



INSTITUTO FEDERAL

Espírito Santo

Campus Cachoeiro de Itapemirim

JavaScript DOM

Apostila Didática

Prof. Rafael Vargas Mesquita

Caro estudante,

É um prazer recebê-lo na apostila sobre **JavaScript DOM**! Este material foi cuidadosamente desenvolvido para proporcionar a você uma jornada envolvente pelos conceitos essenciais do Document Object Model (DOM) em JavaScript. Durante essa jornada, você explorará três pilares fundamentais:

1. Selecionando Elementos: No primeiro capítulo, você mergulhará no processo de seleção de elementos no contexto do HTML, CSS e JavaScript. Compreenderá como o DOM se torna a ponte que conecta esses elementos, permitindo a manipulação dinâmica do conteúdo da página.

2. Manipulando Eventos: No segundo capítulo, você se aprofundará no fascinante mundo dos eventos. Aprenderá a detectar e responder a ações do usuário, como cliques, teclas pressionadas e movimentos do mouse. Descobrirá como transformar esses eventos em interações significativas, tornando suas páginas web mais dinâmicas e responsivas.

3. Trabalhando com Formulários: No terceiro capítulo, você explorará os detalhes dos formulários. Desde a criação até a validação, você dominará as técnicas para coletar dados dos usuários de forma eficaz e segura. Entenderá como o JavaScript pode aprimorar a experiência do usuário, tornando a interação com formulários mais intuitiva e livre de erros.

Esta apostila é mais do que um guia; é um convite para você explorar a riqueza do DOM em JavaScript. Por meio de explicações detalhadas, exemplos práticos e exercícios envolventes, você será capacitado a criar interações web dinâmicas e responsivas.

Desejamos a você uma jornada de aprendizado estimulante e repleta de descobertas. Que este material seja sua bússola confiável na exploração do DOM em JavaScript. Preparado para começar esta emocionante jornada?

Sumário

I

Parte I: Conceitos

1	Selecionando elementos	15
1.1	HTML, CSS, JavaScript	15
1.2	O que é o DOM?	16
1.3	Árvore DOM	17
1.4	Selecionando Elementos	20
1.4.1	Pela tag: <code>getElementsByName()</code>	20
1.4.2	Por identificação: <code>getElementById()</code>	21
1.4.3	Por nome: <code>getElementsByName()</code>	21
1.4.4	Por classe CSS: <code>getElementsByClassName()</code>	21
1.4.5	Por seletor: <code>querySelector</code> e <code>querySelectorAll</code>	21
1.5	Exercícios Propostos	23
1.6	Gabarito dos Exercícios	23
2	Manipulando Eventos	25
2.1	O que são eventos?	25
2.2	Categorias de eventos	25

2.3	Como usar eventos?	26
2.3.1	Manipuladores de eventos in-line	26
2.3.2	Propriedades do manipulador de eventos	26
2.3.3	addEventListener()	27
2.3.4	Exemplo com manipuladores de eventos	27
2.4	Exercícios Propostos	30
2.5	Gabarito dos Exercícios	34
3	Trabalhando com formulários	35
3.1	Arquitetura Cliente/Servidor	36
3.2	O que são Formulários?	38
3.3	Como Validar Formulários?	42
3.3.1	Tipos de Validações	43
3.4	Exercícios Propostos	45
3.5	Gabarito dos Exercícios	46

II

Parte II: Práticas

4	Selecionando elementos - Prática	49
5	Manipulando eventos - Prática	53
5.1	Utilizando manipuladores in-line	54
5.2	Utilizando listener	55
5.3	Delegando eventos	56
5.4	Criando elementos	58
6	Trabalhando com formulários - Prática	61
6.1	Validação com HTML	62

6.2	Validação com JavaScript	64
6.3	Validação com Bootstrap	68

Lista de Figuras

1.1	Exemplo de projeto com HTML, CSS e JavaScript.	15
1.2	DOM como representação em memória (parte 1).	16
1.3	DOM como representação em memória (parte 2).	17
1.4	Analogia entre árvore e árvore DOM	18
1.5	Criando o DOM de uma página HTML	18
1.6	Elementos pai e filhos no DOM	19
1.7	Elementos ascendentes e descendentes no DOM	19
2.1	Como usar eventos?	27
3.1	Interação Cliente/Servidor com um Formulário de Cadastro.	37
3.2	Construção de formulários	38
3.3	Diferença entre os métodos GET e POST.	40
3.4	Cenários de utilização dos métodos GET e POST.	41
3.5	Validação de formulários	44
4.1	Selecionando elementos - Prática	49
5.1	Manipulando eventos - Práticas	53
6.1	Trabalhando com formulários - Prática	61

Lista de Exemplos de Códigos

1.1	Selecionando elementos pela tag	21
1.2	Selecionando elementos pelo ID	21
1.3	Selecionando elementos pelo nome	21
1.4	Selecionando elementos pela classe	21
1.5	Selecionando elementos por seletor CSS	21
1.6	Selecionando todos elementos por seletor CSS	22
2.1	Manipuladores de eventos in-line	26
2.2	Propriedades do manipulador de eventos	27
2.3	addEventListener()	27
2.4	removeEventListener()	29
2.5	Último manipulador ativo para o mesmo ouvinte	29
2.6	Registrando vários manipuladores para o mesmo ouvinte	29
3.1	Sintaxe da construção de um formulários	39
4.1	Selecionando elementos - Prática (index.html)	50
4.2	Selecionando elementos - Prática (style.css)	51
4.3	Selecionando elementos - Prática (script.js)	52
5.1	Utilizando manipuladores in-line - Prática (index.html)	54
5.2	Utilizando manipuladores in-line - Prática (script.js)	55
5.3	Utilizando listener - Prática (index.html)	55
5.4	Utilizando listener - Prática (script.js)	56
5.5	Delegando eventos - Prática (index.html)	57

5.6	Delegando eventos - Prática (style.css)	57
5.7	Delegando eventos - Prática (script.js)	58
5.8	Criando elementos - Prática (index.html)	59
5.9	Criando elementos - Prática (style.css)	59
5.10	Criando elementos - Prática (script.js)	60
6.1	Validando formulários com HTML - Prática (index.html)	63
6.2	Validando formulários com HTML - Prática (style.css)	64
6.3	Validando formulários com JavaScript - Prática (index.html)	66
6.4	Validando formulários com JavaScript - Prática (style.css)	67
6.5	Validando formulários com JavaScript - Prática (script.js)	68
6.6	Validando formulários com Bootstrap - Prática (index.html)	71
6.7	Validando formulários com Bootstrap - Prática (script.js)	72



Parte I: Conceitos

1	Selecionando elementos	15
2	Manipulando Eventos	25
3	Trabalhando com formulários	35

1. Selecionando elementos

1.1 HTML, CSS, JavaScript

Antes de mergulharmos no mundo do DOM (Document Object Model), é importante entendermos as três principais tecnologias que formam a base da web moderna: HTML, CSS e JavaScript.

- HTML: linguagem de marcação utilizada para estruturar os elementos da página, como parágrafos, links, títulos, tabelas, imagens e até vídeos;
- CSS: linguagem de estilos utilizada para definir cores, fontes, tamanhos, posicionamento e qualquer outro valor estético para os elementos da página;
- JavaScript: linguagem de programação utilizada para deixar a página com mais movimento, podendo atualizar elementos dinamicamente.

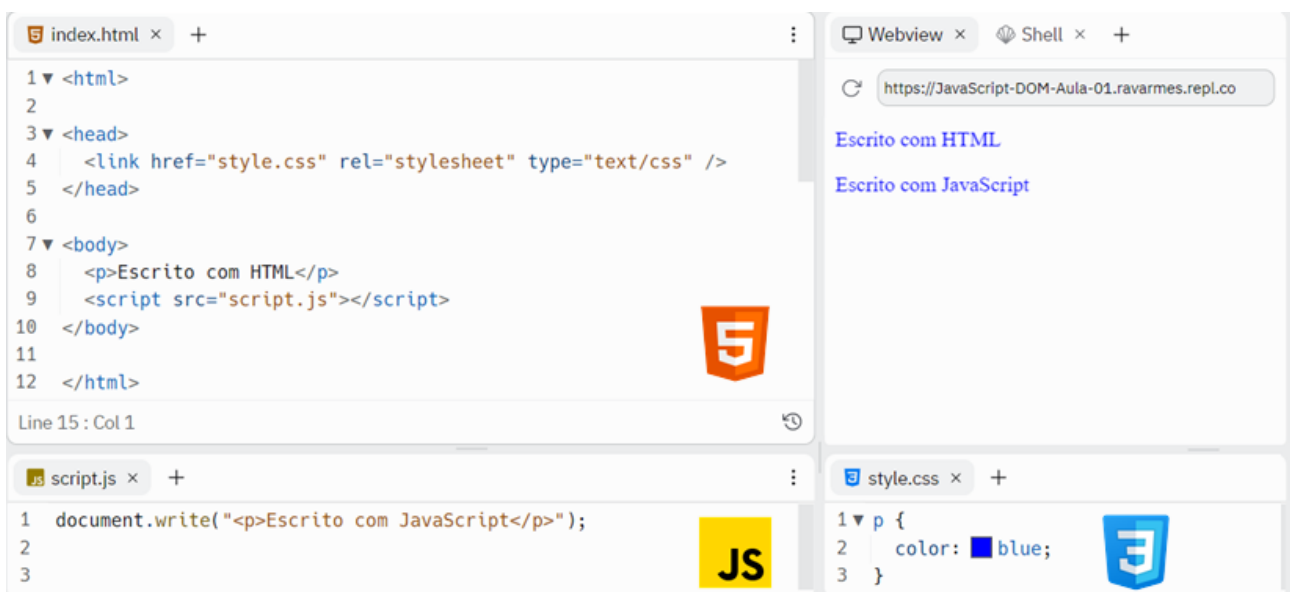


Figura 1.1: Exemplo de projeto com HTML, CSS e JavaScript.

O projeto apresentado na figura 1.1 contém um exemplo de utilização dos três tipos de arquivos:

- HTML: com o conteúdo da página e referências aos arquivos CSS e JavaScript, respectivamente nas linhas 4 e 9;
- CSS: contendo a estilização dos parágrafos (tag <p>);
- JavaScript: com uma instrução de código para escrever um parágrafo na página.

No canto superior direito, temos o resultado da execução do projeto sendo exibido no navegador. A primeira linha exibida na página é um parágrafo escrito pelo arquivo HTML (index.html). A segunda linha é um parágrafo escrito pelo arquivo JavaScript (script.js). Em ambas as linhas a definição da cor azul para o texto dos dois parágrafos foi especificada no arquivo CSS (style.css).

1.2 O que é o DOM?

O DOM, ou Modelo de Objeto de Documento, é uma representação estruturada em forma de árvore de todos os elementos em um documento HTML ou XML. Ele permite que os programadores interajam com os elementos da página de forma programática, possibilitando a alteração dinâmica do conteúdo, estilo e comportamento das páginas web.

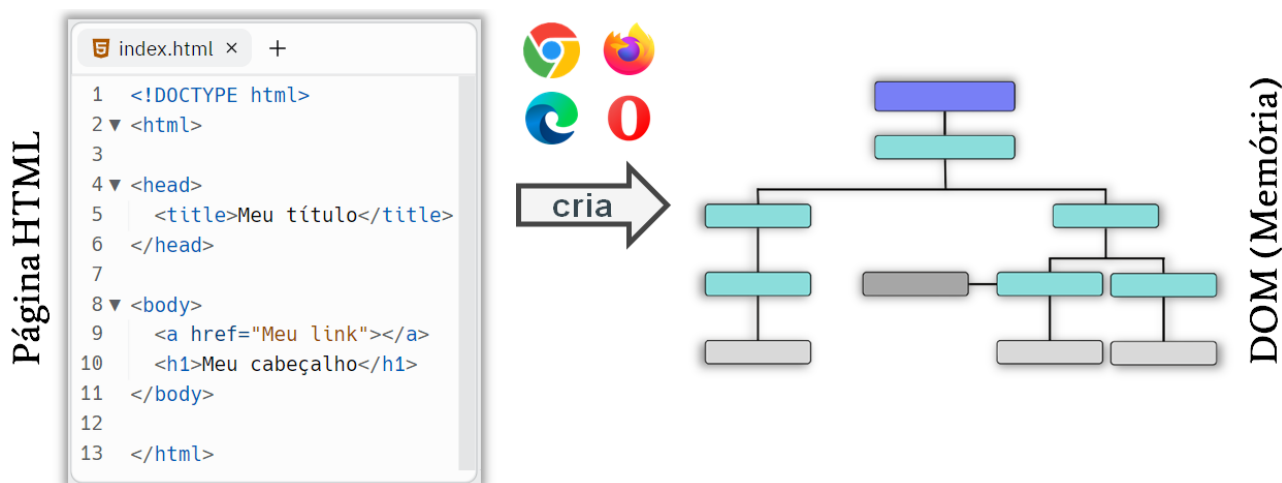


Figura 1.2: DOM como representação em memória (parte 1).

A figura 1.2 mostra o DOM como uma representação em memória da estrutura do arquivo HTML. De forma mais detalhada, corresponde aos dados que compõem a estrutura e o conteúdo de um documento na Web. Quando uma página é carregada, o navegador cria um DOM da página.

Talvez, neste momento, você possa estar se perguntando se o DOM faz parte do HTML ou do

JavaScript? E na verdade, ele não é, necessariamente, parte de nenhum deles. Ele é uma interface de programação para os documentos HTML.

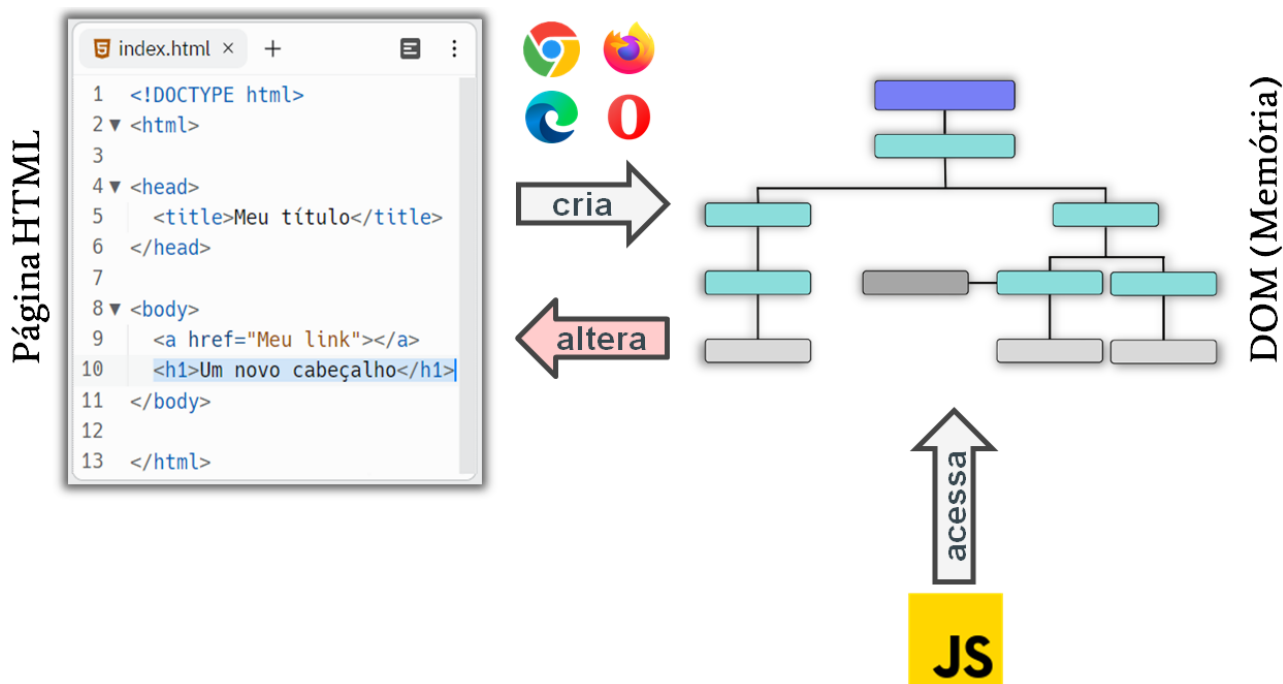


Figura 1.3: DOM como representação em memória (parte 2).

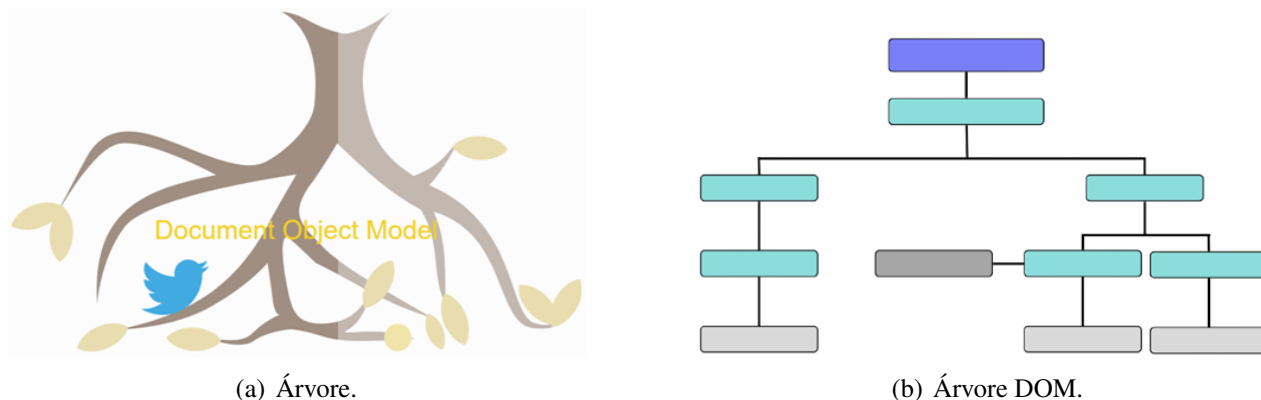
A figura 1.3 mostra como o JavaScript pode acessar o DOM pra refletir alterações na página HTML. Neste caso específico, alteramos o cabeçalho da página (tag <div>).

Para que estas alterações sejam efetivadas é preciso que o JavaScript interaja com os objetos DOM. Estes conceitos ficarão mais claros à seguir.

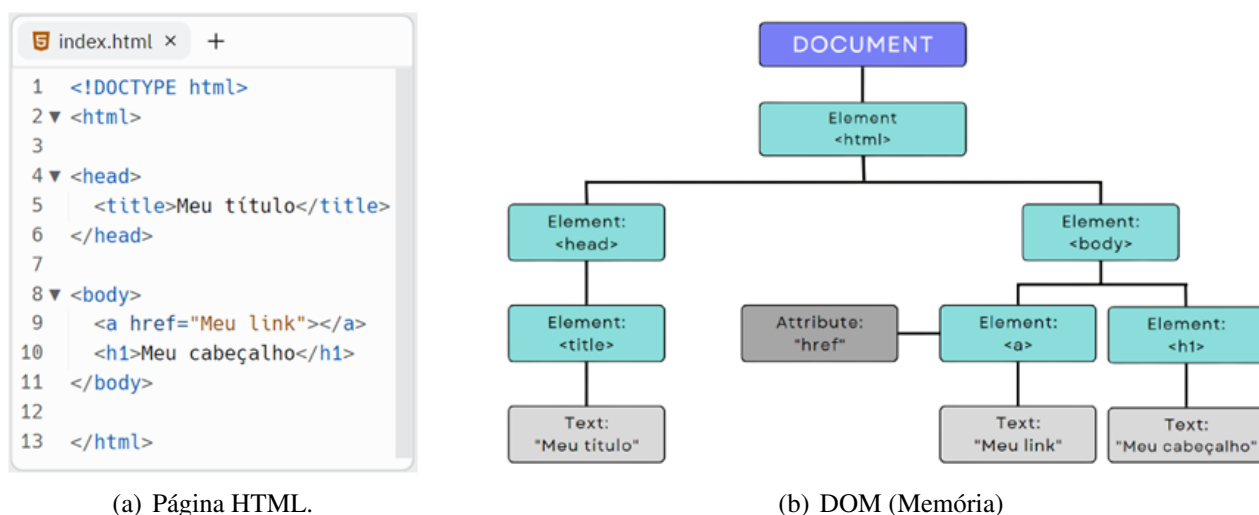
1.3 Árvore DOM

A estrutura da árvore DOM é uma representação hierárquica dos elementos presentes em um documento HTML. Cada elemento, como tags <div>, <p>, , etc., é um nó na árvore, com relacionamentos pai-filho que espelham a organização do documento. Isso permite que os desenvolvedores acessem e manipulem esses elementos usando JavaScript.

Os elementos aninhados de um documento HTML são representados no DOM como uma árvore de objetos. Na figura 1.4 temos uma analogia da árvore DOM 1.4(b) com a árvore à esquerda 1.4(a), percebemos que a análise do DOM considera uma visualização à partir da sua raiz, a qual fica na parte superior. Na figura 1.4(b), cada caixa representa um objeto da árvore DOM.

**Figura 1.4:** Analogia entre árvore e árvore DOM

A figura 1.5 detalha um pouco mais como é criado o DOM. Aqui nós estamos considerando a geração de um DOM, à direita 1.5(b), baseado, na página HTML, à esquerda 1.5(a).

**Figura 1.5:** Criando o DOM de uma página HTML

O DOM representa o documento HTML com seus respectivos objetos. Cada caixa é um nó (Node) do documento e é representado por um objeto.

Note que a figura 1.5(b) contém quatro tipos diferentes de nós. Na raiz da árvore está o nó **Document**, que representa o documento inteiro. Os nós que representam elementos HTML são nós do tipo **Element**. Os nós que representam atributos são nós do tipo **Atributte**. Os nós que representam texto são nós do tipo **Text**.

Document, Element, Text e Atributte são subclasses de *Node*. Todavia, não se preocupe com esses nomes, pois à medida que você for programar utilizando o DOM, vai se acostumando com os termos.

A seguir, nas figuras 1.6 e 1.7, vamos abordar a relação que existe entre os nós. Caso você não esteja familiarizado com as estruturas de árvores na programação de computadores, é útil saber que elas emprestam a terminologia das árvores genealógicas. Portanto:

- O nó imediatamente acima de outro é o pai desse nó;
- Os nós um nível imediatamente abaixo de outro nó são os filhos desse nó;
- Os nós no mesmo nível e com o mesmo pai são irmãos;
- O conjunto de nós a qualquer número de níveis abaixo de outro nó são os descendentes desse nó;
- E o pai, avô e todos os outros nós acima de um nó são os ascendentes desse nó.

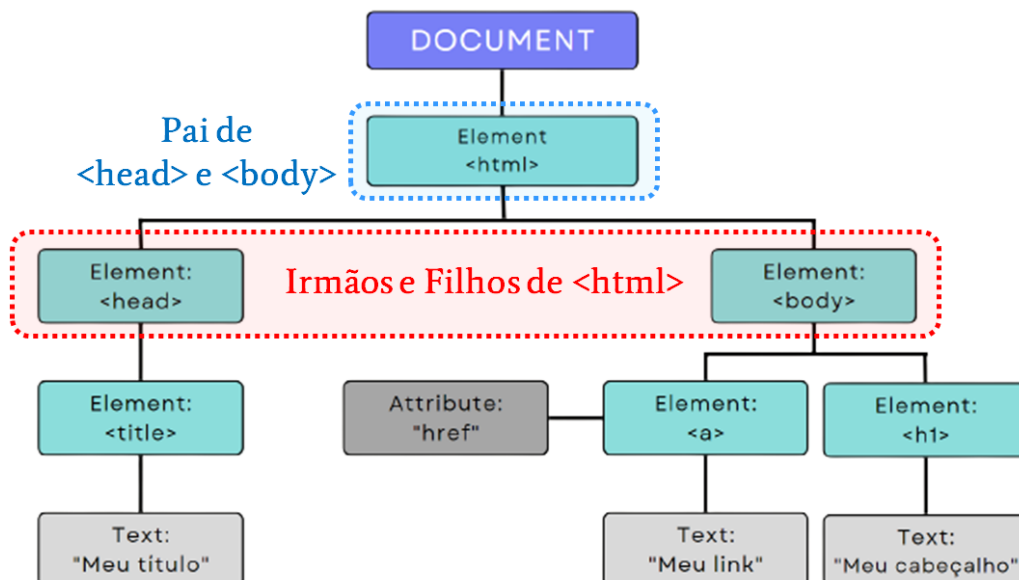


Figura 1.6: Elementos pai e filhos no DOM

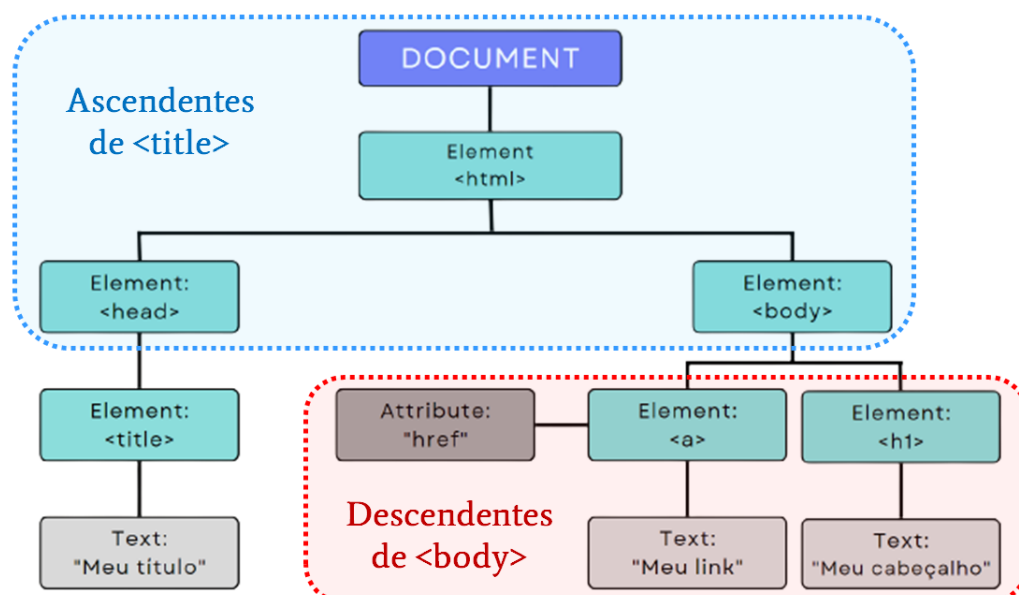


Figura 1.7: Elementos ascendentes e descendentes no DOM

Com o DOM, o JavaScript tem todo o poder necessário para criar HTML dinamicamente:

- Alterando elementos HTML;
- Alterando atributos dos elementos HTML;
- Alterando estilos CSS;
- Removendo elementos HTML e seus atributos;
- Adicionando elementos HTML e seus atributos;
- Reagindo aos eventos que ocorrerem;
- Criando novos eventos.

Os objetos do DOM possuem métodos (funções) e atributos (propriedades). São exemplos de funções do objeto **Document**:

- `getElementById`;
- `write`;
- `addEventListener`.

São exemplos de propriedades do objeto **Document**:

- `innerHTML`;
- `value`;
- `style`.

Ainda neste capítulo, na seção relacionada à prática, vamos trabalhar com estas funções e propriedades.

1.4 Selecionando Elementos

A manipulação dos elementos na página começa com a seleção dos elementos corretos no DOM. Vamos explorar várias técnicas para selecionar elementos de maneira eficiente e flexível:

1.4.1 Pela tag: `getElementsByTagName()`

O método `getElementsByTagName()` permite selecionar todos os elementos que possuem uma determinada tag. Por exemplo, para selecionar todos os parágrafos (`<p>`) em um documento, você pode usar:

```
1 const paragrafos = document.getElementsByTagName("p");
```

Exemplo de Código 1.1: Selecionando elementos pela tag

1.4.2 Por identificação: getElementById()

A seleção por identificação é uma maneira rápida e eficiente de obter um elemento específico com base no seu atributo id. Por exemplo, para selecionar um elemento com o id "titulo", você pode usar:

```
1 const elementoTitulo = document.getElementById("titulo");
```

Exemplo de Código 1.2: Selecionando elementos pelo ID

1.4.3 Por nome: getElementsByName()

O método getElementsByName() permite selecionar todos os elementos que possuem um determinado atributo name. Essa abordagem é mais comum em elementos de formulários. Por exemplo, para selecionar todos os elementos com o atributo name "email", você pode usar:

```
1 const elementosEmail = document.getElementsByName("email");
```

Exemplo de Código 1.3: Selecionando elementos pelo nome

1.4.4 Por classe CSS: getElementsByClassName()

A seleção por classe é uma maneira de agrupar elementos com estilos semelhantes. O método getElementsByClassName() permite selecionar todos os elementos que possuem uma determinada classe CSS. Por exemplo, para selecionar todos os elementos com a classe CSS "destaque", você pode usar:

```
1 const elementosDestaque = document.getElementsByClassName("destaque");
```

Exemplo de Código 1.4: Selecionando elementos pela classe

1.4.5 Por seletor: querySelector e querySelectorAll

A seleção por seletor CSS é uma técnica poderosa que permite escolher elementos com base em seletores CSS, semelhante ao que você faria em uma folha de estilo. O método querySelector() seleciona o primeiro elemento que corresponde ao seletor especificado. Por exemplo:

```
1 const primeiroParagrafo = document.querySelector("p");
```

Exemplo de Código 1.5: Selecionando elementos por seletor CSS

O método `querySelectorAll()` seleciona todos os elementos que correspondem ao seletor. Por exemplo:

```
1 const todosOsParagrafos = document.querySelectorAll("p");
```

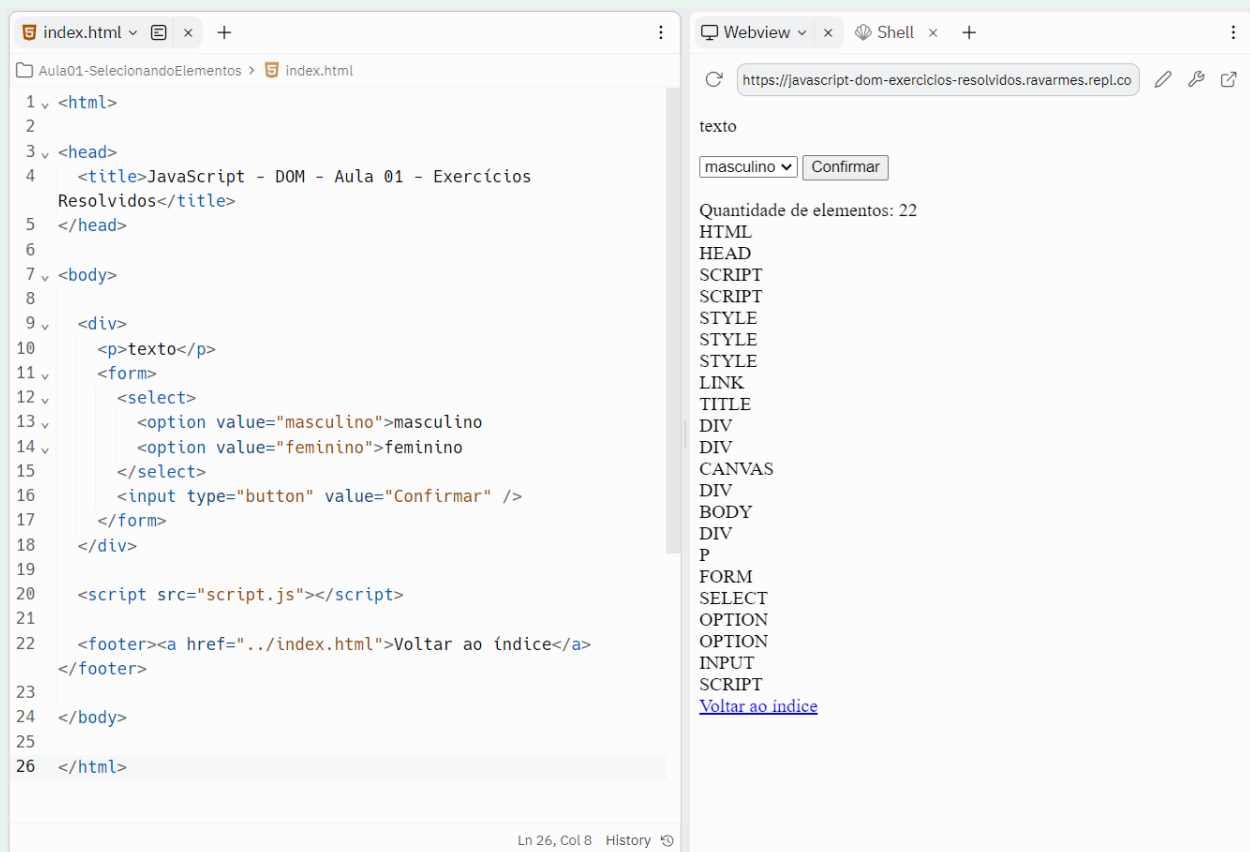
Exemplo de Código 1.6: Selecionando todos elementos por seletor CSS

Essas técnicas de seleção permitem que você acesse e manipule os elementos desejados de forma precisa e eficiente, facilitando a criação de interações dinâmicas em suas páginas web.

1.5 Exercícios Propostos

Exercício 1.1 — Seleccionando elementos: Exercício 01. Elabore um algoritmo no arquivo `script.js`, utilizando linguagem de Programação JavaScript, para contar e mostrar todos elementos existentes em um página codificada no arquivo `index.html`.

Na figura a seguir mostramos à esquerda, o código da página `index.html`. À direita, mostramos o resultado da exibição do site, após a elaboração do algoritmo no arquivo `script.js`.



1.6 Gabarito dos Exercícios

Gabarito 1.1 — Seleccionando elementos: Exercício 01.



2. Manipulando Eventos

2.1 O que são eventos?

Eventos são ocorrências a respeito das quais seu programa será notificado pelo navegador Web. Eles desempenham um papel crucial na interatividade das páginas web, permitindo que você responda às ações do usuário e a outras mudanças no ambiente de execução. Alguns exemplos de eventos incluem:

- O usuário clicando com o mouse sobre um certo elemento ou passando o cursor do mouse sobre um certo elemento;
- O usuário pressionando uma tecla do teclado;
- O usuário redimensionando ou fechando a janela do navegador;
- Uma página da web terminando de carregar;
- Um formulário sendo enviado;
- Um vídeo sendo reproduzido, interrompido, ou terminando sua reprodução;
- Um erro ocorrendo.

2.2 Categorias de eventos

Os eventos são classificados em várias categorias, cada uma relacionada a diferentes tipos de interações e comportamentos da página. Algumas categorias comuns de eventos incluem:

- Mouse
- Teclado
- Formulários
- Recursos

- Rede
- Foco
- WebSocket
- Histórico de sessão
- Animações CSS
- Transição CSS
- Impressão
- Composição de texto
- Tela
- Área de Transferência
- Arrastar e soltar
- Mídia
- Progresso, etc.

2.3 Como usar eventos?

Cada evento disponível possui um manipulador de eventos, que é um bloco de código (geralmente uma função JavaScript definida pelo usuário) que será executado quando o evento for disparado. Existem várias maneiras de associar manipuladores de eventos a elementos HTML:

2.3.1 Manipuladores de eventos in-line

Uma maneira simples de definir um manipulador de eventos é inseri-lo diretamente no atributo HTML do elemento. Por exemplo:

```
1 <button onclick="minhaFuncao()">Clique em mim</button>
```

Exemplo de Código 2.1: Manipuladores de eventos in-line

2.3.2 Propriedades do manipulador de eventos

Outra abordagem é atribuir uma função diretamente a uma propriedade de evento do objeto DOM. Por exemplo:

```
1 const meuElemento = document.getElementById("meuElemento");  
2 meuElemento.onclick = minhaFuncao;
```

Exemplo de Código 2.2: Propriedades do manipulador de eventos**2.3.3 addEventListener()**

Uma forma mais flexível é usar o método `addEventListener()` para associar um manipulador de eventos a um elemento. Isso permite a adição de múltiplos manipuladores para um mesmo evento sem sobrescrever o existente:

```
1 const meuBotao = document.querySelector("button");
2 meuBotao.addEventListener("click", minhaFuncao);
```

Exemplo de Código 2.3: `addEventListener()`**2.3.4 Exemplo com manipuladores de eventos**

Essas são algumas das maneiras de usar eventos para criar interações dinâmicas e responsivas em suas páginas web. Através da manipulação de eventos, você pode fazer suas aplicações responderem de forma inteligente às ações dos usuários e ao ambiente.

**Figura 2.1:** Como usar eventos?

No código apresentado na figura 2.1 temos três botões no documento HTML. Cada um desses botões, quando clicado, irá acionar um manipulador de evento. Ou seja, quando o usuário clicar em algum botão, o navegador Web vai executar algum trecho de código associado a ocorrência deste tipo de evento neste botão em específico.

- O **botão 1**, por exemplo, possui um atributo `onclick` especificado no próprio documento HTML. Este tipo de tratamento de evento é conhecido como **‘manipulador de evento inline’**. O valor do atributo é literalmente o código JavaScript que você deseja executar quando o evento ocorre. Você pode chamar uma função, definida no arquivo de extensão `js`, ou, como neste exemplo, escrever o código diretamente. Neste caso o código é uma mensagem de alerta. No entanto, este tipo de manipulação de evento é uma prática ruim. Para começar, não é uma boa ideia misturar o seu HTML e o seu JavaScript, pois é difícil analisar — manter seu JavaScript em um só lugar é melhor. Caso esteja em um arquivo separado, você vai poder aplicá-lo a vários documentos HTML. Outra situação complicada neste tipo de manipulação seria se tivéssemos 100 botões, por exemplo. Neste caso, você deveria criar 100 atributos para cada botão. E dar manutenção nisso poderia ser um problema. Utilizando outros tipos de manipuladores agente conseguiria fazer o mesmo de forma bem mais fácil.
- No caso do **botão 2**, nós estamos manipulando o evento por meio de propriedade do elemento `button`. Observe que o código HTML na linha 7 possui apenas o elemento `button` e seu identificador. Já no arquivo `script.js`, nas linhas 1 e 2, nós selecionamos o elemento `button` com o id `btn2`, e acessamos a propriedade `onclick` desse elemento selecionado. A propriedade `onclick` é a propriedade do manipulador de eventos que está sendo usada nesta situação. É, essencialmente, uma propriedade como qualquer outra disponível no botão (por exemplo: `btn2.textContent`, ou `btn2.style`), mas é um tipo de propriedade especial — neste tipo de situação, quando você configura a propriedade para ser igual a algum código, esse código vai ser executado quando o evento for acionado no botão.
- O **botão 3** é um exemplo de utilização de **ouvinte**, ou **listener**. O mais novo tipo de mecanismo de evento é definido na Especificação de Eventos Nível 2 do DOM. Ele funciona de maneira semelhante às propriedades do manipulador de eventos, mas a sintaxe é, obviamente, diferente. Dentro da função `addEventListener()`, especificamos três parâmetros — 1º o nome do evento para o qual queremos registrar esse manipulador, 2º o código que compreende a função do manipulador que queremos executar em resposta a ele, e o 3º, opcional na maioria dos navegadores, é um atributo booleano, normalmente com valor `false`, no qual nós definimos se desejamos ou não iniciar uma captura. Como neste material vamos utilizar sempre o valor

false, não entraremos em detalhes sobre a diferença, caso utilizássemos true.

Esse terceiro mecanismo tem algumas vantagens sobre os mecanismos mais antigos discutidos anteriormente.

Para começar, há uma função de contraparte, `removeEventListener()`, que remove um listener adicionado anteriormente. Isso não é significativo para programas pequenos e simples, mas para programas maiores e mais complexos, pode melhorar a eficiência para limpar antigos manipuladores de eventos não utilizados.

```
1 btn.removeEventListener("click", functionA);
```

Exemplo de Código 2.4: `removeEventListener()`

Em segundo lugar, você também pode registrar vários manipuladores para o mesmo ouvinte. Diferente do que conseguimos fazer com o segundo manipulador, no qual, apenas o último evento ficaria registrado.

```
1 myElement.onclick = functionA;  
2 myElement.onclick = functionB;
```

Exemplo de Código 2.5: Último manipulador ativo para o mesmo ouvinte

```
1 myElement.addEventListener("click", functionA);  
2 myElement.addEventListener("click", functionB);
```

Exemplo de Código 2.6: Registrando vários manipuladores para o mesmo ouvinte

Mas então, qual mecanismo você deve usar? O primeiro, definitivamente é melhor evitar. Já em relação aos outros dois, você vai ter que considerar que o segundo é menos poderoso que o terceiro, mas possui maior compatibilidade (sendo suportado desde o Internet Explorer 8). O terceiro é mais poderoso, mas menos compatível (suportados desde o Internet Explorer 9).

Portanto, opte, sempre que possível, pelo terceiro manipulador.

2.4 Exercícios Propostos

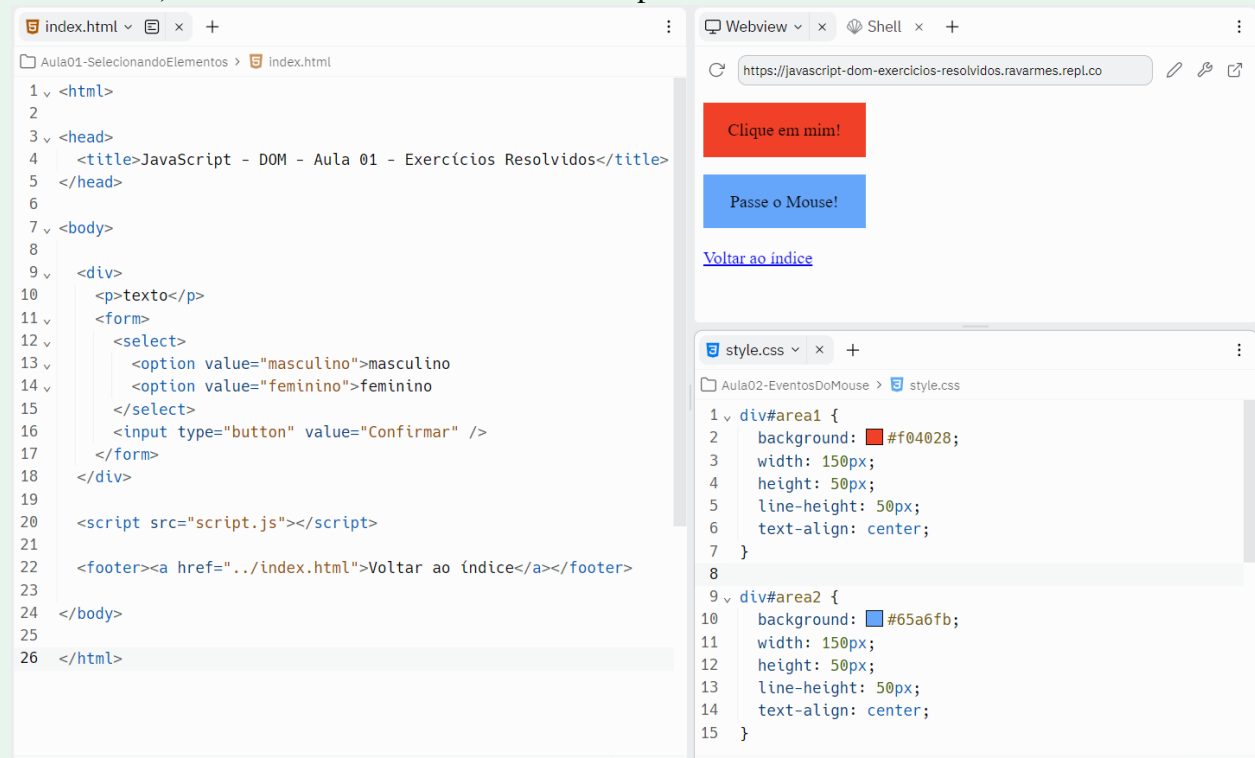
Exercício 2.1 — Eventos do mouse: Exercício 01. Elabore um algoritmo no arquivo script.js para implementar a manipulação de eventos do mouse (com `addEventListener`). Na figura a seguir mostramos à esquerda, o código da página index.html. Na parte superior à direita, mostramos o resultado da exibição do site, após a elaboração do algoritmo no arquivo script.js. Na parte inferior à direita, temos o código do arquivo style.css. O script.js deve considerar os seguintes eventos:

Na área em **vermelho**:

- evento `mousedown` (ocorre quando o usuário pressiona um botão do mouse): altere o texto da área em vermelho para ‘Solte-me!’ e altere o `backgroundColor` para ‘1ec5e5’;
- evento `mouseup` (ocorre quando o usuário solta um botão do mouse): altere o texto da área em vermelho para ‘Obrigado’ e altere o `backgroundColor` para ‘f04028’.

Na área em **azul**:

- evento `mouseover` (ocorre quando o mouse é movido para dentro do elemento que esteja escutando): altere o texto da área em vermelho para ‘Obrigado!’;
- evento `mouseout` (ocorre quando o mouse é movido para fora do elemento que esteja escutando): altere o texto da área em vermelho para ‘Passe o Mouse!’.



Exercício 2.2 — Capturando valores: Exercício 02. Elabore um algoritmo no arquivo script.js, utilizando linguagem de Programação JavaScript, para implementar a captura de valores de campos do tipo text.

Na figura a seguir mostramos à esquerda, o código da página index.html. À direita, mostramos o resultado da exibição do site, após a elaboração do algoritmo no arquivo script.js.

O código a ser elaborado no arquivo script.js deve considerar o seguinte evento: No botão Calcular:

- evento click (ocorre quando o usuário clica o botão do mouse): capture os valores dos dois campos do tipo text (peso e altura). Calcule o $imc = peso / (altura * altura)$. E, por fim, imprima o valor do IMC na div com o id="divIMC".

Observação: para implementação dos eventos utilize a função 'addEventListener'.



Exercício 2.3 — Adicionando elementos: Exercício 03. Elabore um algoritmo no arquivo `script.js`, utilizando linguagem de Programação JavaScript, para implementar a adição de elementos na página.

Na figura a seguir mostramos à esquerda, o código da página `index.html`. À direita, mostramos o resultado da exibição do site, após a elaboração do algoritmo no arquivo `script.js`.

O código a ser elaborado no arquivo `script.js` deve considerar o seguinte evento no botão **Adicionar**:

- evento `click` (ocorre quando o usuário clica o botão do mouse): gere aleatoriamente um número, de 1 até 100, utilizando a instrução `'Math.random() * (100) + 1'`. O número gerado deve ser adicionado como uma nova linha da tabela de números.

Observação: para implementação dos eventos utilize a função `'addEventListener'`.

The screenshot shows a web browser with two panes. The left pane displays the HTML code for `index.html`, and the right pane shows the rendered page.

HTML Code (Left Pane):

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <meta charset="utf-8">
6   <meta name="viewport" content="width=device-width">
7   <title>JavaScript - DOM - Adicionando Elementos</title>
8 </head>
9
10 <body>
11   <h2> Números aleatórios de 1 até 100 </h2>
12
13   <table id="tabela1" border="1">
14     <tr>
15       <th>Números</th>
16     </tr>
17   </table>
18
19   <br>
20
21   <input type="button" id="btnAdicionar" value="Adicionar">
22
23   <br>
24   <br>
25   <script src="script.js"></script>
26 </body>
27
28 </html>
```

Rendered Page (Right Pane):

The page title is "Números aleatórios de 1 até 100". It features a table with the following content:

Números
54
17
64
13
29

Below the table is a button labeled "Adicionar".

Exercício 2.4 — Alterando imagens: Exercício 04. Elabore um algoritmo no arquivo script.js, utilizando linguagem de Programação JavaScript, para implementar a alteração de imagens na página.

Na figura a seguir mostramos à esquerda, o código da página index.html. Na parte superior à direita, mostramos o resultado da exibição do site, após a elaboração do algoritmo no arquivo script.js. Na parte inferior à direita, temos o código do arquivo style.css.

O código a ser elaborado no arquivo script.js deve considerar os seguintes eventos:

No botão **HTML**:

- evento click: altere a imagem para html.png;

No botão **CSS**:

- evento click: altere a imagem para css.png;

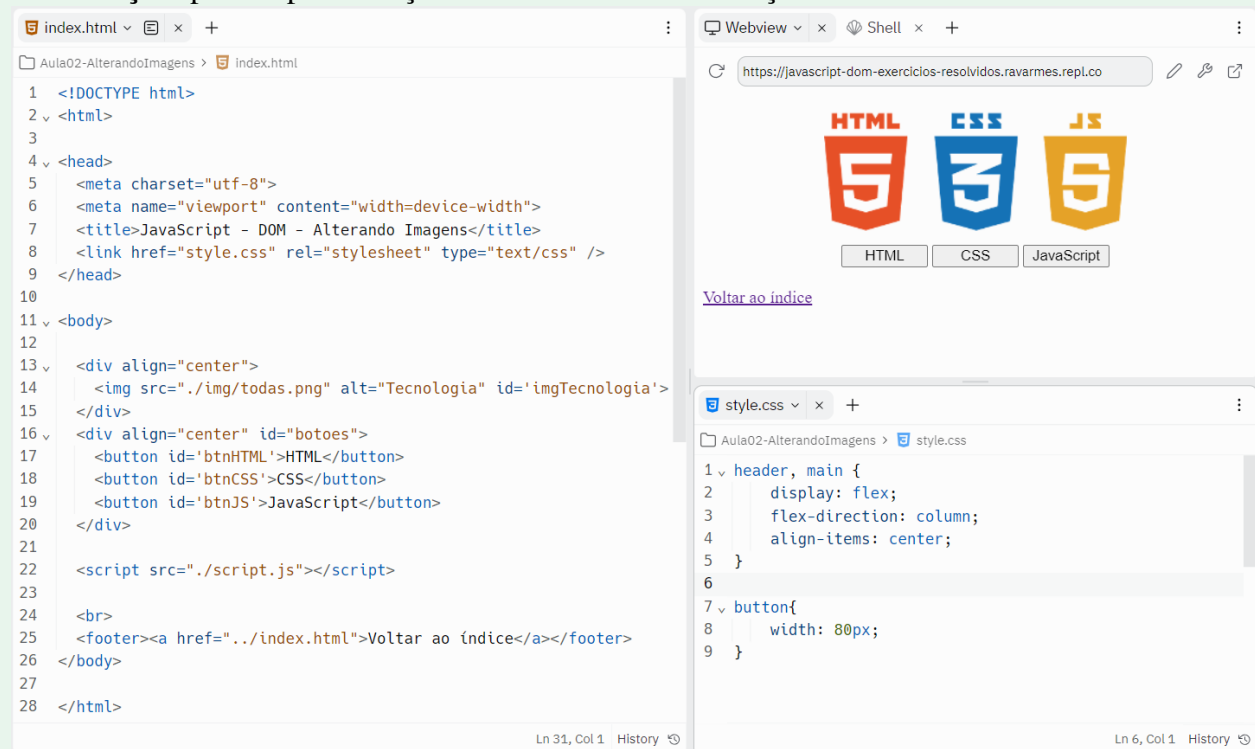
No botão **JS**:

- evento click: altere a imagem para js.png;

Na **imagem**:

- evento click: altere a imagem para todas.png.

Observação: para implementação dos eventos utilize a função ‘addEventListener’.



2.5 Gabarito dos Exercícios

Gabarito 2.1 — Eventos do mouse: Exercício 01.



Gabarito 2.2 — Capturando valores: Exercício 02.



Gabarito 2.3 — Adicionando elementos: Exercício 03.



Gabarito 2.4 — Alterando imagens: Exercício 04.



3. Trabalhando com formulários

Formulários são elementos fundamentais em praticamente todas as páginas da web interativas. Eles fornecem uma maneira para os usuários inserirem dados, enviarem informações e interagirem com os sites. Desde simples campos de login até complexos formulários de compra online, os formulários são a espinha dorsal da interação do usuário na web.

Este capítulo explorará em detalhes a arquitetura cliente/servidor que permite o funcionamento dos formulários na web. Vamos entender como as informações dos formulários são enviadas do cliente (geralmente um navegador web) para o servidor, onde são processadas e respondidas. A compreensão dessa dinâmica é crucial para desenvolvedores web, pois possibilita a criação de experiências de usuário dinâmicas e responsivas.

Em seguida, vamos analisar profundamente o que são formulários na perspectiva do desenvolvimento web. Vamos descobrir como os elementos HTML como `<input>`, `<select>` e `<textarea>` são utilizados para criar diferentes tipos de campos em um formulário. Compreender a estrutura dos formulários é o primeiro passo para criar interfaces intuitivas e eficazes para os usuários.

Além disso, examinaremos uma parte essencial do processo de formulários: a validação. Validar formulários é uma prática fundamental para garantir que os dados inseridos pelos usuários estejam corretos, seguros e no formato esperado. Vamos explorar técnicas modernas para validar formulários utilizando JavaScript DOM. Isso inclui a verificação em tempo real, fornecendo feedback imediato aos usuários, melhorando a usabilidade e reduzindo erros de entrada de dados.

Ao final deste capítulo, você terá uma compreensão abrangente sobre como os formulários funcionam na web, desde a arquitetura cliente/servidor até as melhores práticas para criar e validar formulários interativos e confiáveis. Vamos mergulhar nesse mundo crucial da interação do usuário, capacitando

você a criar experiências web envolventes e eficientes para seus usuários.

3.1 Arquitetura Cliente/Servidor

A arquitetura cliente/servidor é o pilar central da interação entre um usuário e uma aplicação web. Neste modelo, o cliente e o servidor são entidades distintas que se comunicam para fornecer funcionalidades e serviços aos usuários.

Cliente: O cliente é qualquer dispositivo ou software que acessa a web, como um navegador web em um computador, tablet ou smartphone. Ele solicita recursos e serviços do servidor e exibe as informações recebidas aos usuários. Os clientes também são responsáveis pela interação direta com os usuários, como a coleta de dados através de formulários.

Servidor: O servidor é um computador remoto que hospeda e fornece os recursos e serviços solicitados pelos clientes. Ele processa as requisições recebidas, executa a lógica da aplicação e retorna os resultados aos clientes. No contexto dos formulários, o servidor é responsável por validar e armazenar os dados enviados pelos usuários.

A interação começa quando um usuário preenche um formulário em um navegador. Quando o formulário é enviado, o navegador atua como cliente e envia as informações para o servidor. O servidor, por sua vez, processa os dados, realiza operações necessárias e retorna uma resposta ao cliente, que pode ser uma nova página, uma mensagem de sucesso ou outros tipos de feedback.

É importante entender essa arquitetura para criar formulários eficientes. A validação e o processamento dos dados geralmente ocorrem no servidor para garantir segurança e integridade. No entanto, técnicas como a validação do lado do cliente, utilizando JavaScript, também são usadas para melhorar a experiência do usuário, reduzindo a necessidade de interações adicionais com o servidor.

Compreender a dinâmica entre cliente e servidor é crucial para o desenvolvimento de formulários web robustos e seguros. Nos próximos tópicos, exploraremos detalhadamente como os formulários são estruturados em HTML e como a validação pode ser realizada tanto no lado do cliente quanto no lado do servidor para criar experiências de usuário confiáveis e interativas.

Vamos ilustrar na figura 3.1 esse processo com um exemplo de cadastro de usuário, onde o cliente representa o lado do usuário e o servidor é o componente central que processa as informações.

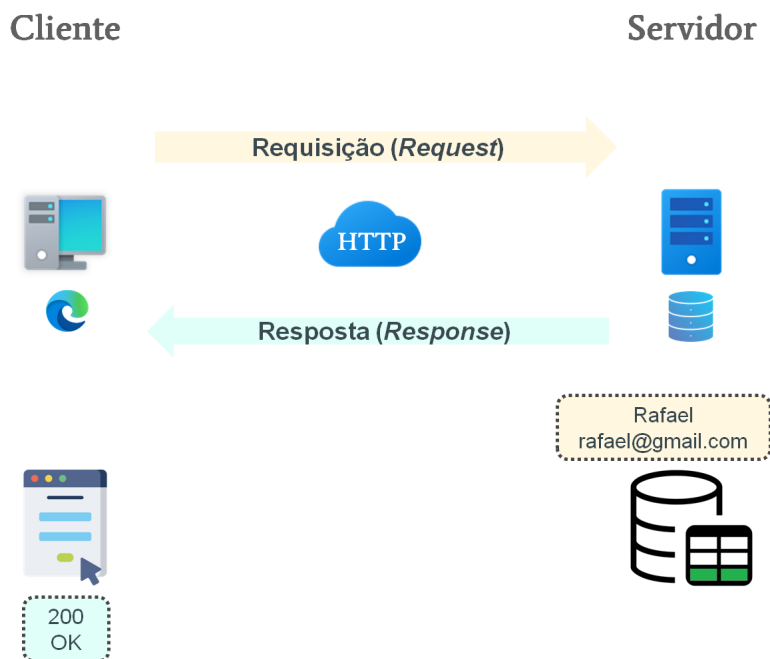


Figura 3.1: Interação Cliente/Servidor com um Formulário de Cadastro.

1. **Cliente:** Representado por um computador com um navegador aberto, neste caso, o Microsoft Edge;
2. **Servidor:** Representado por um gabinete que contém um banco de dados instalado. O banco de dados é o local onde os cadastros de novos usuários do site são armazenados;
3. **Protocolo de Comunicação:** A troca de informações entre cliente e servidor ocorre via protocolo HTTP (Hypertext Transfer Protocol). O HTTP é projetado para a transferência de conteúdo na Web;
4. **Requisição:** Quando um cliente envia dados para o servidor, ele o faz por meio de uma requisição. Neste exemplo, os dados de cadastro são enviados para o servidor;
5. **Processamento no Servidor:** O servidor processa os dados recebidos. Neste caso, um novo usuário é inserido no banco de dados;
6. **Resposta:** Após a inserção bem-sucedida, o servidor envia uma resposta ao cliente. No exemplo, um código de resposta HTTP 200 OK é enviado, indicando que a requisição foi bem-sucedida.

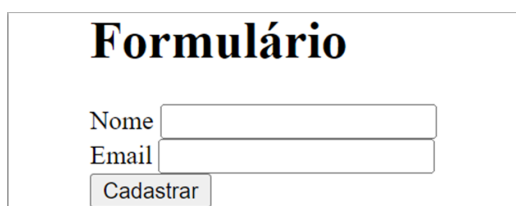
Nesta ilustração, focaremos especificamente na interação dos formulários. Considerando um formulário de cadastro com campos para nome e e-mail, os dados preenchidos pelo usuário são enviados ao servidor via uma requisição. O servidor, ao receber esses dados, os processa e fornece uma resposta ao cliente, indicando o sucesso da operação. Esta comunicação eficaz é essencial para criar formulários.

3.2 O que são Formulários?

Formulários HTML constituem um dos principais pontos de interação entre os usuários e os websites ou aplicativos web. Eles proporcionam aos usuários a capacidade de enviar dados para o website, permitindo uma variedade de interações, desde preencher uma pesquisa até criar uma conta de usuário. Ao preencher um formulário, os usuários inserem informações em campos designados, como nome, e-mail, senha, entre outros. Após o preenchimento, esses dados podem ser enviados para o servidor web associado ao site. No entanto, é importante observar que o processo de configuração de um servidor web capaz de receber e processar esses dados está além do escopo deste material.

A manipulação eficaz de formulários é uma habilidade fundamental para qualquer desenvolvedor web. Compreender como criar formulários intuitivos e eficazes, bem como entender o processo de envio de dados para o servidor, é essencial para oferecer aos usuários uma experiência fluida e interativa. Nos próximos tópicos, exploraremos como validar dados de formulários e garantir a segurança e integridade dos dados enviados pelos usuários.

A figura a seguir apresenta o design e a codificação de um formulário simples, contendo dois campos de entrada de dados e um botão para que o formulário seja enviado ao servidor.



(a) Design de formulário

```
1 <html>
2 <body>
3   <h1>Formulário</h1>
4   <form name="form1" action="/usuarios" method="get">
5     <div>
6       <label for="nome">Nome</label>
7       <input type="text" name="nome" />
8     </div>
9     <div>
10      <label for="email">Email</label>
11      <input type="email" name="email" />
12    </div>
13    <input type="submit" value="Cadastrar" />
14  </form>
15 </body>
16 </html>
```

(b) Código de formulário

Figura 3.2: Construção de formulários

Exploraremos, a seguir, com mais detalhes, o código para elaboração deste formulário simples utilizando HTML. Vamos explicar sobre os principais elementos dessa construção de formulários em HTML.

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4   <h1>Formulário</h1>
5   <form name="form1" action="/cadastro.html" method="post">
6     <div>
7       <label for="nome">Nome</label>
8       <input type="text" name="nome" />
9     </div>
10    <div>
11      <label for="email">Email</label>
12      <input type="email" name="email" />
13    </div>
14    <input type="submit" value="Cadastrar" />
15  </form>
16 </body>
17 </html>
```

Exemplo de Código 3.1: Sintaxe da construção de um formulários

Aqui está uma explicação detalhada dos elementos e atributos utilizados:

1. **Elemento form :** O elemento `<form>` é um container que define o formulário. Ele suporta atributos específicos como `action` e `method`, onde `action` define o local (URL) para onde os dados do formulário serão enviados, e `method` define o método HTTP a ser utilizado, neste caso, POST;
2. **Atributo name do Formulário:** O formulário tem um nome atribuído como "form1". Este nome é útil para identificar o formulário em scripts do lado do cliente ou para manipulação via JavaScript.
3. **Elementos label e input :** O elemento `<label>` é uma legenda associada a um campo específico, definido pelo atributo `for`. No exemplo, temos rótulos para os campos de nome e e-mail. Os elementos `<input>` são usados para coletar dados. O atributo `name` é crucial, pois é usado para identificar os campos quando os dados são enviados ao servidor.

4. **Campo de Envio e Atributo action:** O último `<input>` com o atributo `type="submit"` é um botão de envio. Quando clicado, este botão envia o formulário ao servidor. O atributo `action` define o destino para onde os dados do formulário serão enviados, neste caso, a URL `"/usuarios"` representa o recurso no servidor que processará os dados.

Na comunicação entre cliente e servidor através de formulários HTML, dois métodos principais são utilizados: GET e POST. O método GET é usado para solicitar dados do servidor, enviando os parâmetros do formulário como parte da URL. Isso significa que os dados do formulário são visíveis na barra de endereços do navegador, tornando-o adequado para solicitações simples e sem dados sensíveis. Por outro lado, o método POST envia os dados do formulário como parte do corpo da requisição HTTP, mantendo os dados ocultos da visão do usuário. O método POST é ideal para enviar dados sensíveis, como senhas, uma vez que os dados não são expostos na URL. Ao escolher entre GET e POST, os desenvolvedores precisam considerar a natureza dos dados a serem enviados e a segurança da transmissão. GET é adequado para solicitações de leitura, enquanto POST é preferível para operações que envolvem a criação, atualização ou exclusão de dados, garantindo assim uma transmissão segura e privada.

Para ilustrar as diferenças entre os métodos GET e POST na transmissão de dados de um formulário, considere o seguinte exemplo da figura 3.3 de formulário preenchido com dados de nome e e-mail.

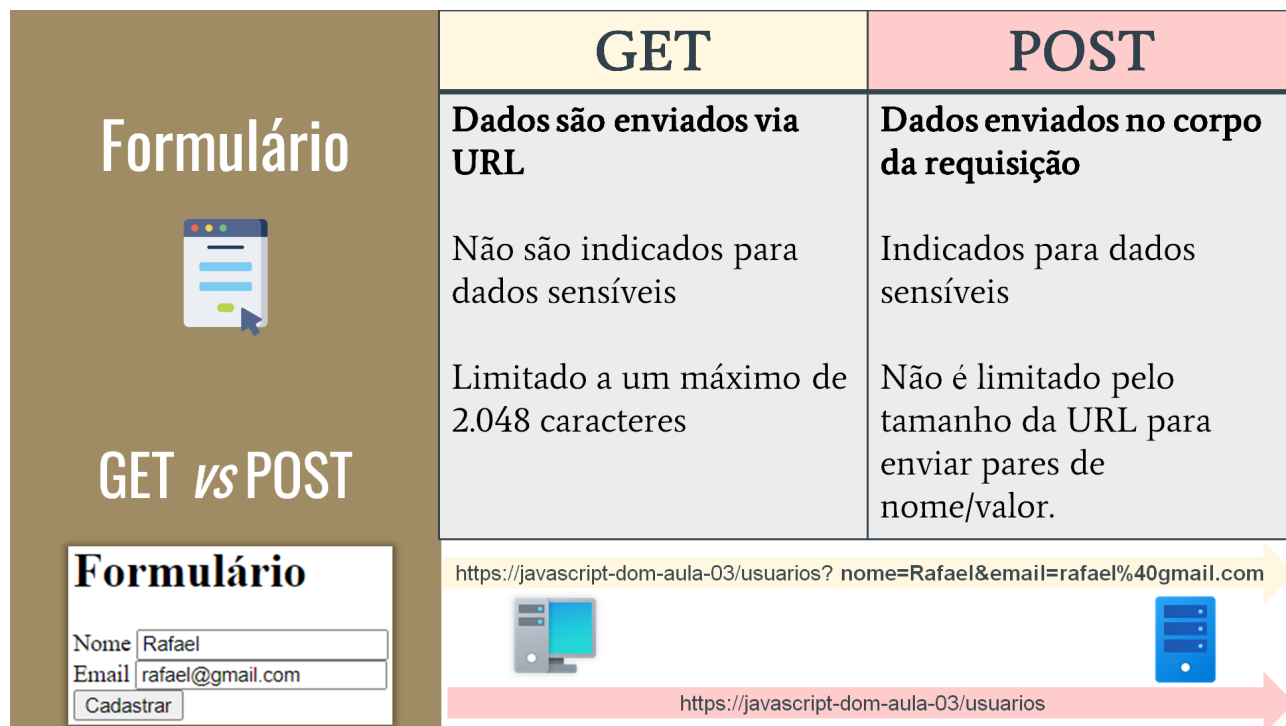


Figura 3.3: Diferença entre os métodos GET e POST.

Caso o atributo `method` do formulário tem sido definido como `GET`, os dados são enviados via URL no formato de chave-valor. Na requisição `GET`, representada pela seta amarela entre cliente e servidor, os dados para os campos nome e e-mail são incluídos. Por exemplo, o campo nome tem o valor **"Rafael"** e o campo e-mail tem o valor **"rafael@gmail.com"**¹. É crucial observar que o método `GET` não é recomendado para dados sensíveis, como senhas ou informações de cartão de crédito, devido à possibilidade de interceptação. Além disso, há um limite para o número máximo de caracteres em uma URL de requisição `HTTP`.

Por outro lado, se o atributo `method` do formulário for definido como `POST`, os dados são incluídos no corpo da requisição, ficando encapsulados e não visíveis na URL. Como mostra a seta vermelha de requisição entre cliente e servidor, os dados são transmitidos de forma mais segura e não estão sujeitos às limitações de tamanho da URL. Portanto, o método `POST` é indicado para o envio de dados sensíveis, garantindo uma transmissão segura e privada.

Ao considerarmos a prática comum no desenvolvimento web, o método `GET` é geralmente utilizado para solicitar listagens, enquanto o método `POST` é empregado para operações de cadastro. Para ilustrar esses cenários, apresentaremos, na figura 3.4, dois exemplos de formulários: um formulário de pesquisa que utiliza `GET` e um formulário de cadastro que utiliza `POST`.

GET	POST
<pre>1 <html> 2 <body> 3 <h1>Formulário</h1> 4 <form name="form1" action="/listagem.js" method="get"> 5 <div> 6 <label for="nome">Nome</label> 7 <input type="text" name="nome" /> 8 </div> 9 <input type="submit" value="Pesquisar" /> 10 </form> 11 </body> 12 </html></pre>	<pre>1 <html> 2 <body> 3 <h1>Formulário</h1> 4 <form name="form1" action="/cadastro.js" method="post"> 5 <div> 6 <label for="nome">Nome</label> 7 <input type="text" name="nome" /> 8 </div> 9 <div> 10 <label for="email">Email</label> 11 <input type="email" name="email" /> 12 </div> 13 <input type="submit" value="Cadastrar" /> 14 </form> 15 </body> 16 </html></pre>
<p>Formulário</p> <p>Nome <input type="text" value="Rafael"/></p> <p><input type="submit" value="Pesquisar"/></p>	<p>Formulário</p> <p>Nome <input type="text" value="Rafael"/></p> <p>Email <input type="text" value="rafael@gmail.com"/></p> <p><input type="submit" value="Cadastrar"/></p> <p>Usuário cadastrado!</p>

Figura 3.4: Cenários de utilização dos métodos `GET` e `POST`.

¹o símbolo @ é representado por %40 devido ao URLEncode.

Formulário de Pesquisa (GET):

Este é o código para o nosso formulário de pesquisa, cujo design em HTML pode ser observado na figura correspondente. Quando o usuário clica no botão **"Pesquisar"** e submete o formulário à página **"listagem.js"** do servidor, usando uma requisição com o método GET, o servidor poderia listar todos os usuários cadastrados que possuem a palavra **"Rafael"** em seus nomes, por exemplo.

Formulário de Cadastro (POST):

Por outro lado, apresentamos o código para um formulário de cadastro, cujo design HTML é mostrado na figura correspondente. Quando o usuário clica no botão **"Cadastrar"** e submete o formulário à página **"cadastro.js"** do servidor, utilizando uma requisição com o método POST, o servidor poderia inserir este novo usuário no banco de dados e exibir uma mensagem de sucesso, indicando que o cadastro foi efetuado com sucesso.

Estes exemplos ilustram a distinção prática entre os métodos GET e POST, sendo o GET apropriado para operações de leitura ou listagem, e o POST preferível para operações de criação ou alteração de dados, garantindo assim uma prática segura e eficaz no desenvolvimento web.

No próximo tópico, discutiremos como validar os dados submetidos por meio deste formulário para garantir integridade e segurança no processo de envio de dados ao servidor.

3.3 Como Validar Formulários?

A validação de formulários desempenha um papel crucial na garantia de que os usuários preencham os campos corretamente, assegurando que os dados enviados funcionem de maneira eficaz em nossas aplicações.

Exemplos de Mensagens de Validação:

- "Sua senha deve ter entre 8 e 30 caracteres e conter pelo menos uma letra maiúscula, um símbolo e um número."
- "Por favor, insira seu número de telefone no formato (xx) xxxx-xxxx."
- "Por favor, insira um endereço de e-mail válido."²
- "Este campo é obrigatório."³

²se a entrada não seguir o formato "email@email.com"

³se o campo não for preenchido

3.3.1 Tipos de Validações

Existem dois tipos principais de validações em formulários:

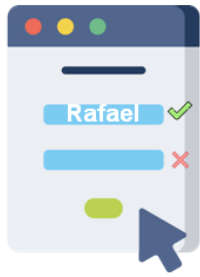
Validação do Lado do Cliente:

Esta é a validação que ocorre no navegador antes que os dados sejam enviados ao servidor. Ela pode ser realizada de duas maneiras:

- **Validação de Formulário Integrada (HTML5):** O HTML5 introduziu atributos para campos de entrada que especificam o tipo de dados esperado (como e-mail, número, etc.) e restrições (como comprimento mínimo ou máximo). O navegador valida automaticamente essas entradas com base nessas especificações.
- **Validação JavaScript:** Os desenvolvedores podem utilizar JavaScript para criar validações personalizadas, indo além das funcionalidades fornecidas pelo HTML5. Isso permite validar entradas de forma mais complexa, como validar um CPF ou verificar se duas senhas digitadas são iguais.

Validação do Lado do Servidor:

Esta é a validação que ocorre no servidor após o envio dos dados pelo cliente. É uma camada adicional de segurança para garantir que apenas dados válidos sejam aceitos. A validação do lado do servidor é essencial, uma vez que a validação do lado do cliente pode ser contornada, seja por usuários maliciosos ou por falhas de implementação no navegador do cliente.



(a) Validação do lado do cliente



Já existe um usuário com nome "Rafael"

(b) Validação do lado do servidor

**Figura 3.5:** Validação de formulários**Validação no Lado do Cliente:**

Considere a figura 3.5(a) que apresenta um cenário onde um formulário de cadastro é preenchido, mas o campo de e-mail é deixado em branco, resultando em um erro de validação. Nesse caso, os dados não são enviados ao servidor, já que a validação ocorre no navegador do cliente. Quando ocorre um erro de validação no lado do cliente, os dados não são submetidos ao servidor, proporcionando uma camada inicial de segurança.

Validação no Lado do Servidor:

Agora, imagine outra situação, apresentada na figura 3.5(b), onde tanto os campos de nome quanto de e-mail são preenchidos corretamente e passam pela validação do lado do cliente. Esses dados são enviados ao servidor. No entanto, há uma regra de negócio que proíbe a adição de usuários com o mesmo nome. Se o servidor detectar essa duplicidade, uma resposta com um código de erro (como o código 500) é enviada ao cliente, representado pelo navegador web. Esta resposta informa o problema ao usuário, permitindo que ele seja notificado do erro e tome as ações apropriadas.

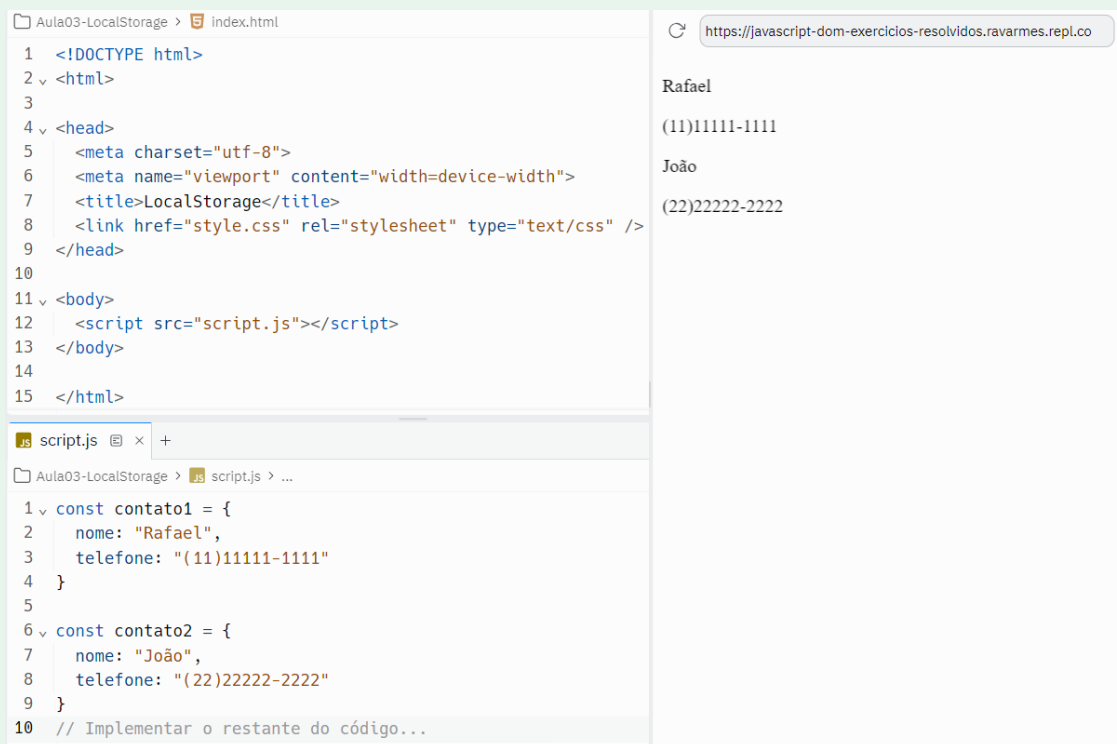
3.4 Exercícios Propostos

Exercício 3.1 — Trabalhando com formulários: Exercício 01. Elabore um algoritmo no arquivo `script.js`, utilizando linguagem de Programação JavaScript, para utilizar o recurso de `localStorage`. O `localStorage` é um recurso do navegador que permite armazenar dados de forma persistente no computador do usuário. É uma forma de armazenamento local que pode ser acessada e utilizada pelo JavaScript em páginas da web. Anteriormente os dados seriam armazenados via Cookies, mas hoje temos diversas opções.

Estes dados são armazenados no navegador da web do usuário. Especificamente, eles são armazenados no computador ou dispositivo em que o navegador está sendo executado. Cada navegador tem um local específico onde os dados do `localStorage` são mantidos, em outras palavras, quem gerencia todo o armazenamento é o navegador e nós só temos que nos preocupar com o Javascript.

O código elaborado no arquivo `script.js` deverá armazenar no `localStorage` um array de JSON's que representam contatos (com os atributos `nome` e `telefone`), por exemplo:

```
[
  { "nome": "Rafael", "telefone": "(11) 11111-1111" },
  { "nome": "João", "telefone": "(22) 22222-2222" }
]
```



Exercício 3.2 — Trabalhando com formulários: Exercício 02. Elabore um algoritmo no arquivo script.js, utilizando linguagem de Programação JavaScript, para listar, inserir e remover contatos, os quais serão exibidos em uma tabela na página codificada no arquivo index.html.

Os dados dos contatos deverão ser armazenados utilizando o recurso de localStorage (visto no exercício anterior), o qual nos permite armazenar dados de forma simples e sem expiração.

Na figura a seguir mostramos o resultado da exibição do site, após a elaboração do algoritmo no arquivo script.js.

Contatos

Nome

Telefone

Inserir

#	Nome	Telefone	Ação
0	João da Silva	(11)11111-1111	<div>Excluir</div>
1	Maria dos Santos	(22)22222-2222	<div>Excluir</div>

Estamos utilizando no código desta página index.html o Framework Bootstrap que fornece estruturas de CSS para a criação de sites e aplicações responsivas de forma rápida e simples.

3.5 Gabarito dos Exercícios

Gabarito 3.1 — Trabalhando com formulários: Exercício 01.



■

Gabarito 3.2 — Trabalhando com formulários: Exercício 02.



■



Parte II: Práticas

4	Selecionando elementos - Prática	49
5	Manipulando eventos - Prática	53
6	Trabalhando com formulários - Prática	61

4. Seleccionando elementos - Prática

Nesta seção vamos apresentar uma prática, desenvolvida no ambiente do Repl.it para trabalhar os conceitos sobre seleção de elementos utilizando JavaScript DOM.

Na figura 4.1 apresentamos o projeto da prática a ser elaborada. O objetivo principal do projeto é alterar a página HTML original, representada pela figura à esquerda, utilizando JavaScript DOM, para chegar no resultado apresentado na figura à direita.

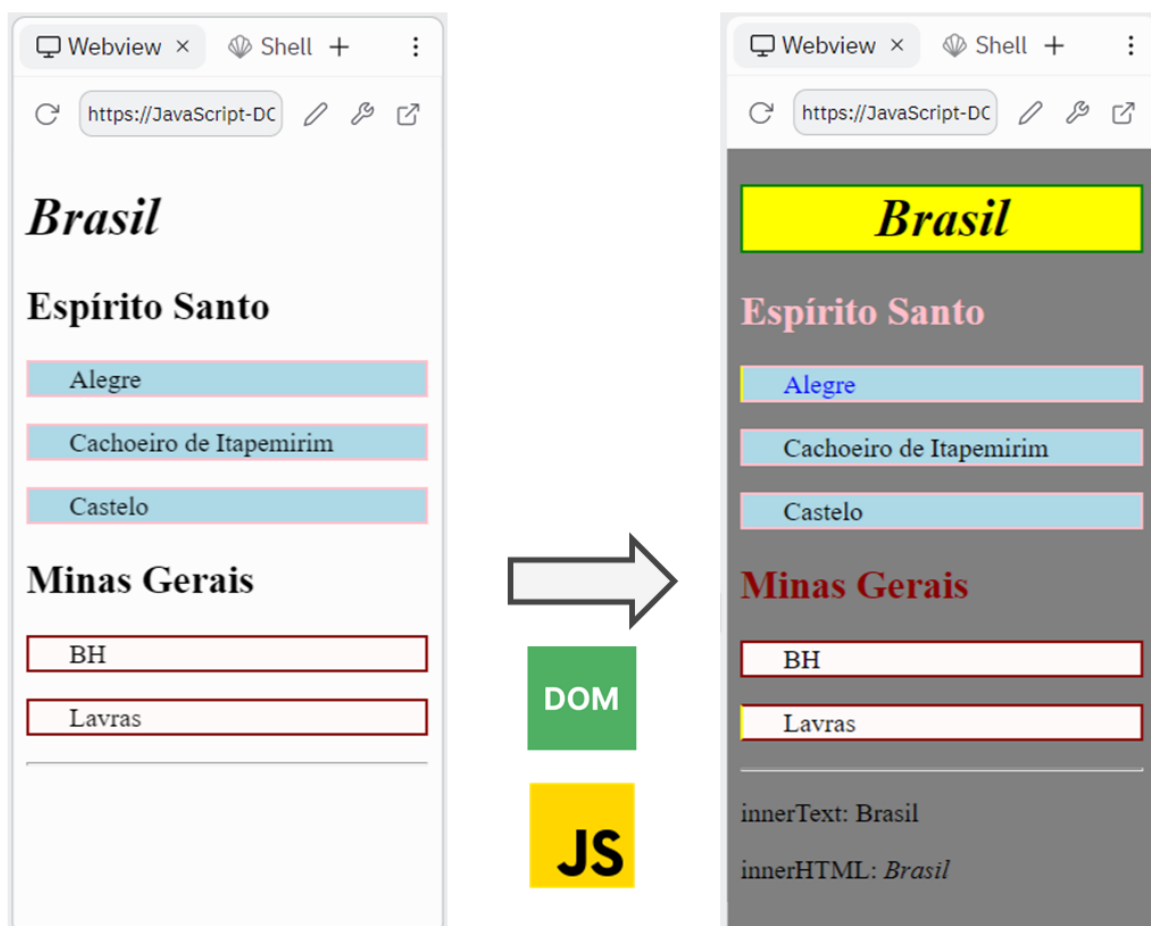


Figura 4.1: Seleccionando elementos - Prática

Capítulo 4. Selecionando elementos - Prática

A seguir serão apresentados os códigos para implementação dessa prática. Você poderá assistir ao vídeo com o passo a passo da implementação acessando o link:



```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <meta charset="utf-8">
6   <title> JavaScript - DOM - Selecionando Elementos </title>
7   <link href="style.css" rel="stylesheet" type="text/css" />
8 </head>
9
10 <body>
11   <h1 id="pais_brasil"> <i> Brasil </i> </h1>
12   <h2 id="uf_es"> Espírito Santo </h2>
13   <p name="cidade" class="cidade-es"> Alegre </p>
14   <p name="cidade" class="cidade-es"> Cachoeiro de Itapemirim </p>
15   <p name="cidade" class="cidade-es"> Castelo </p>
16   <h2 id="uf_mg"> Minas Gerais </h2>
17   <p name="cidade" class="cidade-mg"> BH </p>
18   <p name="cidade" class="cidade-mg"> Lavras </p>
19   <hr name="rodape">
20   <script src="script.js"></script>
21 </body>
22
23 </html>
```

Exemplo de Código 4.1: Selecionando elementos - Prática (index.html)

Capítulo 4. Seleccionando elementos - Prática

```
1 .cidade-es {
2     text-indent: 3ch;
3     border: 2px solid Pink;
4     background-color: LightBlue;
5 }
6
7 .cidade-mg {
8     text-indent: 3ch;
9     border: 2px solid DarkRed;
10    background-color: Snow;
11 }
```

Exemplo de Código 4.2: Seleccionando elementos - Prática (style.css)

```
1 var corpo = document.body;
2 corpo.style.background = "Gray";
3 document.title = "Novo título";
4
5 // getElementById
6 var h1_brasil = document.getElementById("pais_brasil");
7 h1_brasil.style.background = "Yellow";
8 document.write("<p> innerText: " + h1_brasil.innerText);
9 document.write("<p> innerHTML: " + h1_brasil.innerHTML);
10
11 // getElementsByTagName
12 var array_h2 = document.getElementsByTagName("h2");
13 var h2_es = array_h2[0];
14 var h2_mg = array_h2[1];
15 h2_es.style.color = "Pink";
16 h2_mg.style.color = "DarkRed";
17
18 // getElementsByName
19 var array_cidade = document.getElementsByName("cidade");
```

Capítulo 4. Seleccionando elementos - Prática

```
20 var cidade_0 = array_cidade[0];
21 cidade_0.style.borderLeftColor = "Yellow";
22
23 // getElementsByClassName
24 var array_cidade_mg = document.getElementsByClassName("cidade-mg");
25 var cidade_mg_1 = array_cidade_mg[1];
26 cidade_mg_1.style.borderLeftColor = "Yellow";
27
28 // querySelector
29 var pais = document.querySelector("#pais_brasil");
30 pais.style.border = "2px solid Green";
31 pais.style.textAlign = "center";
32
33 var cidade_es_0 = document.querySelector(".cidade-es");
34 cidade_es_0.style.color = "Blue";
```

Exemplo de Código 4.3: Seleccionando elementos - Prática (script.js)

5. Manipulando eventos - Prática

Nesta seção vamos apresentar quatro práticas, desenvolvidas no ambiente do Repl.it para trabalhar os conceitos sobre manipulação de eventos utilizando JavaScript DOM.

Os projetos vão abordar, basicamente, três finalidades:

- Primeiro, como **utilizar manipuladores in-line e listeners**;
- Segundo, como **delegar eventos** e;
- Terceiro, como **criar novos elementos** na página HTML.

Na figura 5.1 apresentamos a tela principal dos projetos a serem detalhados nesta seção.

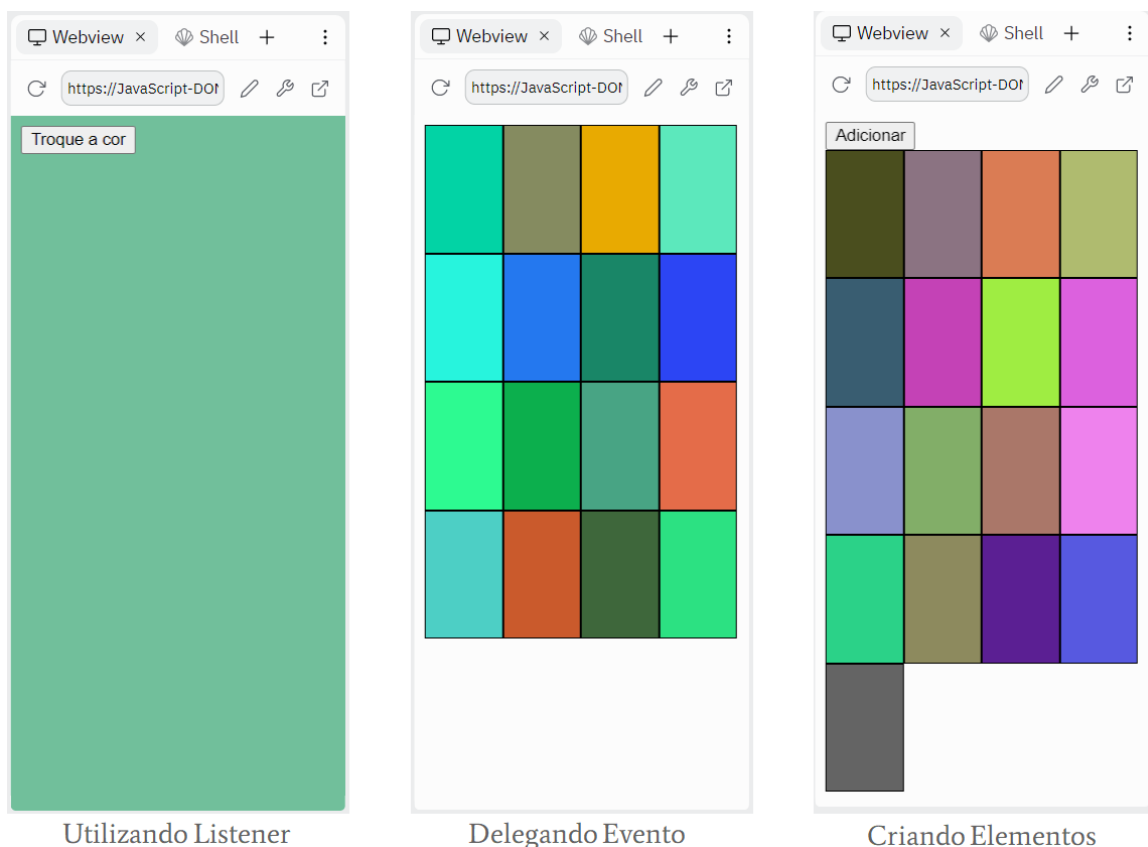


Figura 5.1: Manipulando eventos - Práticas

A seguir serão apresentados os códigos para implementação dessas práticas, bem como, disponibilizados os links para assistir aos vídeos com o passo a passo da implementação.

5.1 Utilizando manipuladores in-line

Neste primeiro projeto o clique com o mouse no botão, implica na troca da cor do background para uma cor aleatória. Esta mesma finalidade será elaborada em dois projeto diferentes utilizando dois tipos de manipuladores de eventos: manipuladores in-line e utilizando listener. Apenas pra reforçar a diferença entre os dois métodos. Nesta seção estamos apresentando a utilização de **manipuladores de eventos in-line**.



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>JavaScript - DOM - Aula 02 - Eventos</title>
5 </head>
6 <body>
7   <button onclick="trocarCor()">Troque a cor</button>
8   <script src="script.js"></script>
9 </body>
10 </html>
```

Exemplo de Código 5.1: Utilizando manipuladores in-line - Prática (index.html)

```
1 function random(number) {
2     return Math.floor(Math.random() * (number + 1));
3 }
4
5 function trocarCor() {
6     var rndCor = "rgb(" + random(255) + "," + random(255) + "," + random
7         (255) + ")";
8     document.body.style.backgroundColor = rndCor;
9 }
```

Exemplo de Código 5.2: Utilizando manipuladores in-line - Prática (script.js)

5.2 Utilizando listener

Este segundo projeto, conforme mencionado anteriormente, tem a mesma finalidade da prática anterior.

Todavia utilizando **listener** para manipular os eventos.



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>JavaScript - DOM - Aula 02 - Eventos</title>
5 </head>
6 <body>
7     <button>Troque a cor</button>
8     <script src="script.js"></script>
9 </body>
10 </html>
```

Exemplo de Código 5.3: Utilizando listener - Prática (index.html)

```
1 var btn = document.querySelector("button");
2 btn.addEventListener("click", trocarCor);
3
4 function random(number) {
```

```
5   return Math.floor(Math.random() * (number + 1));
6 }
7
8 function trocarCor() {
9   var rndCor = "rgb(" + random(255) + "," + random(255) + "," + random
    (255) + ")";
10  document.body.style.backgroundColor = rndCor;
11 }
```

Exemplo de Código 5.4: Utilizando listener - Prática (script.js)

5.3 Delegando eventos

Neste projeto, vamos criar uma div pai que será chamada de container. E dentro dessa div, vamos criar outras 16 div's. Quando o usuário clicar com o mouse em alguma das div's filhas, sua cor será alterada. A ideia de delegação de evento está no fato de que o listener será associado apenas à div pai. E, mesmo assim, vamos conseguir tratar as div's filhas de maneira específica.



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>JavaScript - DOM - Aula 02 - Eventos</title>
5   <link href="style.css" rel="stylesheet" type="text/css" />
6 </head>
7 <body>
8   <div id="container">
9     <div class="quadrado"></div>
10    <div class="quadrado"></div>
11    <div class="quadrado"></div>
12    <div class="quadrado"></div>
13    <div class="quadrado"></div>
14    <div class="quadrado"></div>
```



```
15 <div class="quadrado"></div>
16 <div class="quadrado"></div>
17 <div class="quadrado"></div>
18 <div class="quadrado"></div>
19 <div class="quadrado"></div>
20 <div class="quadrado"></div>
21 <div class="quadrado"></div>
22 <div class="quadrado"></div>
23 <div class="quadrado"></div>
24 <div class="quadrado"></div>
25 </div>
26 <script src="script.js"></script>
27 </body>
28 </html>
```

Exemplo de Código 5.5: Delegando eventos - Prática (index.html)

```
1 .quadrado {
2   height: 100px;
3   width: 24%;
4   float: left;
5   border-width: 1px;
6   border-style: solid;
7 }
```

Exemplo de Código 5.6: Delegando eventos - Prática (style.css)

```
1 const container = document.querySelector("#container");
2 container.addEventListener("click", trocarCor);
3
4 function random(number) {
5   return Math.floor(Math.random() * (number + 1));
6 }
7
8 function trocarCor(e) {
```

```
9   var rndCor = "rgb(" + random(255) + "," + random(255) + "," + random(255) + ")";
10  e.target.style.backgroundColor = rndCor;
11 }
```

Exemplo de Código 5.7: Delegando eventos - Prática (script.js)

5.4 Criando elementos

Por fim, vamos fazer uma pequena alteração no projeto anterior, criando um botão no início da página. O tratamento do evento de click neste botão implica na criação de novas div's no documento HTML. Ou seja, vamos utilizar o JavaScript para criar novos elementos na página.



```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>JavaScript - DOM - Aula 02 - Eventos</title>
5    <link href="style.css" rel="stylesheet" type="text/css" />
6  </head>
7  <body>
8    <button>Adicionar</button>
9    <div id="container">
10      <div class="quadrado"></div>
11      <div class="quadrado"></div>
12      <div class="quadrado"></div>
13      <div class="quadrado"></div>
14      <div class="quadrado"></div>
15      <div class="quadrado"></div>
16      <div class="quadrado"></div>
17      <div class="quadrado"></div>
18      <div class="quadrado"></div>
19      <div class="quadrado"></div>
```

```
20     <div class="quadrado"></div>
21     <div class="quadrado"></div>
22     <div class="quadrado"></div>
23     <div class="quadrado"></div>
24     <div class="quadrado"></div>
25     <div class="quadrado"></div>
26 </div>
27 <script src="script.js"></script>
28 </body>
29 </html>
```

Exemplo de Código 5.8: Criando elementos - Prática (index.html)

```
1 .quadrado {
2     height: 100px;
3     width: 24%;
4     float: left;
5     border-width: 1px;
6     border-style: solid;
7 }
```

Exemplo de Código 5.9: Criando elementos - Prática (style.css)

```
1 const container = document.querySelector("#container");
2 container.addEventListener("click", trocarCor);
3
4 const btn = document.querySelector("button");
5 btn.addEventListener("click", adicionar);
6
7 function random(number) {
8     return Math.floor(Math.random() * (number + 1));
9 }
10
11 function trocarCor(e) {
```

```
12  var rndCor = "rgb(" + random(255) + "," + random(255) + "," + random  
    (255) + ")";  
13  e.target.style.backgroundColor = rndCor;  
14  }  
15  
16  function adicionar() {  
17      var div = document.createElement("div");  
18      div.className = "quadrado";  
19      div.style.backgroundColor = "rgb(100,100,100)";  
20      container.appendChild(div);  
21  }
```

Exemplo de Código 5.10: Criando elementos - Prática (script.js)

6. Trabalhando com formulários - Prática

Nesta seção vamos apresentar três práticas, desenvolvidas no ambiente do Repl.it para trabalhar os conceitos sobre validação de formulários utilizando JavaScript DOM.

Na figura 6.1 apresentamos a tela inicial dos três projetos das práticas a elaboradas. O objetivo principal destes projetos é mostrar as diferenças de validações de formulários utilizando HTML, JavaScript e Bootstrap.

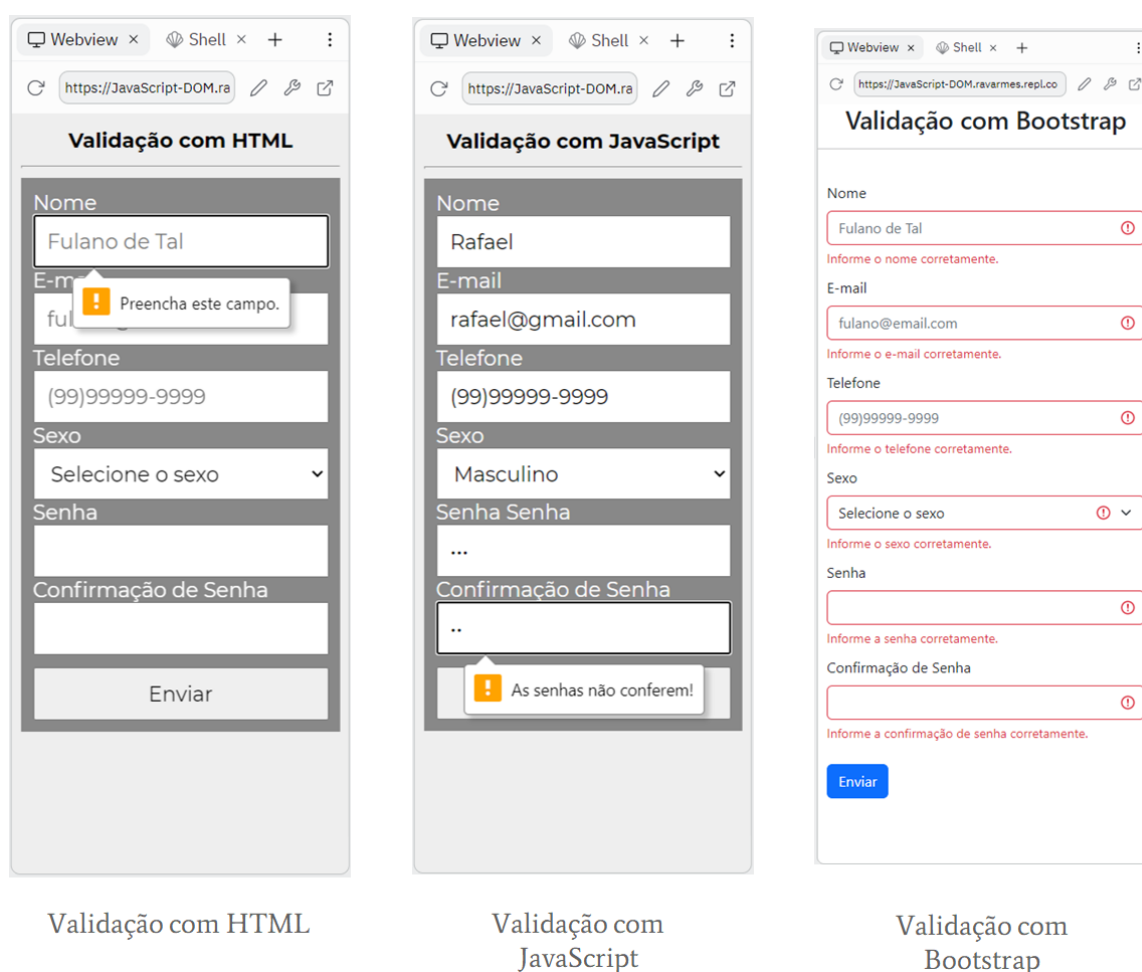


Figura 6.1: Trabalhando com formulários - Prática

6.1 Validação com HTML

Esta é uma breve demonstração da utilização do projeto, com validações realizadas exclusivamente por meio dos recursos do HTML5. Todos os campos são validados quanto ao preenchimento obrigatório. O campo de e-mail requer a formatação correta com o símbolo @, e o campo de telefone possui validação para um padrão específico de formatação da entrada.

No entanto, a verificação da igualdade entre os valores dos campos **"senha"** e **confirmação de senha"** não é realizada apenas com os recursos nativos do HTML5. Para implementar essa validação, uma rotina de tratamento onclick de um botão Submit em um formulário pode ser empregada. Essa rotina pode retornar false para evitar que o navegador envie o formulário. Essa prática é útil quando a entrada do usuário não passa na validação no lado do cliente.

Mesmo com valores diferentes nos campos de senha e confirmação de senha, a validação padrão do HTML5 é contornada. Portanto, é necessária uma implementação personalizada para garantir a igualdade dos valores nesses campos.



```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <meta charset="utf-8">
6   <meta name="viewport" content="width=device-width">
7   <title>JavaScript - DOM - Formulários - Validação com HTML</title>
8   <link href="style.css" rel="stylesheet" type="text/css" />
9 </head>
10
11 <body>
12
13   <h1>Validação com HTML</h1>
14   <hr>
15   <form name="form1" action="#" method="GET">
16     <label for="txtNome">Nome</label>
```

```
17     <input type="text" id="txtNome" name="nome" placeholder="Fulano de
18         Tal" required>
19 <label for="txtEmail">E-mail</label>
20 <input type="email" id="txtEmail" name="email" placeholder="
21     fulano@email.com" required>
22 <label for="txtTelefone">Telefone</label>
23 <input type="text" id="txtTelefone" name="telefone" placeholder="
24     (99) 99999-9999" pattern="\(\d{2}\)\d{5}-\d{4}"
25     required>
26 <label for="selSexo">Sexo</label>
27 <select id="selSexo" name="sexo" required>
28     <option value="" selected>Selecione o sexo</option>
29     <option value="M">Masculino</option>
30     <option value="F">Feminino</option>
31 </select>
32 <label for="txtSenha">Senha</label>
33 <input type="password" id="txtSenha" name="senha" required>
34 <label for="txtConfSenha">Confirmação de Senha</label>
35 <input type="password" id="txtConfSenha" name="confSenha" required>
36 <input type="submit" class="btn" value="Enviar">
37 </form>
38
39 <script src="script.js"></script>
40 </body>
41 </html>
```

Exemplo de Código 6.1: Validando formulários com HTML - Prática (index.html)

```
1 * {
2     box-sizing: border-box;
3     font-family: 'Montserrat';
4     font-size: 13pt;
```

```
5  }
6
7  body {
8      background-color: #eee;
9  }
10
11 h1 {
12     text-align: center;
13 }
14
15 form {
16     background-color: #888;
17     color: white;
18     padding: 10px;
19 }
20
21 input, select {
22     padding: 10px;
23     width: 100%;
24 }
25
26 .btn {
27     margin-top: 10px;
28 }
```

Exemplo de Código 6.2: Validando formulários com HTML - Prática (style.css)

6.2 Validação com JavaScript

Neste segundo projeto, as validações adotadas são as mesmas que no primeiro projeto, com uma ênfase especial na validação implementada em JavaScript para verificar a igualdade entre os valores dos campos de senha e confirmação de senha. É importante notar que o formulário só será enviado quando

esses campos estiverem preenchidos com valores iguais.

Em relação à ordem de chamadas de rotinas, é importante ressaltar que um elemento do documento ou outro objeto pode ter mais de uma rotina de tratamento de evento registrada para um tipo de evento específico. Quando ocorre um evento apropriado, o navegador deve chamar todas as rotinas de tratamento, seguindo as seguintes regras de ordem de chamada:

- As rotinas de tratamento registradas pela configuração de uma propriedade do objeto ou atributo HTML, se houver, são sempre chamadas primeiro.
- As rotinas de tratamento registradas com `addEventListener()` são chamadas na ordem em que foram registradas.

A validação em JavaScript implementada neste contexto verifica se os valores nos campos de senha e confirmação de senha são iguais antes de permitir o envio do formulário, assegurando assim que esses campos estejam devidamente correspondidos antes da submissão dos dados.



```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <meta charset="utf-8">
6   <meta name="viewport" content="width=device-width">
7   <title>JavaScript - DOM - Formulários - Validação com JavaScript</
   title>
8   <link href="style.css" rel="stylesheet" type="text/css" />
9 </head>
10
11 <body>
12
13   <h1>Validação com JavaScript</h1>
14   <hr>
15   <form name="form1" action="#" method="GET">
16     <label for="txtNome">Nome</label>
```

```

17     <input type="text" id="txtNome" name="nome" placeholder="Fulano de
        Tal" required>
18     <label for="txtEmail">E-mail</label>
19     <input type="email" id="txtEmail" name="email" placeholder="
        fulano@email.com" required>
20     <label for="txtTelefone">Telefone</label>
21     <input type="text" id="txtTelefone" name="telefone" placeholder="
        (99) 99999-9999" pattern="\(\d{2}\)\d{5}-\d{4}"
22         required>
23     <label for="selSexo">Sexo</label>
24     <select id="selSexo" name="sexo" required>
25         <option value="" selected>Selecione o sexo</option>
26         <option value="M">Masculino</option>
27         <option value="F">Feminino</option>
28     </select>
29     <label for="txtSenha">Senha Senha</label>
30     <input type="password" id="txtSenha" name="senha" required>
31     <label for="txtConfSenha">Confirmação de Senha</label>
32     <input type="password" id="txtConfSenha" name="confSenha" required>
33     <input type="submit" class="btn" value="Enviar">
34 </form>
35
36     <script src="script.js"></script>
37 </body>
38
39 </html>

```

Exemplo de Código 6.3: Validando formulários com JavaScript - Prática (index.html)

```

1 * {
2     box-sizing: border-box;
3     font-family: 'Montserrat';
4     font-size: 13pt;

```

```
5 }  
6  
7 body {  
8   background-color: #eee;  
9 }  
10  
11 h1 {  
12   text-align: center;  
13 }  
14  
15 form {  
16   background-color: #888;  
17   color: white;  
18   padding: 10px;  
19 }  
20  
21 input, select {  
22   padding: 10px;  
23   width: 100%;  
24 }  
25  
26 .btn {  
27   margin-top: 10px;  
28 }
```

Exemplo de Código 6.4: Validando formulários com JavaScript - Prática (style.css)

```
1 const txtSenha = document.getElementById("txtSenha");  
2 const txtConfSenha = document.getElementById("txtConfSenha");  
3  
4 txtSenha.addEventListener("input", validarSenhas);  
5 txtConfSenha.addEventListener("input", validarSenhas);  
6
```

```
7 function validarSenhas() {  
8     if (txtSenha.value != txtConfSenha.value) {  
9         txtConfSenha.setCustomValidity("As senhas não conferem!");  
10    } else {  
11        txtConfSenha.setCustomValidity("");  
12    }  
13 }
```

Exemplo de Código 6.5: Validando formulários com JavaScript - Prática (script.js)

6.3 Validação com Bootstrap

Por fim, a validação JavaScript aplicada em conjunto com a utilização do framework Bootstrap. A validação utilizada JavaScript – A diferença em relação à validação apresentada anteriormente está na utilização do framework Bootstrap para melhorar a experiência do usuário à partir de um site amigável e responsivo.



```
1 <!DOCTYPE html>  
2 <html>  
3  
4 <head>  
5     <meta charset="utf-8">  
6     <meta name="viewport" content="width=device-width">  
7  
8     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/  
9         css/bootstrap.min.css" rel="stylesheet"  
10         integrity="sha384-GLh1TQ8iRABdZLl603oVMWSktQOp6b7In1Zl3/  
11         Jr59b6EGGoI1aFkw7cmDA6j6gD" crossorigin="anonymous">  
12  
13     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/  
14         js/bootstrap.bundle.min.js"  
15         integrity="sha384-w76AqPfDkMBDXo30jS1Sgez6pr3x5M1Q1ZAGC+nuZB+  
16         EYdgRZgiwxhTBTkF7CXvN" crossorigin="anonymous">
```

```
12     defer></script>
13
14     <script src="script.js" defer></script>
15     <title>JavaScript - DOM - Formulários Bootstrap</title>
16 </head>
17
18 <body class="p-3 m-0 border-0">
19
20     <h1 class="text-center">Validação com Bootstrap</h1>
21     <hr>
22
23     <form name="form1" action="#" method="GET" class="row g-3 needs-
24         validation" novalidate>
25         <div class="form-row">
26             <label for="txtNome" class="form-label">Nome</label>
27             <input type="text" id="txtNome" name="nome" placeholder="Fulano
28                 de Tal" class="form-control" required>
29             <div class="invalid-feedback">Informe o nome corretamente.</div>
30         </div>
31         <div class="form-row">
32             <label for="txtEmail" class="form-label">E-mail</label>
33             <input type="email" id="txtEmail" name="email" placeholder="
34                 fulano@email.com" class="form-control" required>
35             <div class="invalid-feedback">Informe o e-mail corretamente.</div>
36         </div>
37         <div class="form-row">
38             <label for="txtTelefone" class="form-label">Telefone</label>
39             <input type="text" id="txtTelefone" name="telefone" placeholder="
40                 (99) 99999-9999" pattern="\(\d{2}\)\d{5}-\d{4}"
41                 class="form-control" required>
```

```
38     <div class="invalid-feedback">Informe o telefone corretamente.</div>
39   </div>
40   <div class="form-row">
41     <label for="selSexo" class="form-label">Sexo</label>
42     <select id="selSexo" name="sexo" class="form-select" required>
43       <option value="" selected>Selecione o sexo</option>
44       <option value="M">Masculino</option>
45       <option value="F">Feminino</option>
46     </select>
47     <div class="invalid-feedback">Informe o sexo corretamente.</div>
48   </div>
49   <div class="form-row">
50     <label for="txtSenha" class="form-label">Senha</label>
51     <input type="password" id="txtSenha" name="senha" class="form-control" required>
52     <div class="invalid-feedback">Informe a senha corretamente.</div>
53   </div>
54   <div class="form-row">
55     <label for="txtConfSenha" class="form-label">Confirmação de Senha
56       </label>
57     <input type="password" id="txtConfSenha" name="confSenha" class="form-control" required>
58     <div class="invalid-feedback">Informe a confirmação de senha
59       corretamente.</div>
60   </div>
61   <br>
62   <input type="submit" class="btn btn-primary me-md-2" value="Enviar">
```

```
63 </body>
64
65 </html>
```

Exemplo de Código 6.6: Validando formulários com Bootstrap - Prática (index.html)

```
1  (() => {
2    'use strict'
3
4    const forms = document.querySelectorAll('.needs-validation')
5
6    Array.from(forms).forEach(form => {
7      form.addEventListener('submit', event => {
8        if (!form.checkValidity()) {
9          event.preventDefault()
10         event.stopPropagation()
11       }
12
13       form.classList.add('was-validated')
14     }, false)
15   })
16 })()
17
18 const txtSenha = document.getElementById("txtSenha");
19 const txtConfSenha = document.getElementById("txtConfSenha");
20
21 txtSenha.addEventListener('input', validarSenhas);
22 txtConfSenha.addEventListener('input', validarSenhas);
23
24 function validarSenhas() {
25   if (txtSenha.value !== txtConfSenha.value) {
26     txtConfSenha.setCustomValidity("As senhas não conferem!");
27   } else {
```

```
28     txtConfSenha.setCustomValidity("");  
29 }  
30 }
```

Exemplo de Código 6.7: Validando formulários com Bootstrap - Prática (script.js)