# Tema 14 – JavaScript HTML DOM – Parte 04

## Introdução

Olá, futuras mentes desenvolvedoras, sejam bem-vindos do curso de Frontend do Instituto da Oportunidade Social. Nessa aula, você aprenderá a trabalhar com entrada de dados no formulário com o JS manipulando o Document Object Model (Modelo de objeto de documentos).

## Criação do projeto inicial

Siga os passos para criar o projeto:

1. Abra o VS Code e escolha um diretório de trabalho para o seu projeto.
2. Crie um diretório para seu projeto com o nome representativo, por exemplo, 14\_JS\_DOM\_Parte\_04.
3. Crie um arquivo dentro do diretório do projeto com o nome index.html.
4. Insira o seguinte código no seu arquivo index.html.

|  |  |
| --- | --- |
| 1 | <!DOCTYPE html>  <html lang="pt-br">      <head>          <meta charset="UTF-8" />          <link rel="shortcut icon" href="#" />          <link rel="stylesheet" href="./css/style.css" />          <title>JavaScript com HTML DOM - Parte 04</title>      </head>      <body>          <header>              <h1>Agendamento de Clientes</h1>          </header>          <section class="container">              <form id="my-form">                  <h1>Adicionar Usuário</h1>                  <div class="msg"></div>                  <div>                      <label for="name">Nome:</label>                      <input type="text" id="name" />                  </div>                  <div>                      <label for="email">E-mail:</label>                      <input type="text" id="email" />                  </div>                  <div class="msg\_email"></div>                  <div>                      <label for="horario">Horário:</label>                      <select name="hours" id="horario">                          <option value="--">---</option>                          <option value="9:00h">9:00</option>                          <option value="10:00h">10:00</option>                          <option value="11:00h">11:00</option>                          <option value="12:00h">12:00</option>                      </select>                  </div>                  <input class="btn" type="submit" value="Enviar" />              </form>              <ul id="users"></ul>          </section>          <script src="./js/main.js"></script>      </body>  </html> |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |
| 12 |
| 13 |
| 14 |
| 15 |
| 16 |
| 17 |
| 18 |
| 19 |
| 20 |
| 21 |
| 22 |
| 23 |
| 24 |
| 25 |
| 26 |
| 27 |
| 28 |
| 29 |
| 30 |
| 31 |
| 32 |
| 33 |
| 34 |
| 35 |
| 36 |
| 37 |
| 38 |
| 39 |
| 40 |
| 41 |
| 42 |

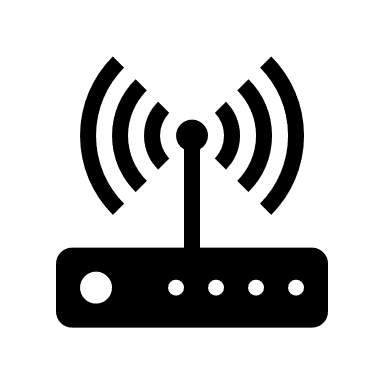
1. Esse código mostra na linha 6 a marcação <link> com o arquivo CSS e no atributo **href** mostra que ele está em uma pasta chamada **css**. Portanto, vamos criar o diretório **css** e o arquivo style.css dentro desse projeto com o seguinte código.

|  |
| --- |
| \* {      margin: 0;      padding: 0;      box-sizing: border-box;  }  body {      font-family: Arial, Helvetica, sans-serif;      line-height: 1.6;  }  ul {      list-style: none;  }  ul li {      padding: 5px;      background: #f4f4f4;      margin: 5px 0;  }  header {      background: #f4f4f4;      padding: 1rem;      text-align: center;  }  .container {      margin: auto;      width: 500px;      overflow: auto;      padding: 3rem 2rem;  }  #my-form {      padding: 2rem;      background: #f4f4f4;  }  #my-form label {      display: block;  }  #my-form input[type='text'] {      width: 100%;      padding: 8px;      margin-bottom: 10px;      border-radius: 5px;      border: 1px solid #ccc;  }  #my-form select {      width: 20%;      padding: 8px;      margin-bottom: 10px;      border-radius: 5px;      border: 1px solid #ccc;  }  .btn {      display: block;      width: 100%;      padding: 10px 15px;      border: 0;      background: #333;      color: #fff;      border-radius: 5px;      margin: 5px 0;  }  .btn:hover {      background: #444;  }  .bg-dark {      background: #333;      color: #fff;  }  .error {      font-size: 0.8rem;      background: #fff;      color: red;  } |

1. Esse código mostra também a marcação <script> sem nenhum código JavaScript entre a abertura e o fechamento da tag, apenas o atributo **src** com o valor main.js. Isso significa que o código JavaScript está em um arquivo externo. Portanto, temos que criar esse novo arquivo **main.js** dentro do diretório do projeto.
2. Vamos deixar o arquivo main.js vazio e à medida que vamos aprendendo coisas novas iremos completar a implementação do código JavaScript.
3. Fique atento com a estrutura usada nesse projeto.

Text

Description automatically generated with medium confidence

1. Abra o arquivo index.html, clique no botão **Go Live** da extensão **Live Server**. A página inicial mostrada será:

Graphical user interface, application

Description automatically generated

## Colocando código em JS para trabalhar com o formulário

A aplicação que desejamos fazer é a seguinte: Esse é um formulário de agendamento de clientes. O agendamento será mostrado apenas na interface do usuário, não vamos salvar em nenhum servidor de backend, portanto se a sua recarregar os dados serão perdidos. Então, a nossa agenda de clientes funcionará da seguinte forma:

* Você irá preencher o Nome e o E-mail, observe que o campo está com o atributo do HTML **required** (linha 19 do código HTML).
* Assim que os campos estiverem corretamente preenchidos e você clicar no botão submete, as informações serão exibidas na lista <ul> disponível na linha 27 do código HTML.

Vamos implementar nosso código em JS para manipular as informações e os elementos do formulário. Siga os passos para completar o código do JavaScript

1. Primeiro, vamos brincar de alterar a cor do botão para enviar os dados no formulário. No arquivo **main.js**, insira o seguinte código.

|  |  |
| --- | --- |
| 1 | // Método querySelector  const btn = document.querySelector('.btn');  btn.style.background = 'red'; |
| 2 |
| 3 |

1. Veja o resultado apresentado no navegador.

Graphical user interface, application

Description automatically generated

Observe que o método **querySelector** busca pelo elemento com a classe **.btn** (linha 2), que é encontrada no botão na linha 35 do código HTML. Então, esse elemento é retornado e, assim, é possível alterarmos a cor de fundo do elemento (linha 3).

1. Agora vamos buscar os elementos do formulário e salvá-los em variáveis para trabalharmos com eles.

|  |  |
| --- | --- |
| 5 | // Busca pelos elementos do Formulário  const myForm = document.querySelector('#my-form');  const nameInput = document.querySelector('#name');  const emailInput = document.querySelector('#email');  const horario = document.querySelector('#horario');  const msg = document.querySelector('.msg');  const msg\_email = document.querySelector('.msg\_email');  const userList = document.querySelector('#users'); |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |
| 12 |

Na linha 6, a instrução busca pelo elemento com a ID igual a #my-form através do método querySelector, que retorna o elemento <form> do formulário.

Na linha 7, a instrução busca pelo elemento com a ID igual a #name também através do método querySelector, que retorna o primeiro elemento <input> dentro do formulário.

Na linha 8, a instrução busca pelo elemento com a ID igual a #email também através do método querySelector, que retorna o segundo elemento <input> dentro do formulário.

Na linha 9, a instrução busca pelo elemento com a ID igual a #horario também através do método querySelector, que retorna o elemento <selector> dentro do formulário.

Na linha 10, a instrução busca pelo elemento com a classe igual a .msg também através do método querySelector, que retorna o elemento <div> logo abaixo do <input> name na linha 16 do HTML. Esse elemento vai ser usado para exibir uma mensagem caso o valor do campo name esteja vazio.

Na linha 11, a instrução busca pelo elemento com a classe igual a .msg\_email também através do método querySelector, que retorna o elemento <div> logo abaixo do <input> email na linha 25 do HTML. Esse elemento vai ser usado para exibir uma mensagem caso o usuário insira um email inválido.

Na linha 12, a instrução busca pelo elemento com a ID igual a #users também através do método querySelector, que retorna o elemento <ul> fora do formulário e dentro do documento HTML. Esse elemento será usado para inserirmos as novas informações do agendamento.

## Método addEventListener

O método **addEventListener** “vigia” um evento de um elemento específico. Esse evento pode ser o clique de um botão, a submissão de um formulário, etc. No nosso exemplo, vamos utilizar esse método para “vigiar” o evento de submissão do formulário.

### Vamos praticar

1. Continuando a implementação do projeto, no arquivo **main.js**, insira o seguinte código.

|  |  |
| --- | --- |
| 14 | // Método addEventListener  myForm.addEventListener('submit', onSubmit); |
| 15 |

Nesse código mostrado acima, o método está registrado para o evento **submit** do elemento myForm, que é o nosso formulário. Observe que esse elemento chama a função **onSubmit**, que vamos implementar a seguir.

1. No arquivo **main.js**, insira o seguinte código.

|  |  |
| --- | --- |
| 17 | // Função onSubmit  function onSubmit(e) {      e.preventDefault();      // console.log(nameInput.value);      if (nameInput.value === '' || emailInput.value === '') {          // alert('Por favor, preencha os dados.');          msg.classList.add('error');          msg.innerHTML = 'Por favor, preencha os dados.';          setTimeout(() => msg.remove(), 3000);      } else {          // console.log('sucess');          const li = document.createElement('li');          li.appendChild(              document.createTextNode(                  `${nameInput.value} : ${emailInput.value} : ${horario.value}`              )          );          userList.appendChild(li);          // Limpa o formulário          nameInput.value = '';          emailInput.value = '';          horario.getElementsByTagName('option')[0].selected = 'selected';  nameInput.focus(); //Coloca o foco no elmento      }  } |
| 18 |
| 19 |
| 20 |
| 21 |
| 22 |
| 23 |
| 24 |
| 25 |
| 26 |
| 27 |
| 28 |
| 29 |
| 30 |
| 31 |
| 32 |
| 33 |
| 34 |
| 35 |
| 36 |
| 37 |
| 38 |
| 39 |
| 40 |
| 41 |

1. Veja o resultado apresentado no navegador.

Graphical user interface, application

Description automatically generated

A instrução **e.preventDefault()** na linha 19 é para evitar que a página seja recarregada por causa da ação de submeter o formulário, pois essa é uma ação padrão do HTML quando estamos trabalhando com formulários.

Na linha 21, testamos se os campos do nome e do e-mail estão vazios, caso um deles esteja vazio a mensagem no **<div class="msg"></div>** irá aparecer por 3 segundos (**setTimeout(() => msg.remove(), 3000),** na linha 25)

A propriedade **classList.add** na linha 23 adiciona uma classe no elemento específico, que no caso é o elemento **<div>** com a classe **.msg**.

Caso os campos estejam preenchidos o **else** (na linha 26) será executado. Na linha 28, a instrução cria um elemento **<li>** para inserir um item na lista. A inserção do item na lista é feita pelas instruções nas linhas 29 até 34.

O método appendChild adiciona um nó ao final da lista de elementos filhos de um elemento pai especificado. No nosso caso, esse método vai inserido itens ao final da lista.

Por fim, as instruções nas linhas 36 até 38 limpam o conteúdo do formulário e coloca o cursor no campo do nome (**nameInput.focus()**)

## Validando e-mail com RegEx

O **Regex** (regular expression) é uma sequência de caracteres que especifica um padrão de busca. Normalmente, esses padrões são usados por algoritmos de busca de strings para realizar operações de "localizar" ou de "localizar e substituir" e também para validação de entrada de dados. Você pode testar qualquer tipo de expressão regular no site <https://regex101.com/>.

### Vamos praticar

1. Continuando a implementação do projeto, no arquivo **main.js**, insira o seguinte código.

|  |  |
| --- | --- |
| 43 | // Validando e-mail  emailInput.addEventListener('change', (e) => {      let padrao = new RegExp(/.\*@.\*\..\*/i);      if (!padrao.test(emailInput.value)) {          // alert('Por favor, insira um e-mail válido.');          msg\_email.classList.add('error');          msg\_email.innerHTML = 'Por favor, insira um e-mail válido.';          setTimeout(() => msg.remove(), 3000);      }  }); |
| 44 |
| 45 |
| 46 |
| 47 |
| 48 |
| 49 |
| 50 |
| 51 |
| 53 |

1. Veja o resultado apresentado no navegador.

Graphical user interface, application

Description automatically generated

A instrução na linha 43 vincular a execução de um Arrow Function ao evento **change** do campo de e-mail. Quando você terminar de digitar e sai do campo, a função de verificação do e-mail é executada.

A expressão testada é /.\*@.\*\..\*/i na linha 43, que verifica se tem um símbolo de arroba (@) e um ponto final no e-mail. Não é a melhor das validações de e-mail, mas funciona para o nosso caso. Se você quiser pode testar a seguinte expressão:

/^[a-z0-9.]+@[a-z0-9]+\.[a-z]+\.([a-z]+)?$/i

Graphical user interface, text, application

Description automatically generated

Caso o e-mail esteja inválido de acordo com o teste lógico **!padrao.test(emailInput.value)**, a mensagem de erro irá aparecer (linhas 47 até 50).

## Para aprender mais

Procure sempre aprender e estudar mais. Seguem alguns links para você estudar e aprender mais:

addEventListener:

<https://developer.mozilla.org/pt-BR/docs/Web/API/EventTarget/addEventListener>

<https://www.w3schools.com/jsref/met_element_addeventlistener.asp>

appendChild:

<https://developer.mozilla.org/pt-BR/docs/Web/API/Node/appendChild>

<https://www.w3schools.com/jsref/met_node_appendchild.asp>

Regex:

<https://regex101.com/>

<https://en.wikipedia.org/wiki/Regular_expression>

Javascript e Formulários:

<https://www.youtube.com/watch?v=OR8ySydmqLQ>

<https://www.youtube.com/watch?v=evS7nVIbsw4>

<https://www.youtube.com/watch?v=qMxhzUh2gUc>

<https://www.youtube.com/watch?v=NoZOqtK6ecI>