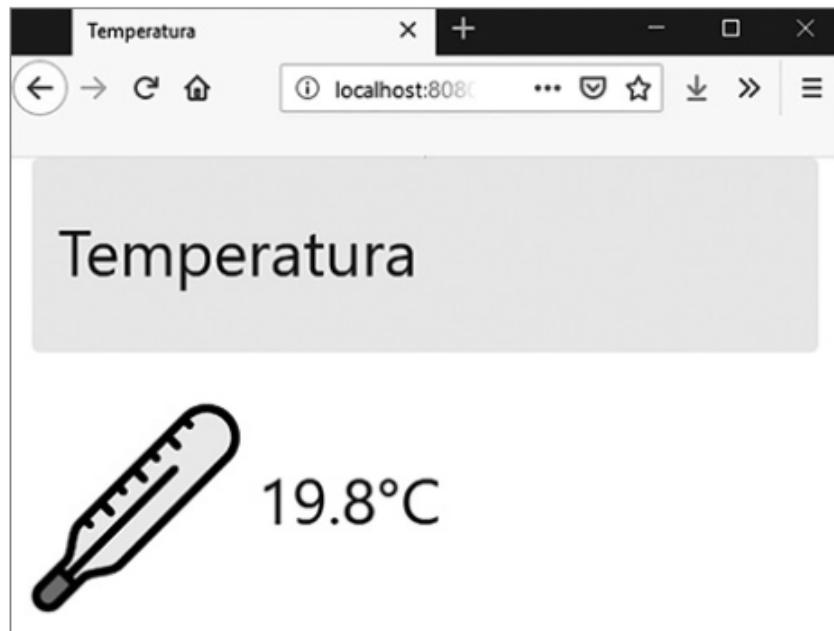


Figura 9.8 – Exibição da temperatura.

## 9.7 Armazenamento dos Dados de Sensores

Para continuar, tomaremos como base o exemplo anterior. Vamos expandir sua funcionalidade para armazenar os dados do sensor de temperatura em um banco de dados. Com o intuito de mostrar a integração do Node.js com um banco de dados relacional, adotaremos o **MariaDB** nessa implementação, que consiste em uma ramificação open source do MySQL.

|  |   |
|--|---|
| <br><b>DICA</b> | Instale o XAMPP (Apache + MariaDB + PHP + Perl). Acesse o link<br><a href="http://www.apachefriends.org/pt_br/index.html">http://www.apachefriends.org/pt_br/index.html</a> . |
|--|---|

Usaremos o phpMyAdmin (`localhost/phpmyadmin`) para criar a base de dados `iot` do exemplo. Nessa base de dados, criaremos a tabela `temperatura`, que traz os campos `valor` (float) e `data_hora` (datetime) (Figura 9.9).

The screenshot shows the phpMyAdmin interface. On the left, there's a tree view of databases: 'New', 'information\_schema', 'iot' (selected), 'New', 'mysql', 'performance\_schema', 'phpmyadmin', and 'test'. On the right, under 'iot', the 'temperatura' table is selected. The 'Estrutura da tabela' (Table Structure) tab is active, displaying the following schema:

| # | Nome      | Tipo     | Agrupamento (Collation) | Atributos | Nulo |
|---|-----------|----------|-------------------------|-----------|------|
| 1 | valor     | float    |                         |           | Não  |
| 2 | data_hora | datetime |                         |           | Não  |

Below the table structure, there are buttons for 'Add' (with value 1), 'column(s)', 'after data\_hora', and 'Executar'.

Figura 9.9 – Tabela temperatura.

Na pasta da aplicação, execute a instalação do módulo mariadb, conforme o código a seguir.



```
npm install mariadb
```

Agora, basta alterar o arquivo `termistor-web.js` para realizar a conexão com o servidor de banco de dados. Essa ação também é indicada para inserir o registro na tabela temperatura sempre que houver a requisição AJAX para atualizar a temperatura na página HTML. Dessa forma, a primeira etapa consiste em criar um pool com os parâmetros necessários para a conexão com o banco de dados.



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

var pool = mariadb.createPool({
  host: 'localhost',
  database: 'iot',
  user: 'usuario',
  password: 'senha'
});
```

Em seguida, criaremos uma função que realizará a conexão ao MariaDB e inserirá um registro na tabela temperatura, como ilustra o trecho de código-fonte a seguir.



```
-----  
function inserir(valor) {  
  pool.getConnection()  
    .then(con => {  
      var agora = new Date().toISOString().slice(0, 19).replace('T', ' ');  
      con.query("INSERT INTO temperatura VALUES (?, ?)", [valor, agora])  
        .then((res) => {  
          console.log(res);  
          con.end();  
        })  
        .catch(error => {  
          console.log("Erro comando: " + error);  
          con.end();  
        });  
    }).catch(error => {  
      console.log("Erro conexão:" + error);  
    });  
}  
-----
```

A alteração que será realizada na rotina responde à requisição AJAX, em que haverá inserção da função inserir.

{Node.js}

```
-----  
app.post('/obter/temperatura', function(req, res) {  
  res.writeHead(200, {'Content-Type': 'text/plain'});  
  res.write(temp.toFixed(1));  
  res.end();  
  inserir(temp.toFixed(1));  
});  
-----
```

Para concluir a implementação, editaremos a página que contém as funções JavaScript executadas no navegador e aumentaremos o intervalo entre as atualizações de 10 segundos (10.000 milissegundos) para 60 segundos (60.000 milissegundos), isto é, 1 minuto.

{JS}

```
window.setTimeout(atualizar, 10000);
```

Veja na Figura 9.10 um exemplo dos dados enviados pela aplicação Node.js, devidamente armazenados no servidor de banco de dados, visualizados a partir do phpMyAdmin.

| A mostrar registos de 0 - 7 (8 total, A consulta demorou 0,0000 segundos.)  |                     |
|---|---------------------|
| SELECT * FROM `temperatura`   |                     |
| <input type="checkbox"/> Mostrar tudo   Número de registos: 25 <input type="button" value="▼"/> <input type="text"/> Filtrar registos: <input type="button" value="Pesquisar esta tabela"/> |                     |
| <b>+ Opções</b>   |                     |
| valor   | data_hora           |
| 17  | 2019-05-24 10:13:27 |
| 16.9  | 2019-05-24 10:14:27 |
| 16.9  | 2019-05-24 10:15:27 |
| 16.9  | 2019-05-24 10:16:27 |
| 16.9  | 2019-05-24 10:17:27 |
| 17  | 2019-05-24 10:18:27 |
| 17  | 2019-05-24 10:19:27 |
| 17.1  | 2019-05-24 10:20:27 |

Figura 9.10 – Dados armazenados no MariaDB.

## 9.8 Consulta ao Banco de Dados

Para explicar como realizar uma consulta aos dados armazenados no banco de dados, adicionaremos uma nova página à aplicação. Essa página será usada para exibir os dados que estão armazenados na tabela temperatura implementada no MariaDB.

Para começar, acrescente um link à página termistor.html, conforme ilustra o código-fonte a seguir.

&lt;/&gt;

```
<a href="relacao/temperatura">Histórico...</a>
```

Em seguida, na pasta da aplicação, crie a pasta `views` e utilize o `npm` para instalar o Framework Embedded JavaScript templating (EJS), que já utilizamos anteriormente.

&gt;-

```
md views  
npm install ejs
```

Dentro da pasta `views`, salvaremos a página que será criada para mostrar data, hora e temperatura armazenadas na respectiva tabela do banco de dados. Essa página será chamada de `historico.ejs`. Observe a seguir.



```
<!DOCTYPE html>  
  
<html>  
  <head>  
    <title>Histórico</title>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <link rel="stylesheet"  
      href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">  
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js">  
    </script>  
    <script  
      src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/popper.min.js">  
    </script>  
    <script  
      src="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">  
    </script>  
    <script type="application/javascript" src="scripts/termistor-frontend.js">  
    </script>  
  </head>  
  <body>  
    <div class="container">  
      <div class="jumbotron">  
        <h1>Histórico de Temperatura</h1>  
      </div>  
      <table class="table table-hover">
```

```
<tr class="table-primary">
<td>Data e Hora</td>
<td>Temperatura</td>
</tr>
<% dados.forEach(function(item) { %>
<tr>
<td><%= item.data_hora %></td>
<td align="right"><%= item.valor.toFixed(1) %>°C</td>
</tr>
<% }) %>
</table>
</div>
</body>
</html>
```

historico.ejs

Conforme destaca o trecho de programa a seguir, observe o uso do EJS que receberá o conjunto de dados, no formato JSON. Em seguida, esses dados serão enviados pela aplicação Node.js para serem renderizados no cliente. Lembre-se de que os elementos do EJS são delimitados por `<% %>` e apresentam, como vantagem, o fato de serem especificados em JavaScript, o que facilita muito a sua utilização.

</>

```
<% dados.forEach(function(item) { %>
<tr>
<td><%= item.data_hora %></td>
<td align="right"><%= item.valor.toFixed(1) %>°C</td>
</tr>
<% }) %>
```

Na aplicação Node.js implementada no arquivo `termistor-web.js`, definimos que o parser será realizado usando o formato JSON e adicionamos o framework EJS como 'view engine'.

{Node.js}

```
app.use(bodyParser.json());
app.set('view engine', 'ejs');
```

Por fim, realizamos o tratamento da requisição. Veja a seguir.

{Node.js}

```
app.get('/relacao/temperatura', function (req, res) {
  pool.getConnection()
    .then(con => {
      con.query("SELECT DATE_FORMAT(data_hora, '%d/%m/%Y %H:%i') AS data_hora, valor
      FROM temperatura ORDER BY data_hora DESC")
        .then(registros => {
          res.render('historico.ejs', { dados: registros });
          con.end();
        })
        .catch(erro => {
          console.log("Erro comando: " + erro);
          con.end();
        });
    })
    .catch(erro => {
      console.log("Erro conexão:" + erro);
    });
});
```

É importante observar que realizamos uma consulta ao servidor de banco de dados. O resultado obtido está em formato JSON. Esse resultado é aplicado ao renderizador de páginas do EJS. Veja na Figura 9.11 um exemplo da página exibindo os dados armazenados no servidor de banco de dados.

The screenshot shows a web browser window with the title bar 'Histórico'. The address bar displays 'localhost:80'. The main content area has a heading 'Histórico de Temperatura'. Below it is a table with two columns: 'Data e Hora' and 'Temperatura'. The data rows are as follows:

| Data e Hora      | Temperatura |
|------------------|-------------|
| 24/05/2019 16:21 | 19.1°C      |
| 24/05/2019 13:33 | 19.9°C      |
| 24/05/2019 13:32 | 20.1°C      |
| 24/05/2019 13:31 | 20.1°C      |
| 24/05/2019 13:30 | 19.9°C      |

Figura 9.11 – Exibição dos dados.

## 9.9 Chart.js

Com base no exemplo anterior, podemos acrescentar à página de histórico um gráfico que exibirá a temperatura referente a cada data. Utilizaremos a **Chart.js**, que é uma versátil biblioteca de gráficos implementada em HTML5 e JavaScript.

Na página "historico.ejs", salva na pasta "views" da aplicação, acrescente a chamada ao arquivo em JavaScript que implementa a biblioteca. Veja a seguir o exemplo de código-fonte.



```
<script      src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0/dist/Chart.min.js">
</script>
```

Em seguida, no corpo da página HTML, adicione o elemento `canvas`, que será responsável pela exibição do gráfico, além da rotina JavaScript que configurará o gráfico.

</>

```
<canvas id="grafico"></canvas>
<script type='text/javascript'>
var rotulo = [];
<% dados.forEach(function(item) { %>
rotulo.push('<%=item.data_hora %>');
<% }) %>
var dado = [];
<% dados.forEach(function(item) { %>
dado.push(<%= item.valor.toFixed(1) %>);
<% }) %>
var objeto = grafico.getContext('2d');
var graf = new Chart(objeto, {
type: 'line',
data: {
labels: rotulo,
datasets: [
label: 'Temperatura (°C)',
data: dado,
borderWidth: 1
]
},
options: {
scales: {
yAxes: [
ticks: {
beginAtZero: true
}
]
}
}
})
```

```
    }
}

} ;


```

</script>

Ainda sobre o trecho de código-fonte anterior, observe que usaremos os mesmos dados (data, hora e temperatura) utilizados para montar a tabela histórico que já é exibida na página. Esses dados são carregados nos vetores `rotulo` e `dado` para serem, posteriormente, usados no gráfico.

Em seguida, no objeto `graf`, instanciado a partir da classe `Chart`, definimos o tipo de gráfico, rótulos e valores de cada eixo.

Veja na Figura 9.12 um exemplo de como o gráfico será exibido na página de histórico de temperatura.



Figura 9.12 – Gráfico criado com o Chart.js.

Veja a seguir como fica o código-fonte completo da página `histórico.ejs`, logo após a inserção do gráfico.



```
<!DOCTYPE html>

<html>
  <head>
    <title>Histórico</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js">
    </script>
    <script
      src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/popper.min.js">
    </script>
    <script
      src="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0/dist/Chart.min.js">
    </script>
  </head>
  <body>
    <div class="container">
      <div class="jumbotron">
        <h1>Histórico de Temperatura</h1>
      </div>
      <canvas id="grafico"></canvas>
      <script type='text/javascript'>
        var rotulo = [];
        <% dados.forEach(function(item) { %>
```

```
rotulo.push('<%=item.data_hora %>');
<% }) %>
var dado = [];
<% dados.forEach(function(item) { %>
dado.push(<%= item.valor.toFixed(1) %>);
<% }) %>
var objeto = grafico.getContext('2d');
var graf = new Chart(objeto, {
type: 'line',
data: {
labels: rotulo,
datasets: [{

label: 'Temperatura (°C)',
data: dado,
borderWidth: 1
}]
},
options: {
scales: {
yAxes: [{

ticks: {

beginAtZero: true
}
}
]
}
}
});
</script>


||
||
||


```

```
<td>Data e Hora</td>
<td>Temperatura</td>
</tr>
<% dados.forEach(function(item) { %>
<tr>
<td><%= item.data_hora %></td>
<td align="right"><%= item.valor.toFixed(1) %>°C</td>
</tr>
<% }) %>
</table>
</div>
</body>
</html>
```

historico.ejs

## 9.10 ThingSpeak

A ThingSpeak é uma plataforma muito popular para a IoT. Essa plataforma recebe dados de sensores, realiza o armazenamento em nuvem, provê gráficos e análises estatísticas. Também apresenta um mecanismo que permite reagir, ou seja, realizar uma ação a partir de determinadas situações que podem ser identificadas de acordo com a análise dos dados.



DICA

Acesse o site da plataforma **ThingSpeak** <<https://thingspeak.com/>> e crie uma conta de usuário gratuita.

Após criar uma conta de usuário e acessar a plataforma, é preciso abrir um canal para realizar o armazenamento dos dados. Para fazer isso, selecione a opção "Canais" no menu; depois selecione "Meus Canais" e clique no botão "New Channel" (Figura 9.13).

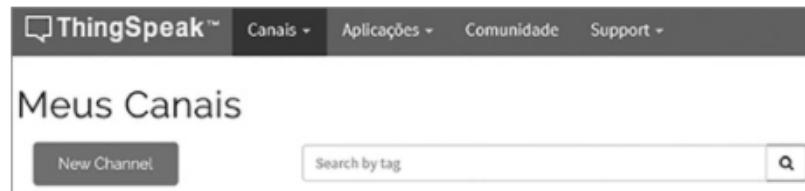


Figura 9.13 – Criação de um canal.

O canal pode ser chamado de "iot" e deve conter a temperatura no Campo 1 (Figura 9.14). Após preencher essas informações, clique no botão "Save Channel", localizado no rodapé da página.

|                  |   |
|------------------|---|
| <b>Nome</b>      | iot   |
| <b>Descrição</b> | Dados de temperatura enviados pelo termistor    |
| <b>Campo 1</b>   | temperatura <input checked="" type="checkbox"/> |
| <b>Campo 2</b>   | <input type="checkbox"/>                        |
| <b>Campo 3</b>   | <input type="checkbox"/>                        |

Figura 9.14 – Nome e estrutura do canal.

Ao concluir o processo de criação do canal, será mostrada a página padrão do canal (Figura 9.15).

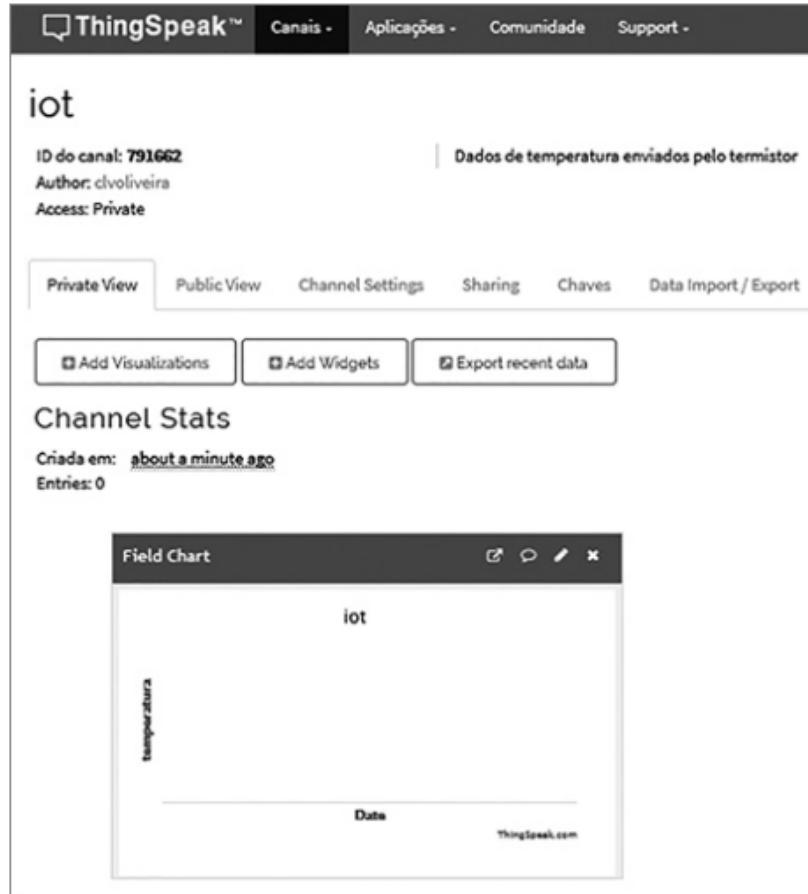


Figura 9.15 – Exibição do canal.

## 9.11 Envio de Dados para a Nuvem

Após criar o canal na plataforma ThingSpeak, é possível criar a aplicação Node.js. Essa aplicação deve obter os dados do sensor e, em seguida, enviá-los para o canal que foi criado.

Na pasta da aplicação, use o npm para instalar as bibliotecas **johnny-five**, já usada em aplicações anteriores. Ela possibilita a interação com o Arduino e com a **thingspeakclient**, que permite enviar e recuperar dados de um canal armazenado na ThingSpeak.



```
npm install johnny-five  
npm install thingspeakclient
```

A relação de materiais e a montagem do circuito eletrônico são apresentadas a seguir. Ou seja, obteremos os dados de temperatura de um termistor conectado à entrada analógica do Arduino.

|  |   |
|--|---|
| <br><b>Relação de materiais</b> | <ul style="list-style-type: none"><li>• 1 Arduino Uno R3 (ou qualquer outro modelo).</li><li>• 1 Termistor NTC de 10k Ohms.</li><li>• 1 Resistor de 10k Ohms (marrom, preto e laranja).</li><li>• 1 Protoboard.</li><li>• Cabos de ligação.</li></ul> |
|--|---|

Você vê na Figura 9.16 a montagem do circuito eletrônico que será usado nesse projeto.

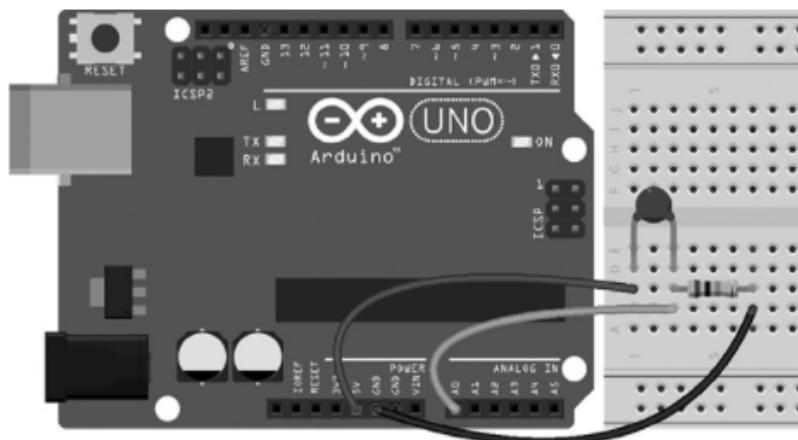


Figura 9.16 – Montagem do circuito eletrônico.

O programa deve enviar a temperatura obtida pelo sensor em um determinado intervalo de tempo. Inicialmente, definimos os valores para as variáveis-chave e canal, que devem ser obtidas na aba Chaves do canal (Figura 9.17).

The screenshot shows the 'Chaves' (Keys) section of the ThingSpeak interface for a channel named 'iot'. At the top, there are tabs for 'Private View', 'Public View', 'Channel Settings', 'Sharing', 'Chaves' (which is selected), and 'Data Import / Export'. Below the tabs, the channel information is displayed: ID do canal: 791662, Author: cdivoliveira, and Access: Private. A note states 'Dados de temperatura enviados pelo termistor'. The 'Chave de Escrita' (Write Key) field contains 'VNQAJ67H5TMAPJRX'. A button labeled 'Gerar nova chave de escrita' (Generate new write key) is present. The 'Read API Keys' section shows a 'Chave' (Key) field containing '68PS6883ALGUC9IF' and a 'Nota' (Note) field which is empty. Buttons for 'Salvar' (Save) and 'Apagar chave' (Delete key) are at the bottom. A 'Gerar nova chave de leitura' (Generate new read key) button is also visible.

Figura 9.17 – Chaves.

Também é preciso definir o intervalo entre as leituras que, no exemplo, é de 10 minutos. Em seguida, criaremos um objeto para o sensor. Ao criar esse objeto, usaremos como parâmetros o pino do Arduino ao qual o sensor está conectado (A0) e o intervalo entre as leituras.

O intervalo define a frequência que o evento data será gerado. É nesse evento que realizaremos a leitura do sensor e enviaremos para a ThingSpeak.

{Node.js}

```
var five = require("johnny-five");
var ThingSpeakClient = require('thingspeakclient');
```

```
var arduino = new five.Board();
var cliente = new ThingSpeakClient();
var chave = 'INSERIR_SUA_CHAVE';
var canal = 791662;
var intervalo = 600000; // = 10 minutos
arduino.on("ready", function() {
  var A0 = new five.Sensor({
    pin: "A0",
    freq: intervalo});
  A0.on("data", function() {
    var temp = getTemperatura(this.value);
    cliente.attachChannel(canal, { writeKey:chave}, function(erro, resp) {
      if (!erro) {
        console.log('Conectado ao canal!');
      }
      else {
        console.log('ERRO: ' + erro);
      }
    });
    cliente.updateChannel(canal, {field1: temp.toFixed(1)}, function(erro, resp) {
      if (!erro && resp > 0) {
        console.log('Enviado: ' + temp.toFixed(1) + '°C. Resposta: ' + resp);
        console.log();
      }
      else {
        console.log('ERRO: ' + erro);
      }
    });
  });
});
```

```

function getTemperatura(volts) {
var tempK, tempC;
tempK = Math.log(10000.0 * (1024.0 / volts - 1));
tempK = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * tempK * tempK)) * tempK);
tempC = tempK - 273.15;
return tempC;
}

```

termistor-thingspeak.js

Vamos prosseguir com a análise do código-fonte. Após realizar a leitura da temperatura, usaremos o método `attachChannel` para se conectar ao canal. Nesse método, informamos o número de identificação (`id`) do canal e a chave para escrita. Na sequência, o método `updateChannel` insere o dado no canal. Observe na Figura 9.18 que podemos usar a visão do canal, disponível na plataforma ThingSpeak, para visualizar um gráfico desenvolvido com dados armazenados.

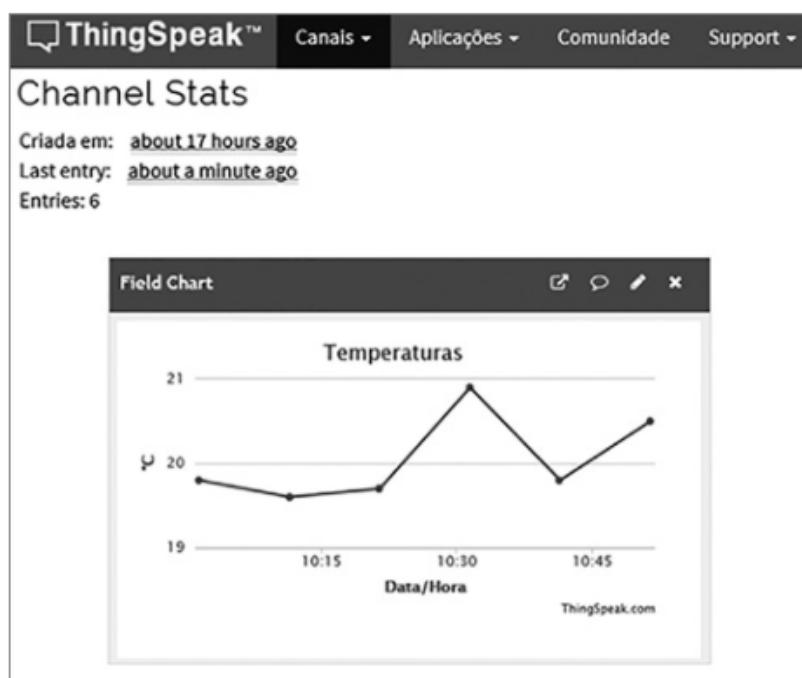


Figura 9.18 – Visualização dos dados do canal.

## 9.12 Visualização dos Dados

A informação armazenada em um canal da plataforma ThingSpeak pode ser acessada de diversas maneiras. Como vimos anteriormente, é possível usar a própria plataforma para isso. Outra possibilidade é customizar a exibição e inserir os gráficos dos dados do canal em uma página web.

A customização é realizada por meio da opção "Field Chart Options" (Figura 9.19).

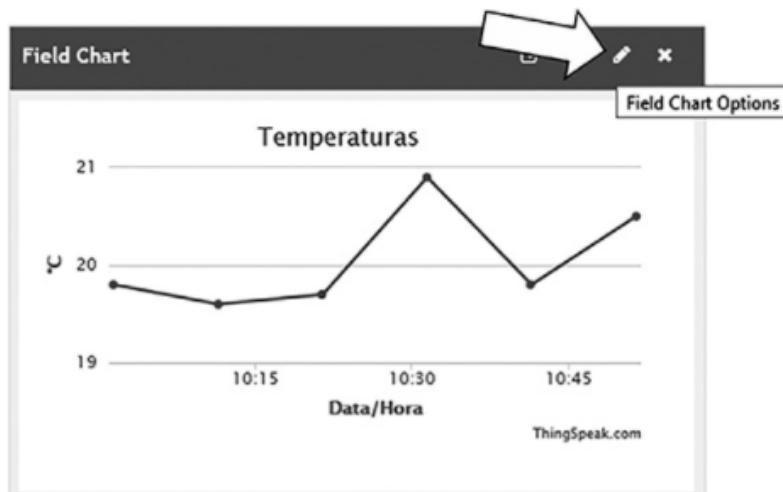


Figura 9.19 – Customização do gráfico.

No quadro apresentado na Figura 9.20, é possível personalizar várias opções de exibição do gráfico. No exemplo, definiremos o título do gráfico e os rótulos para os eixos "x" e "y". Após realizar as alterações, clique no botão "Save".

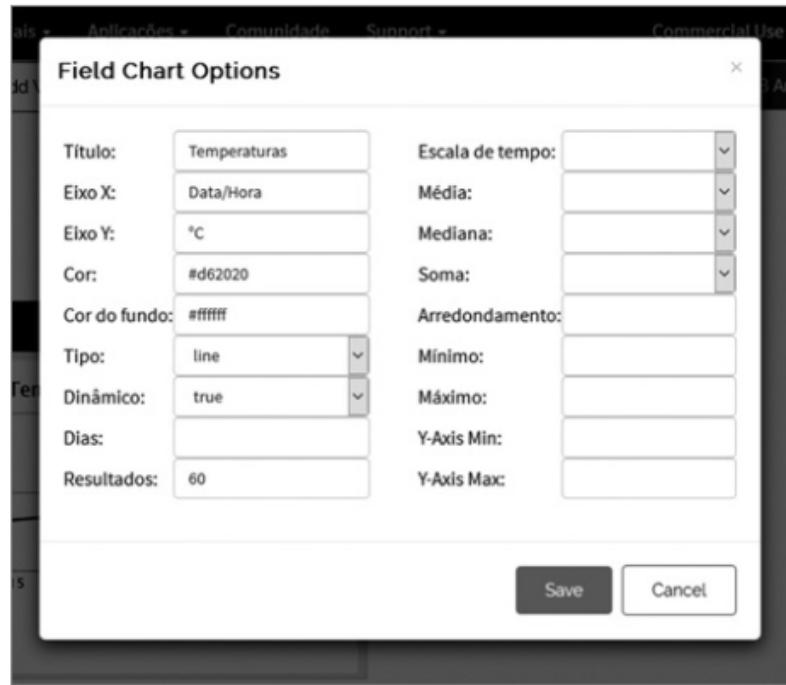


Figura 9.20 – Opções para exibição do gráfico.

Na opção "Field Chart iFrame" (Figura 9.21), podemos obter o código-fonte necessário para inserir o gráfico em uma página HTML.



Figura 9.21 – Opções para exibição do gráfico.

A seguir, veja um exemplo do iFrame gerado automaticamente por essa opção.



```
<iframe width="450" height="260" style="border: 1px solid #cccccc;"  
src="https://thingspeak.com/channels/791662/charts/1?  
bgcolor=%23ffffffff&color=%23d62020&dynamic=true&results=60&title=Temperaturas&  
type=line&xaxis=Data%2FHora&yaxis=%C2%B0C">  
</iframe>
```

No código-fonte a seguir, você observa a criação da página HTML completa que contém, entre outros elementos, o iFrame responsável pela exibição do gráfico.

</>

```
<!DOCTYPE html>
<html>
<head>
<title>Gráfico de Temperatura</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js">
</script>
<script
src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js">
</script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
</script>
<script type="application/javascript" src="scripts/termistor-frontend.js">
</script>
</head>
<body>
<div class="container">
<div class="jumbotron">
<h1>Gráfico de Temperatura</h1>
</div>
&ampnbsp
<iframe width="450" height="260" style="border: 1px solid #cccccc;">
<script>
src="https://thingspeak.com/channels/791662/charts/1?</script>

```

```
bgcolor=%23ffffff&color=%23d62020&dynamic=true&results=60&title=Temperaturas&type=line&xaxis=Data%2FHora&yaxis=%C2%B0C"></iframe>
</div>
</body>
</html>
```

temperatura-thingspeak.html

A Figura 9.22 mostra como a página criada é vista no navegador.



Figura 9.22 – Página contendo o gráfico.

### VAMOS PRATICAR!

1. A partir do diagrama de montagem mostrado na Figura 9.23 e da respectiva relação de materiais, crie uma aplicação Node.js que mantenha o LED aceso enquanto a chave tátil estiver pressionada; caso contrário, o LED deverá permanecer apagado.



Figura 9.23 – Diagrama de montagem.

|  |  |
|--|--|
| <br><b>Relação de materiais</b> | <ul style="list-style-type: none"><li>• 1 Arduino Uno R3 (ou qualquer outro modelo).</li><li>• 1 Resistor de 220 Ohms (vermelho, vermelho e marrom) ou 330 Ohms (laranja, laranja e marrom).</li><li>• 1 Resistor de 10 kOhms (marron, preto e laranja).</li><li>• 1 Led (qualquer cor).</li><li>• 1 Chave táctil.</li><li>• 1 Protoboard.</li><li>• Cabos de ligação.</li></ul> |
|--|--|

2. Considerando ainda a Figura 9.23, crie uma aplicação Node.js que ative o LED quando o usuário pressionar a chave tátil e o mantenha aceso até que ela seja pressionada novamente.
3. A partir do diagrama apresentado na Figura 9.23, utilize o Node.js para implementar um servidor que tenha uma página web capaz de mostrar o estado da chave tátil, ou seja, se ela está pressionada ou não.
4. Adotando como base a Figura 9.23, elabore uma aplicação Node.js que, sempre que a chave tátil for pressionada, insira em uma tabela no MariaDB um registro que contenha a data e a hora do em que foi pressionada.
5. Crie um servidor Node.js que implemente uma página com todos os registros armazenados pela rotina criada no exercício anterior.

# Desenvolvimento para Dispositivos Móveis

## INICIANDO OS ESTUDOS

---

Com o aumento constante da venda de dispositivos móveis, cresce a demanda por novos softwares para esse tipo de plataforma. Neste capítulo, trabalharemos com o Ionic, framework de desenvolvimento para dispositivos móveis. Essa plataforma oferece a abordagem híbrida, uma das mais modernas abordagens de desenvolvimento mobile. Aplicativos híbridos usam tecnologias comuns para web – como HTML, CSS e JavaScript –, e podem ser usados por diversas plataformas diferentes.

Atualmente, existem muitas opções de ferramentas para desenvolvimento de aplicativos para dispositivos móveis (*mobile*). Por isso, são muitos os questionamentos sobre qual tecnologia adotar em projetos. Algumas plataformas mais recentes apostam na questão multiplataforma, em que o aplicativo pode rodar tanto na plataforma Android quanto na plataforma iOS. E, com o mercado de dispositivos móveis em expansão, muitas soluções surgem todos os dias. A demanda por inovação é constante e crescente.

Uma tendência atual é o desenvolvimento chamado híbrido, em que não é necessário criar o aplicativo usando a linguagem nativa da plataforma (por exemplo, o Swift no iOS ou Java no Android). No desenvolvimento híbrido, é possível usar recursos como HTML, CSS e JavaScript. Essa abordagem permite desenvolver projetos mais flexíveis, assim como oferece liberdade para criar interfaces diversificadas que agregam novos recursos tecnológicos.

Nesse contexto, entre os frameworks gratuitos e open-source mais utilizados, destacamos o Ionic, criado para ser utilizado facilmente por desenvolvedores web. Esse framework utiliza prioritariamente a linguagem JavaScript, além de HTML e CSS. O Ionic está na versão 4.0 e será o foco deste capítulo.

## 10.1 Framework Ionic

O Ionic é uma plataforma desenvolvida com base em outro framework de código aberto muito famoso chamado Cordova (mantido pela Fundação Apache). O Cordova foi desenvolvido com a tecnologia PhoneGap, da Adobe.



DICA

Para conhecer mais sobre a plataforma Cordova, acesse o link <https://cordova.apache.org/>.

A base do Ionic é a linguagem JavaScript, que será utilizada na construção da camada lógica de desenvolvimento e interação. Além disso, o HTML será empregado na estruturação de conteúdo, e o CSS realizará a formatação de interface e estilos. Em relação às tecnologias embutidas no framework, utilizaremos a AngularJS, plataforma do Google, para o desenvolvimento front-end em aplicações para web.



DICA

Baixe e instale o framework Ionic no link <https://ionicframework.com/>. Para a instalação, siga as instruções disponíveis em: <https://ionicframework.com/docs/installation/cli>.

A instalação do framework Ionic inclui um console (Ionic CLI), que deve ser utilizado para iniciar o servidor de aplicação e montar toda a hierarquia de projeto.

Para começar o trabalho, abra um prompt de comando no Ionic. No exemplo, usaremos o sistema operacional Windows. Em seguida, digite o comando **ionic start**. O CLI pedirá para você digitar um nome para o aplicativo e/ou projeto. Nossa projeto terá o nome de MeuApp. A Figura 10.1 mostra o terminal.

```
C:\>ionic start  
Every great app needs a name!  
Please enter the full name of your app. You can change this at any time. To  
bypass this prompt next time, supply name, the first argument to ionic start.  
? Project name: MeuApp
```

Figura 10.1 – Console do Ionic.

Para começar o desenvolvimento do projeto, temos que definir um template básico para o layout inicial. Os três templates básicos são:

- **Tabs:** layout com base em tabs ou abas.
- **Sidemenu:** layout que inclui uma estrutura de menu lateral.
- **Blank:** cria um projeto com uma página em branco.

O ponto inicial é desenvolver um aplicativo utilizando o modelo "blank". Para isso, devemos digitar no console o comando **ionic start**, seguido pelo nome do projeto. Veja o código-fonte ilustrado a seguir.



```
-----  
ionic start MeuApp blank  
-----
```

Nesse momento, será "montada" toda a estrutura da aplicação. O framework precisa ter acesso à internet para baixar alguns arquivos, o que pode demorar alguns minutos.

Uma dica é criar um diretório exclusivo para projetos Ionic e executar o comando anterior que está salvo no diretório. Por exemplo, crie um diretório chamado "Ionic" dentro do diretório raiz (C:\). Quando executar a montagem de um novo projeto, será criado um diretório com o nome do projeto (C:\Ionic\MeuApp).

## 10.2 Nosso Primeiro Aplicativo

A estrutura do projeto criado pelo Ionic pode parecer muito complexa a princípio. São criados diversos arquivos em diferentes pastas, todas com funções específicas. Algumas pastas são arquivos de configurações, outras concentram os arquivos de estilos e formatação, e há ainda aquelas com os códigos-fonte da aplicação.

Não abordaremos todas as pastas ou arquivos. Em termos práticos, a principal pasta que exploraremos é a pasta **src**, em que estão os principais arquivos, estruturas de programação e formatação da aplicação.

Antes de começar a programação do nosso primeiro aplicativo, aprenderemos como executar o projeto criado e, ao mesmo tempo, testar se está tudo certo até o momento.

No terminal de comando, entre no diretório do projeto que, no exemplo, seria C:\Ionic\MeuApp. Uma vez no

diretório, basta executar o comando a seguir.



```
ionic serve
```

O comando faz com que a aplicação seja executada no servidor local Ionic e a interface do aplicativo seja exibida no navegador padrão, no endereço localhost:8100/home. A Figura 10.2 mostra o resultado da aplicação já carregada no navegador.

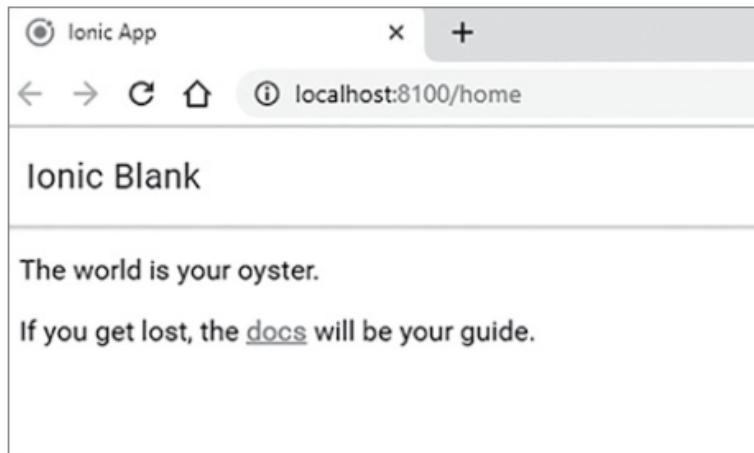


Figura 10.2 – Execução do projeto criado.

Agora que todo o processo de instalação, criação e execução do projeto está funcionando adequadamente, realizaremos a primeira alteração no código-fonte base de projeto.

Dentro do diretório do projeto, vá em `src > app > home > home.page.html`. Alteraremos as mensagens e o título do aplicativo. Nesse arquivo, altere a tag `<ion-title>` para mudar o título e a tag `<ion-content>` para alterar a mensagem do corpo da página.



```
<ion-header>  
<ion-toolbar>  
<ion-title>  
Meu primeiro aplicativo
```

```
</ion-title>
</ion-toolbar>
</ion-header>
<ion-content>
<div class="ion-padding">
<p>Olá mundo!</p>
</div>
</ion-content>
```

home.page.html

A Figura 10.3 mostra o resultado da alteração já no navegador. Na tag `<ion-title>` colocamos o novo título, que aparece no topo da página. Na tag `<ion-content>` fica a maior parte de nossos componentes da interface com o usuário. Nesse caso, usamos uma tag padrão HTML para inserir um parágrafo contendo a frase "Olá mundo".

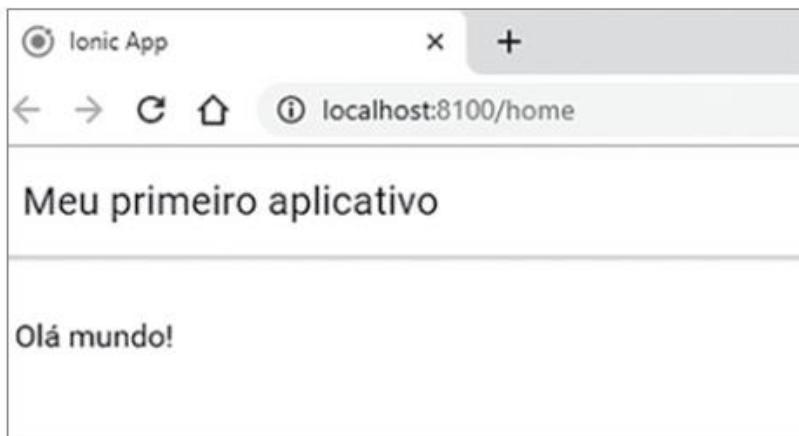


Figura 10.3 – Execução do aplicativo após as alterações.

### 10.3 Desenvolvendo uma Calculadora Básica

No projeto seguinte, desenvolveremos o aplicativo "calculadora simples". O aplicativo permitirá ao usuário digitar dois valores numéricos e, em seguida, realizar a seleção da operação desejada (adição, subtração, divisão ou multiplicação).

Para começar a criação do projeto **CalculadoraSimples**, é preciso seguir os passos da criação do projeto anterior. Para a interface, criaremos dois campos de texto para a entrada dos valores numéricos, além de um botão para cada uma das operações.

Para os botões, usaremos as tags `<ion-button>`. Para melhorar a legibilidade da interface, usaremos o atributo `size` (tamanho) com o valor `large` (grande). Também utilizaremos uma referência da variável "resultado" (`{{resultado}}`), que será abordada mais adiante.

Note que a tag `<ion-content>` apresenta o atributo `class` com o valor `ion-padding`, definindo uma distância entre o conteúdo de um elemento e suas bordas. Veja a seguir o código-fonte responsável pela interface do aplicativo.



```
<ion-header>
<ion-toolbar>
<ion-title>
Calculadora Simples
</ion-title>
</ion-toolbar>
</ion-header>
<ion-content class="ion-padding">
<ion-list>
<ion-item>
<ion-label color="primary"> Digite um número:</ion-label>
<ion-input [(ngModel)]="num1" name="num1"></ion-input>
</ion-item>
<ion-item>
<ion-label color="primary"> Digite um número:</ion-label>
<ion-input [(ngModel)]="num2" name="num2"></ion-input>
</ion-item>
</ion-list>
<ion-button size="large" full (click)="somar()"> + </ion-button>
<ion-button size="large" full (click)="subtrair()"> - </ion-button>
<ion-button size="large" full (click)="dividir()"> / </ion-button>
<ion-button size="large" full (click)="multiplicar()"> X</ion-button>
<h1> {{resultado}} </h1>
```

```
</ion-content>
```

home.page.html

A Figura 10.4 mostra a interface em formato de tela similar ao que estaria disponível em um dispositivo móvel.



Figura 10.4 – Interface desenvolvida.



**DICA** Quando desejar visualizar a execução de um projeto na resolução de um dispositivo móvel, utilize as ferramentas de desenvolvedor do seu navegador favorito (por exemplo, Google Chrome). Acesse Visualização > Desenvolvedor > Ferramentas para desenvolvedores.

Após o desenvolvimento da interface do aplicativo, passaremos para a programação. No arquivo `src > app > home > home.page.ts` criaremos três variáveis. São elas:

- **num1**: receberá o valor do primeiro campo;
- **num2**: receberá o valor do segundo campo;
- **resultado**: receberá o resultado do cálculo efetuado.

Note que, no arquivo anterior (`home.page.html`), utilizamos `[ (ngModel) ]`, que corresponde a um atributo de linguagem utilizado em formulários para vincular e atribuir valores e objetos ao projeto. Por exemplo, a linha de código a seguir cria um campo de texto vinculado à variável **num1**. Essa referência direta facilita a passagem de dados entre as camadas de apresentação e lógica de nosso aplicativo.



```
<ion-input [(ngModel)]="num1" name="num1"></ion-input>
```

Também é importante ressaltar que, no código-fonte existente no arquivo `home.page.html`, criamos quatro botões. Cada botão chama uma função diferente. Cada uma dessas funções utiliza as variáveis `num1` e `num2` para obter os valores inseridos nos campos e realizar uma operação solicitada. Assim, o valor é descrito na variável `resultado`, apresentada na interface pela referência `{{ resultado }}`. Veja a seguir o código-fonte que deve ser incluído na classe principal do arquivo `home.page.ts`.



```
export class HomePage {  
  
  num1 : string;  
  num2 : string;  
  resultado : string;  
  constructor() {}  
  somar(){  
    var n1 = parseFloat(this.num1);  
    var n2 = parseFloat(this.num2);  
    var soma = n1 + n2;  
    this.resultado = soma.toString();  
  }  
  subtrair(){  
    var n1 = parseFloat(this.num1);  
    var n2 = parseFloat(this.num2);  
    var subtracao = n1 - n2;  
    this.resultado = subtracao.toString();  
  }  
  dividir(){  
    var n1 = parseFloat(this.num1);  
    var n2 = parseFloat(this.num2);  
  }  
}
```

```
if(n2 != 0){  
    var divisao = n1 / n2;  
    this.resultado = divisao.toString();  
} else{  
    this.resultado = "Não foi possível realizar a divisão!";  
}  
}  
  
multiplicar(){  
    var n1 = parseFloat(this.num1);  
    var n2 = parseFloat(this.num2);  
    var multiplicacao = n1 * n2;  
    this.resultado = multiplicacao.toString();  
}
```

---

[home.page.ts](#)

Observe que todas as funções são bem similares. A única diferença está na subtração, que possui uma estrutura condicional if-else para verificar se o divisor inserido é diferente de zero (nesse caso, não é possível realizar a divisão).

Em todas as funções, as variáveis são strings, por isso, os valores utilizados nas operações serão convertidos em valores numéricos reais com a função `parseFloat()`.

Veja na Figura 10.5 um exemplo de uso do aplicativo. Note os campos preenchidos e veja o resultado obtido após clicar no botão de adição.



Figura 10.5 – Execução da "calculadora simples".

## 10.4 Aplicativo para Cálculo de Índice de Massa Corporal (IMC)

Nesse terceiro projeto (podemos nomear como **CalculoIMC**), utilizaremos mais uma opção de formulário com a estrutura **ion-select-option**. Essa é uma estrutura interessante, pois traduz a necessidade de interação com usuário em ambiente de telas reduzidas, como em smartphones e tablets.

Em geral, um elemento select em formulários web abre uma caixa de opções abaixo do componente. Em dispositivos móveis, isso pode ser um problema, por causa do tamanho limitado da tela e da dificuldade de selecionar alguma opção utilizando os dedos.

Veja a seguir o código-fonte que será utilizado para realizar a criação da interface do aplicativo.



---

```
<ion-header>
<ion-toolbar>
<ion-title>
Cálculo de IMC
```

```
</ion-title>
</ion-toolbar>
</ion-header>
<ion-content class="ion-padding">
<ion-item>
<ion-label color="secondary">Sua altura: </ion-label>
<ion-input [(ngModel)]="altura" name="altura"></ion-input>
</ion-item>
<ion-item>
<ion-label color="secondary">Seu peso: </ion-label>
<ion-input [(ngModel)]="peso" name="peso"></ion-input>
</ion-item>
<ion-select [(ngModel)]="opcao" placeholder="Selecione um gênero">
<ion-select-option value="f">Feminino</ion-select-option>
<ion-select-option value="m">Masculino</ion-select-option>
</ion-select>
<ion-button full (click)="calcularIMC()">Calcular IMC</ion-button>
<h1>{{mensagem}}</h1>
</ion-content>
```

---

home.page.html

A Figura 10.6 exibe a tela que foi desenvolvida em HTML para o aplicativo.

## Cálculo de IMC

Sua altura:

---

Seu peso:

---

Selecione um gênero

▼

**CALCULAR IMC**

Figura 10.6 – Tela principal do aplicativo.

O componente **ion-select-option** abre um "pop-up" em primeiro plano, dando as opções para seleção. Na Figura 10.7, vemos o "pop-up" em primeiro plano gerado pela estrutura.

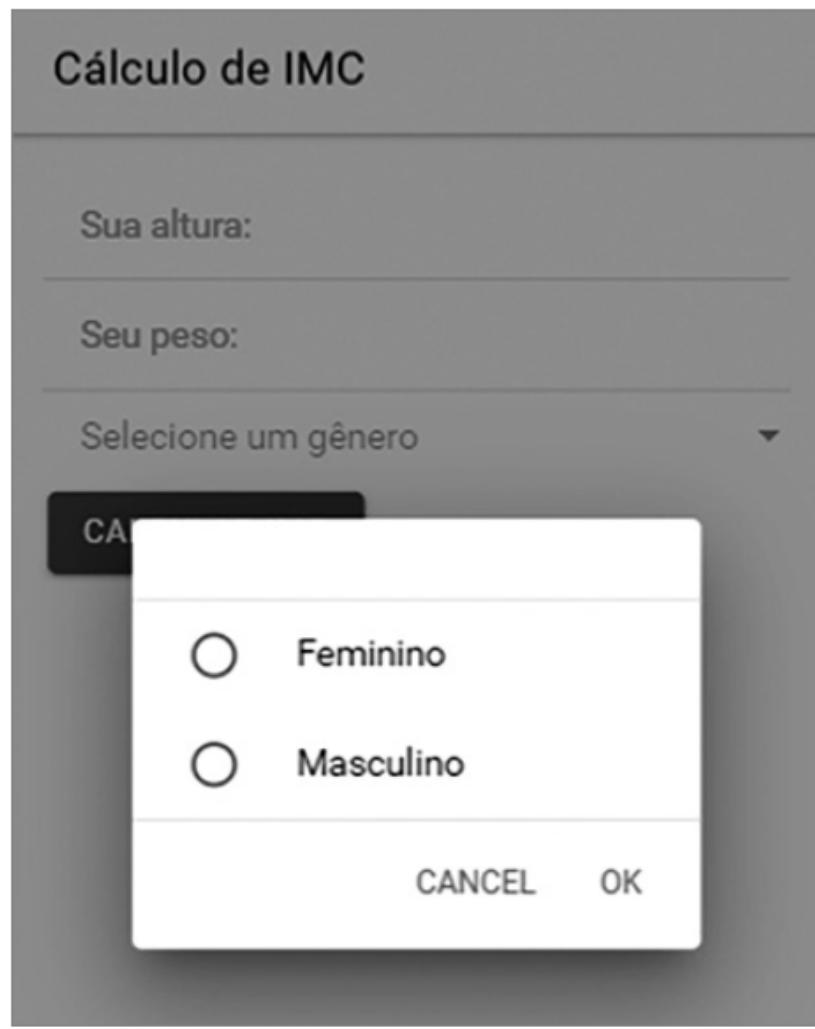


Figura 10.7 – Componente ion-select-option.

O Ionic<sup>1</sup> oferece muitos elementos desenvolvidos especialmente para oferecer boa usabilidade no ambiente destinado aos dispositivos móveis. Uma importante fonte de consulta é sua própria documentação.

Ao executar o aplicativo, basta digitar o peso, a altura e selecionar o sexo. Ao clicar no botão, o cálculo do IMC será realizado por meio da aplicação da fórmula peso/altura<sup>2</sup>. Veja a seguir o código-fonte que deve ser inserido na classe principal do arquivo `home.page.ts`.

{Ionic}

```
export class HomePage {  
  peso : string;
```

```
altura : string;
opcao : string;
mensagem : string;
constructor() {}
calcularIMC(){
var peso = parseFloat(this.peso);
var altura = parseFloat(this.altura);
var imc = peso / Math.pow(altura,2);
if(this.opcao == "f"){
if(imc < 19.1)
this.mensagem = "Você está abaixo do peso";
else if(imc < 25.8)
this.mensagem = "Você está com o peso normal";
else
this.mensagem = "Você acima do peso";
} else if(this.opcao == "m"){
if(imc < 20.7)
this.mensagem = "Você está abaixo do peso";
else if(imc < 26.4)
this.mensagem = "Você está com o peso normal";
else
this.mensagem = "Você acima do peso";
} else{
this.mensagem = "Entre com todos os dados!";
}
}
}
```

home.page.ts

A Figura 10.8 mostra o resultado das entradas e a escolha do gênero feminino.

## Cálculo de IMC

Sua altura: 1.75

Seu peso: 60

Feminino

**CALCULAR IMC**

Você está com o peso  
normal

Figura 10.8 – Componente ion-select-option.

## 10.5 Aplicativo de Contatos Telefônicos

No quinto projeto de aplicativo, utilizaremos um arquivo JSON que contém os dados que serão acessados pela aplicação.

Abordaremos aqui também o conceito de "service", disponível na versão 4.0 do framework Ionic. Esse conceito está diretamente ligado à preocupação de isolar os dados e a aplicação. Isso é muito comum em práticas de acesso a banco de dados, arquivos ou APIs (*Application Programming Interface*) de serviços externos.

Simularemos, assim, uma fonte de dados em um arquivo JSON, que contém dados referentes a contatos telefônicos, incluindo identificador (**id**), nome e número de telefone.

Veja a seguir o arquivo com a extensão **.ts**, mas já com o padrão JSON. O uso da extensão em TypeScript facilita a referência ao volume de dados e, ao mesmo tempo, acessa como um objeto JSON.



```
export var CONTATOS= [  
  {id: 1, nome:"Fulano", telefone:"1111-2222"},  
  {id: 2, nome:"Sicrano", telefone:"5544-3322"},  
  {id: 3, nome:"Beltrano", telefone:"3333-4466"}  
]
```

`dadoscontatos.ts`

Para a utilização do "services" com Ionic, criaremos uma instância de servidor para essa função. Depois da criação do projeto, executaremos uma linha de comando dentro do diretório do projeto. Supondo que o aplicativo tenha o nome de Contatos, acessaremos o diretório do projeto e executaremos o comando no prompt, como mostrado a seguir.



```
ionic generate service services/contatos
```

Após o término da implementação do "service", uma nova pasta será criada no projeto (src > app > services). Ela contém o arquivo **contatos.service.ts**, que será alterado. Dentro dessa pasta, criaremos outra pasta chamada dados (src > app > services > dados).

Dentro da pasta dados, criaremos o arquivo que contém as informações descritas no formato JSON, isto é, src > app > services > dados > dadoscontatos.ts.

A interface exibe apenas o nome dos contatos cadastrados. Ao clicar em algum item da lista, uma caixa de alerta

mostrará o número de telefone em primeiro plano. No código-fonte a seguir temos a implementação da interface.

{Ionic}

```
<ion-header>
<ion-toolbar>
<ion-title>
  Lista de Contatos
</ion-title>
</ion-toolbar>
</ion-header>

<ion-content class = "ion-padding">
<ion-item *ngFor="var contato of contatos" tappable
(click)="verTelefone(contato)">
  <h1>{{contato.nome}}</h1>
</ion-item>
</ion-content>
```

dadoscontatos.ts

Note que há uma estrutura de repetição \*ngFor, que coleta cada um dos itens de uma coleção – nesse caso, cada um dos contatos carregados do arquivo. Em {{ contato.nome }} a referência é o nome de cada contato da lista. Ao clicar nesse item, chamamos a função verTelefone(), cujo parâmetro é o objeto "contato" referente ao item.

Veja na Figura 10.9 um exemplo da interface criada. Ela permite exibir uma lista com o nome dos contatos.

| Lista de Contatos |
|-------------------|
| Fulano            |
| Sicrano           |
| Beltrano          |

Figura 10.9 – Lista dos contatos cadastrados.

Na sequência, alteraremos o arquivo **contato.service.ts**. Nesse arquivo foram implementadas todas as funções que possibilitam consumir os dados do arquivo. Importaremos os contatos, indicando o diretório que contém o arquivo.

Em seguida, definiremos duas funções importantes. A primeira função é a **buscarTodosContatos**, que tem como objetivo retornar todos os contatos do arquivo quando houver uma chamada. A segunda função, chamada **buscarContato**, retorna um contato específico quando ele for selecionado na interface principal. Veja a seguir o código-fonte completo.

{Ionic}

```
import { Injectable } from '@angular/core';
import { CONTATOS } from './dados/dadoscontatos';
@Injectable({
providedIn: 'root'
})
export class ContatosService {
private contatos: any;
```

```
constructor() {  
  this.contatos=CONTATOS;  
}  
  
buscarTodosContatos() {  
  return this.contatos;  
}  
  
buscarContato(id) {  
  for (var i = 0; i < this.contatos.length; i++) {  
    if (this.contatos[i].id === parseInt(id)) {  
      return this.contatos[i];  
    }  
  }  
  return null;  
}  
}
```

contato.service.ts

Na próxima etapa, a lógica principal do aplicativo será implementada. No início do programa, é preciso importar o arquivo com as funções de *service* (conta.service) e o componente de alerta (AlertController), que será utilizado para exibir o número de telefone. No construtor da classe, são usados como parâmetros a referência ao service e a referência ao componente de alerta. Carregaremos também todos os contatos no objeto **contato**.

O próximo passo é criar a função verTelefone(), que recebe o objeto referente ao contato e mostra em uma caixa de alerta o número do telefone. Para essa função, adotaremos o conceito de **função assíncrona**.

A função assíncrona, em nosso contexto, tem como objetivo executar de modo paralelo ao restante da aplicação. Ela fica em espera, até que o usuário interaja com a caixa de alerta.

Usaremos ainda o **async** na definição da função, denotando que ela é assíncrona. Usaremos também a palavra reservada **await** na criação e na apresentação da caixa de alerta, para sinalizar a espera. Observe a seguir o código-fonte completo.

{Ionic}

```
import { Component } from '@angular/core';  
import { ContatosService } from '../services/contatos.service';
```

```
import { AlertController } from '@ionic/angular';
@Component({
selector: 'app-home',
templateUrl: 'home.page.html',
styleUrls: ['home.page.scss'],
})
export class HomePage {
public contatos: any;
constructor( public contatosService:ContatosService,
public alertaTelefone:AlertController) {
this.contatos = this.contatosService.buscartodosContatos();
}
async verTelefone(contato){
var alerta = await this.alertaTelefone.create({
header: "Telefone",
message: contato.telefone,
buttons: ['OK']
});
await alerta.present();
}
}
```

---

home.page.ts

Na Figura 10.10 é possível observar a caixa de alerta que surge em primeiro plano após o usuário clicar sobre o nome de um dos contatos.

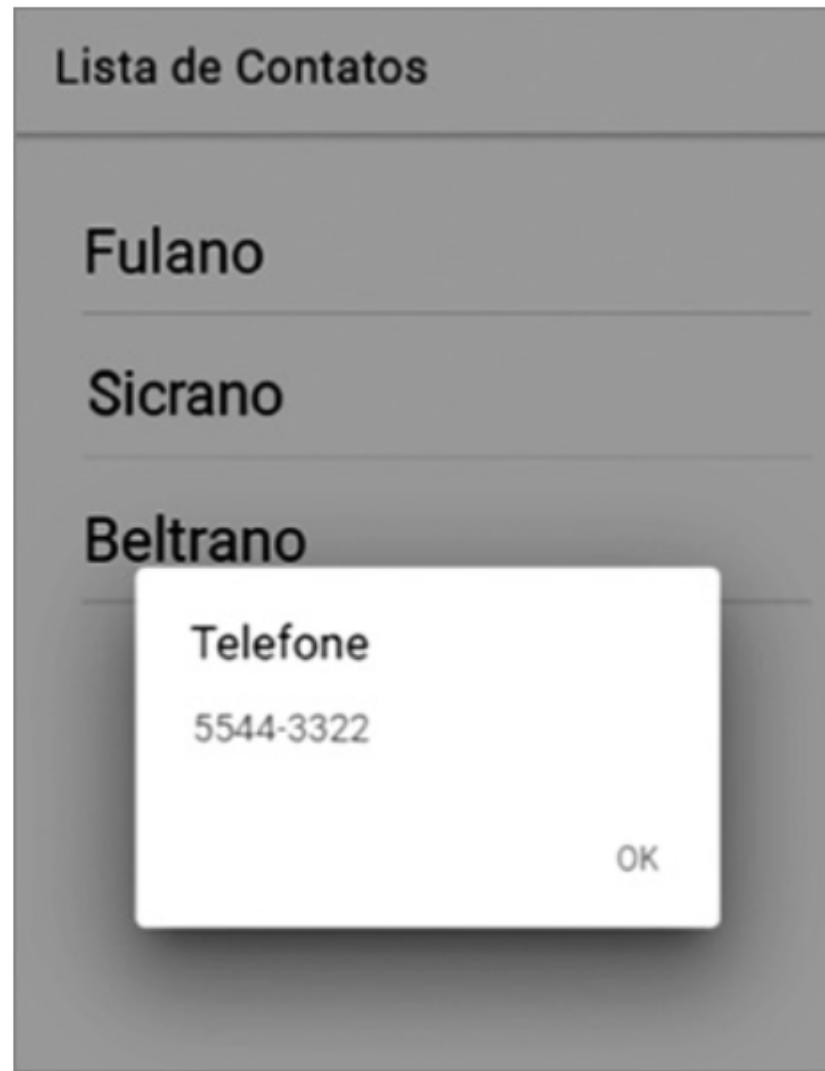


Figura 10.10 – Exibição do telefone de um contato.

### VAMOS PRATICAR!

1. Crie um novo aplicativo com base no primeiro aplicativo apresentado neste capítulo. Deve haver um campo para inserção do seu nome e um botão que, ao ser pressionado, mostre na tela "Olá <seu nome>!".
2. Com base no projeto "calculadora simples", crie um botão para calcular a potenciação. O primeiro campo indica o valor base; o segundo campo indica o valor do expoente.
3. Altere as cores dos campos e dos botões do projeto "Cálculo do IMC", apresentado neste capítulo.
4. Faça duas alterações no projeto "Contatos". Primeiro, acrescente a informação de e-mail para cada um dos contatos. Segundo, modifique o aplicativo para que ele exiba o e-mail junto do número de telefone.
5. Pesquise mais sobre alertas em Ionic, consultando sua documentação no link <https://ionicframework.com/docs/api/alert>. Em seguida, acrescente mais uma propriedade, como a "subheader", no projeto "Contatos".

# Referências Bibliográficas

IONIC FRAMEWORK. Docs. s.d. Disponível em: <https://ionicframework.com/docs>. Acesso em: 18 dez. 2019.

MOZILLA. Elementos HTML. s.d. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element>. Acesso em: 18 dez. 2019.

\_\_\_\_\_. How CSS works. s.d. Disponível em: [https://developer.mozilla.org/en-US/docs/Learn/CSS/First\\_steps/How\\_CSS\\_works](https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/How_CSS_works). Acesso em: 18 dez. 2019.

\_\_\_\_\_. Primeiros passos com CSS. s.d. Disponível em: [https://developer.mozilla.org/pt-BR/docs/Learn/CSS/First\\_steps](https://developer.mozilla.org/pt-BR/docs/Learn/CSS/First_steps). Acesso em: 18 dez. 2019.

\_\_\_\_\_. Introdução ao HTML. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Aprender/HTML>. Acesso em: 18 dez. 2019.

\_\_\_\_\_. Recursos de desenvolvedores para desenvolvedores. Disponível em: <https://developer.mozilla.org/pt-BR/>. Acesso em: 18 dez. 2019.

\_\_\_\_\_. Referência de CSS. s.d. Disponível em: [https://developer.mozilla.org/pt-BR/docs/Web/CSS/CSS\\_Reference](https://developer.mozilla.org/pt-BR/docs/Web/CSS/CSS_Reference). Acesso em: 18 dez. 2019.

NODE.JS. About docs. Disponível em: <https://nodejs.org/en/docs/>. Acesso em: 18 dez. 2019.