**JavaScript\_Programando na linguagem da web – Micro-certificação Tecnologia WEB - ALURA**

**Aula 01 - Consolidando o seu conhecimento**

## O primeiro arquivo .js

Para dar início ao nosso projeto, vamos começar criando nosso primeiro arquivo Javascript, o principal.js, que vai conter o código inicial que faremos neste primeiro capítulo. Para **manter uma boa organização de código** crie o arquivo principal.js dentro da pasta **/js**. Sua estrutra de pastas deve ficar assim:

introducao-javascript

├── css

│ ├── index.css

│ └── reset.css

├── js

│ └── principal.js

├── img

├── favicon.ico

└── index.html

## Importando o arquivo

Claro que após criar um novo arquivo Javascript, não pode esquecer de importá-lo ao final da tag <body> em nosso HTML, utilizando a tag <script>:

// Restante do código HTML

</section>

</main>

<!-- Importação dos Javascripts AQUI -->

<script src="js/principal.js"></script>

</body>

</html>

Repare que colocamos no atributo src o caminho relativo para nosso arquivo principal.js, ou seja a partir da pasta atual, vamos descer para a pasta js/ e em seguida linkar com arquivo.

## Trocando o título da página

Para dar início aos nossos conhecimentos com o Javascript, vamos trocar o título da nossa página de Aparecida Nutrição para Aparecida Nutricionista, para praticar. Em nosso principal.js vamos:

1- O primeiro passo é selecionar o elemento da página que queremos alterar, neste caso o :

<h1>Aparecida Nutrição</h1>

Sabemos que é possível selecionar um elemento através de sua tag, e como só temos este h1 na página poderíamos facilmente utilizá-lo, porém também vimos que não é uma boa prática utilizar um seletor de tag quando estamos selecionando um elemento do HTML, pois este h1 poderia facilmente ser trocado por um h2 e nosso código quebraria.

Vamos adicionar uma classe no h1 para que a seleção do elemento sempre seja feita atráves desta classe:

<h1 class="titulo">Aparecida Nutrição</h1>

2- Agora que o h1 já tem uma classe, podemos utilizar a **função seletora** do Javascript que é responsável de trazer um elemento do mundo do HTML para o mundo do Javascript que é a querySelector(). Sabemos que queremos buscar na nossa página pelo h1, então vamos chamar a querySelector() na variável que representa a nossa página para o Javascript , o document:

document.querySelector(".titulo");

Como queremos manipular este elemento selecionado, vamos colocá-lo em uma variável de nome semântico para poder ter acesso a ele depois:

var titulo = document.querySelector(".titulo");

3- Por último, como queremos trocar o texto do h1, vamos acessar a propriedade textContent do elemento selecionado, que é a propriedade que tem o contéudo de texto daquela tag. Como queremos **alterar** o contéudo de texto daquela tag, vamos **trocar** o valor da propriedade textContent para um novo texto, que é o nosso novo título Aparecida Nutricionista:

titulo.textContent = "Aparecida Nutricionista"

Ao recarregar a página você deve ver que assim que a página é carregada o novo título já aparece no topo!

Apesar deste capítulo inicial ter sido curto, aprendemos pontos importantes como:

* Separar a organização de nosso código em arquivos .js
* Importar arquivos .js sempre ao final do body
* A representação do HTML pela variável document
* A função que faz busca querySelector()
* Preferir selecionar por #id ou .classe

No próximo avançaremos mais em nossos conhecimentos de Javascript.

**Aula 02 - Consolidando o seu conhecimento**

## Buscando os dados do primeiro paciente

Neste capítulo vamos calcular o IMC do primeiro paciente , buscando os seus dados de altura e peso, fazendo as validações necessárias e em seguida vamos preencher o IMC na tabela.

Em seu arquivo principal.js, vá executando o passo a passo abaixo:

1- Vamos primeiro selecionar o primeiro paciente, que é uma <tr> com a nossa conhecida função document.querySelector() e salvar na variável paciente. Vamos usar o seletor de #id. Se você ainda não colocou, adicione o id primeiro-paciente no primeiro <tr> de nossa tabela:

<!-- restante do HTML -->

<tr class="paciente" id="primeiro-paciente">

<td class="info-nome">Paulo</td>

<td class="info-peso">100</td>

<td class="info-altura">2.00</td>

<td class="info-gordura">10</td>

<td class="info-imc">0</td>

</tr>

Em seguida no seu principal.js, selecione o paciente:

var paciente = document.querySelector("#primeiro-paciente");

2- Agora com o paciente em mãos, podemos buscar dentro dele as informações que precisamos que são a altura e o peso da paciente. Vamos utilizar a função querySelector() desta vez para buscar dentro da váriavel paciente, pelo <td> que contêm a altura e o peso:

var tdPeso = paciente.querySelector(".info-peso");

var tdAltura = paciente.querySelector(".info-altura");

3- Mas sabemos que não estamos interessados nos <td> e sim no conteudo de texto de cada um deles, então vamos extraí-los:

var tdPeso = paciente.querySelector(".info-peso");

var tdAltura = paciente.querySelector(".info-altura");

var peso = tdPeso.textContent;

var altura = tdAltura.textContent;

4- Com os valores de peso e altura em mãos, podemos calcular o IMC. Não vamos nos esquecer dos parênteses também para que nossa conta seja feita como esperamos:

var imc = peso / (altura \* altura);

5- Por último, com o IMC calculado devemos preencher a coluna de IMC do paciente selecionado com novo valor calculado. Vamos primeiro pegar o <td> que irá guardar o IMC:

var tdImc = paciente.querySelector(".info-imc");

6- Agora vamos colocar o valor calculado dentro do tdImc:

tdImc.textContent = imc;

Seu código deve estar assim:

var paciente = document.querySelector("#primeiro-paciente");

var tdPeso = paciente.querySelector(".info-peso");

var tdAltura = paciente.querySelector(".info-altura");

var peso = tdPeso.textContent;

var altura = tdAltura.textContent;

var imc = peso / (altura \* altura);

var tdImc = paciente.querySelector(".info-imc");

tdImc.textContent = imc;

Agora, ao recarregar a página, você verá que o IMC do primeiro paciente substituiu o valor apresentado no HTML.

Já estamos conseguindo calcular o IMC, porém falta validar os dados que vêm da tabela. Vamos lá:

1- O primeiro passo é fazer uma verificação do peso, vamos estabelecer que um peso é inválido se ele for menor ou igual a 0 quilos **OU** maior que 1000 quilos. Para isto, vamos utilizar um *if* para checar o peso, e a condição lógica de **OU** (||):

if(peso <= 0 || peso > 1000){

console.log("Peso inválido");

}

2- Não basta apenas exibir no console a validação. Vamos escrever na coluna do IMC também o erro de peso inválido:

if(peso <= 0 || peso > 1000){

console.log("Peso inválido");

tdImc.textContent = "Peso inválido!";

}

3- Vamos repetir a mesma lógica para a altura, só que considerando inválido uma altura menor ou igual a 0 metros ou maior ou igual a 3 metros.

if(altura <= 0 || altura >= 3){

console.log("Altura inválida");

tdImc.textContent = "Altura inválida!";

}

4- Agora não basta verificar se a altura ou o peso estão errados, só devemos calcular o IMC se ambos passarem na validação. Para isto, vamos criar duas variáveis, **pesoEhValido** e **alturaEhValida**, iniciar seus valores como **true**. Caso as validações falhem, trocaremos este valor para **false**:

var alturaEhValida = true;

var pesoEhValido = true;

if(peso <= 0 || peso > 1000){

console.log("Peso inválido");

tdImc.textContent = "Peso inválido!";

pesoEhValido = false;

}

if(altura <= 0 || altura >= 3){

console.log("Altura inválida");

tdImc.textContent = "Altura inválida!";

alturaEhValida = false;

}

5- Agora vamos fazer uma lógica para apenas calcular o IMC se o peso **E\*\* a altura forem válidos. Vamos utilizar o operador lógico de \*\*E** (&&):

var alturaEhValida = true;

var pesoEhValido = true;

if(peso <= 0 || peso > 1000){

console.log("Peso inválido");

tdImc.textContent = "Peso inválido!";

pesoEhValido = false;

}

if(altura <= 0 || altura >= 3){

console.log("Altura inválida");

tdImc.textContent = "Altura inválida!";

alturaEhValida = false;

}

if(pesoEhValido && alturaEhValida){

var imc = peso / ( altura \* altura);

tdImc.textContent = imc;

}

O seu código final deve estar assim:

var paciente = document.querySelector("#primeiro-paciente");

var tdAltura = paciente.querySelector(".info-altura");

var tdPeso = paciente.querySelector(".info-peso");

var tdImc = paciente.querySelector(".info-imc");

var altura = tdAltura.textContent;

var peso = tdPeso.textContent;

var alturaEhValida = true;

var pesoEhValido = true;

if (peso <= 0 || peso > 1000) {

console.log("Peso inválido!");

tdPeso.textContent = "Peso inválido!";

pesoEhValido = false;

}

if (altura <= 0 || altura >= 3) {

console.log("Altura inválida!");

tdAltura.textContent = "Altura inválida!";

alturaEhValida = false;

}

if (alturaEhValida && pesoEhValido) {

var imc = peso / (altura \* altura);

tdImc.textContent = imc;

} else {

tdImc.textContent = "Altura e/ou peso inválidos!"

}

Agora estamos validando antes de realizar o cálculo do IMC!

Chegamos ao fim de mais um capítulo, e neste aqui utilizamos um pouco da lógica de programação e das funções do Javascript para validar nosso primeiro paciente.

Neste capítulo vimos:

* O operador lógico de OU ( || )
* O operador lógico de E ( && )
* Busca de elementos

**Aula 03 - Consolidando o seu conhecimento**

Por enquanto nosso código só calcula e valida o IMC de **um** único paciente. Vamos atacar este problema neste capítulo para que o IMC de todos os pacientes sejam carregados:

1- Primeiro passo então é deixar de selecionar apenas um paciente e passar a selecionar **todos** os pacientes. Faremos isto buscando por todas as <tr> que tem a classe paciente. Claro, devemos também utilizar outra função, a função querySelectorAll() que é capaz de retornar mais de um elemento quando faz a busca no HTML. Faça a substituição no seu código para trocar o nome da variável paciente para pacientes e também faça a substituição da função e do seletor:

var pacientes = document.querySelectorAll(".paciente");

console.log(paciente);

2- Recarregando a página agora, veremos que o console dará um erro, mas é por que trocamos o nome da variável e ainda não adaptamos nosso código para trabalhar com um *array* de pacientes em vez de um único. Para validar as informações e calcular o IMC de cada paciente do array precisamos **percorrer** este array utilizando um clássico **for**:

for(var i = 0; i < pacientes.length ; i++){

var paciente = pacientes[i];

console.log(paciente);

}

3- Agora que conseguimos percorrer a lista de pacientes, basta mover o nosso código que validava e calculava o IMC do paciente para dentro da tabela. O código final deve ficar assim:

var pacientes = document.querySelectorAll(".paciente");

for(var i = 0; i < pacientes.length ; i++){

var paciente = pacientes[i];

var tdPeso = paciente.querySelector(".info-peso");

var tdAltura = paciente.querySelector(".info-altura");

var tdImc = paciente.querySelector(".info-imc");

var peso = tdPeso.textContent;

var altura = tdAltura.textContent;

var alturaEhValida = true;

var pesoEhValido = true;

if(peso <= 0 || peso >= 1000){

console.log("Peso inválido");

tdImc.textContent = "Peso inválido!";

pesoEhValido = false;

}

if(altura <= 0 || altura >= 3.0){

console.log("Altura inválida");

tdImc.textContent = "Altura inválida!";

alturaEhValida = false;

}

if(pesoEhValido && alturaEhValida){

var imc = peso / ( altura \* altura);

tdImc.textContent = imc;

}

}

4- Por último, podemos limitar o número de casas decimas do IMC utilizando a função .toFixed(). Faça a substuição para imprimir o IMC na tabela com apenas duas casas decimais:

// Restante do código

if(pesoEhValido && alturaEhValida){

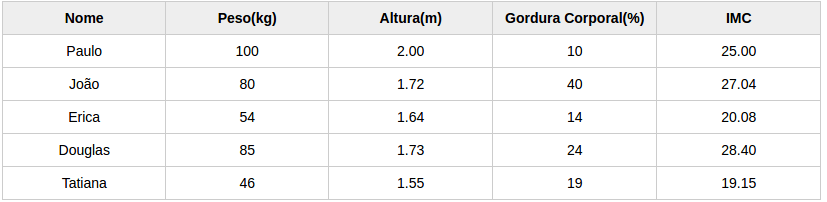
var imc = peso / ( altura \* altura);

//ALTERAÇÃO AQUI...

tdImc.textContent = imc.toFixed(2);

}

}



Agora cada paciente deve estar com IMC sendo cálculado!

Vamos melhorar um pouco a visualização de nossa tabela, para que os erros pintem a linha de vermelho além escrever na coluna de IMC quando ocorrer um erro.

1- Vamos trocar a cor da linha colocando uma nova classe na mesma, que será a classe paciente-invalido. Como primeiro passo crie uma nova classe no seu arquivo index.css como a classe abaixo:

.paciente-invalido{

background-color: lightcoral;

}

Nossa classe troca o background-color do elemento para a cor lightcoral, que é um vermelho claro.

2- Agora que temos uma classe com o estilo que queremos, devemos aplicá-la quando acontecer um erro, e os erros são detectados dentro de cada if. No if de peso, adicione a classe paciente-invalido a linha da tabela que isto ocorreu ( ou seja, na variável paciente):

if(peso <= 0 || peso >= 1000){

console.log("Peso inválido");

tdImc.textContent = "Peso inválido!";

pesoEhValido = false;

paciente.classList.add("paciente-invalido");

}

Estamos alterando o estilo colocando uma classe para facilitar a manutenção de nosso código, afinal se precisarmos alterar a cor, basta alterar o valor da propriedade background-color no CSS.

2- Faça a mesma coisa agora para quando a altura for inválida:

if(altura <= 0 || altura >= 3.00){

console.log("Altura inválida");

tdImc.textContent = "Altura inválida!";

alturaEhValida = false;

paciente.classList.add("paciente-invalido");

}

Pronto, agora temos um visual melhor quando ocorrer um erro em nossa tabela!

Neste capítulo aprendemos a lidar uma coleção de elementos do HTML e vimos:

* Como selecionar diversos elementos com a função querySelectorAll()
* Relembramos um método clássico de iteração com o **for**
* Vimos como manipular estilos com a propriedade .style
* Vimos que devemos utilizar camelCase quando queremos modificar uma propriedade que tenha duas palavras, como a background-color
* A boa prática de não alterar o estilo diretamente no Javascript e sim modificar as classes dos elementos com a propriedade .classList

**Aula 04 - Consolidando o seu conhecimento**

Neste capítulo vamos focar em adicionar um novo paciente na tabela através de um formulário.

1- O primeiro passo é criar o nosso formulário no HTML. Adicione o seguinte <form> abaixo da tag <main> :

<section class="container">

<h2 id="titulo-form">Adicionar novo paciente</h2>

<form id="form-adiciona">

<div class="grupo">

<label for="nome">Nome:</label>

<input id="nome" name="nome" type="text" placeholder="digite o nome do seu paciente" class="campo">

</div>

<div class="grupo">

<label for="peso">Peso:</label>

<input id="peso" name="peso" type="text" placeholder="digite o peso do seu paciente" class="campo campo-medio">

</div>

<div class="grupo">

<label for="altura">Altura:</label>

<input id="altura" name="altura" type="text" placeholder="digite a altura do seu paciente" class="campo campo-medio">

</div>

<div class="grupo">

<label for="gordura">% de Gordura:</label>

<input id="gordura" name="gordura" type="text" placeholder="digite a porcentagem de gordura do seu paciente" class="campo campo-medio">

</div>

<button id="adicionar-paciente" class="botao bto-principal">Adicionar</button>

</form>

</section>

2- Com o formulário adicionado, vamos começar adicionando um event listener no botão de adicionar ao formulário. Ao final do principal.js, selecione o botão e pendure o listener:

var botaoAdicionar = document.querySelector("#adicionar-paciente");

botaoAdicionar.addEventListener("click", function() {

console.log("Fui clicado!");

});

Teste se o click está funcionando no botão.

3- Se você testou o código acima, deve reparar que a mensagem do console.log não surge no console. Ou se aparece, ela apenas pisca e some rapidamente. Como estamos submetendo um <form> ,por padrão a página é recarregada, logo devemos impedir que isto ocorra. Adicione o parâmetro *event* na função anônima e dentro dela chame o event.preventDefault():

var botaoAdicionar = document.querySelector("#adicionar-paciente");

botaoAdicionar.addEventListener("click", function(event) {

event.preventDefault();

console.log("Fui clicado!");

});

Agora sim ao clicar vemos a mensagem "Fui clicado" surgir no console.

Com nosso formulário criado, podemos implementar o código Javascript responsável por criar um novo paciente na tabela.

Com o form criado e já escutando nossos eventos de click, vamos começar a implementar a adição de um novo paciente na tabela:

1- Nosso primeiro passo é selecionar o <form> para que tenhamos acesso aos campos de input com os dados do paciente. Vamos trazê-lo do HTML:

botaoAdicionar.addEventListener("click", function(event) {

event.preventDefault();

var form = document.querySelector("#form-adiciona");

});

2- Depois de selecionado o form, precisamos ter acesso a cada um dos inputs. Para fazer isto podemos nos apoiar numa característica do form, que é o acesso que temos aos seus inputs como se fossem propriedades do form, bastando usar como nome da propriedade os atributos *name* dos inputs. Como temos inputs com os name's com valor de nome,altura,peso e gordura faremos assim:

botaoAdicionar.addEventListener("click", function(event) {

event.preventDefault();

var form = document.querySelector("#form-adiciona");

var nome = form.nome.value;

var peso = form.peso.value;

var altura = form.altura.value;

var gordura = form.gordura.value;

});

Lembrando que para pegar o valor digitado dentro de um input, devemos acessar a sua propriedade value.

3- Agora com os valores extraídos do formulário HTML para o Javascript, podemos começar a criar um novo paciente na tabela. Sabemos que pacientes são representados por <tr>'s na tabela, então precisamos criar um elemento <tr> dentro do Javascript. Para isto, vamos utilizar a função document.createElement():

botaoAdicionar.addEventListener("click", function(event) {

event.preventDefault();

var form = document.querySelector("#form-adiciona");

var nome = form.nome.value;

var peso = form.peso.value;

var altura = form.altura.value;

var gordura = form.gordura.value;

var pacienteTr = document.createElement("tr");

});

4- Agora precisamos criar cada uma das <td>'s que ficarão dentro da <tr>. Como um paciente tem 5 atributos (nome,peso,altura,gordura e imc).Vamos utilizar a mesma função para criar estas <td's>:

botaoAdicionar.addEventListener("click", function(event) {

event.preventDefault();

var form = document.querySelector("#form-adiciona");

var nome = form.nome.value;

var peso = form.peso.value;

var altura = form.altura.value;

var gordura = form.gordura.value;

var pacienteTr = document.createElement("tr");

var nomeTd = document.createElement("td");

var pesoTd = document.createElement("td");

var alturaTd = document.createElement("td");

var gorduraTd = document.createElement("td");

var imcTd = document.createElement("td");

});

5- Temos que colocar os valores que extraímos do form dentro de cada uma das <td's>, então vamos preênche-las com nossa conhecida .textContent:

botaoAdicionar.addEventListener("click", function(event) {

event.preventDefault();

var form = document.querySelector("#form-adiciona");

var nome = form.nome.value;

var peso = form.peso.value;

var altura = form.altura.value;

var gordura = form.gordura.value;

var pacienteTr = document.createElement("tr");

var nomeTd = document.createElement("td");

var pesoTd = document.createElement("td");

var alturaTd = document.createElement("td");

var gorduraTd = document.createElement("td");

var imcTd = document.createElement("td");

nomeTd.textContent = nome;

pesoTd.textContent = peso;

alturaTd.textContent = altura;

gorduraTd.textContent = gordura;

});

6- Agora temos 5 <td>'s e 1 <tr> , mas elas não estão conectadas. O certo é posicionar as <td>'s dentro de cada <tr>'s. Vamos fazer isso através da função appendChild():

botaoAdicionar.addEventListener("click", function(event) {

event.preventDefault();

var form = document.querySelector("#form-adiciona");

var nome = form.nome.value;

var peso = form.peso.value;

var altura = form.altura.value;

var gordura = form.gordura.value;

var pacienteTr = document.createElement("tr");

var nomeTd = document.createElement("td");

var pesoTd = document.createElement("td");

var alturaTd = document.createElement("td");

var gorduraTd = document.createElement("td");

var imcTd = document.createElement("td");

nomeTd.textContent = nome;

pesoTd.textContent = peso;

alturaTd.textContent = altura;

gorduraTd.textContent = gordura;

pacienteTr.appendChild(nomeTd);

pacienteTr.appendChild(pesoTd);

pacienteTr.appendChild(alturaTd);

pacienteTr.appendChild(gorduraTd);

});

7- Por último precisamos adicionar esta <tr> na tabela. Para isto vamos usar uma tática parecida, selecionar a tabela em seguida usar a função .appendChild():

var botaoAdicionar = document.querySelector("#adicionar-paciente");

botaoAdicionar.addEventListener("click", function(event) {

event.preventDefault();

var form = document.querySelector("#form-adiciona");

var nome = form.nome.value;

var peso = form.peso.value;

var altura = form.altura.value;

var gordura = form.gordura.value;

var pacienteTr = document.createElement("tr");

var nomeTd = document.createElement("td");

var pesoTd = document.createElement("td");

var alturaTd = document.createElement("td");

var gorduraTd = document.createElement("td");

var imcTd = document.createElement("td");

nomeTd.textContent = nome;

pesoTd.textContent = peso;

alturaTd.textContent = altura;

gorduraTd.textContent = gordura;

pacienteTr.appendChild(nomeTd);

pacienteTr.appendChild(pesoTd);

pacienteTr.appendChild(alturaTd);

pacienteTr.appendChild(gorduraTd);

var tabela = document.querySelector("#tabela-pacientes");

tabela.appendChild(pacienteTr);

});

Agora sim estamos adicionando o paciente a partir do formulário, mas veja que ele ainda está sem a coluna de IMC. Vamos resolver isto no próximo capítulo com boas práticas de reuso de código, até lá!

Neste capítulo implementamos a criação de usuários a partir de um formulário, e com isso aprendemos:

* A diferença entre as funções nomeadas e as funções anônimas
* A escutar eventos do browser com a função addEventListener()
* Que a função criadora de elementos é .createElement()
* A pegar o valor de um input por meio da propriedade .value
* A acessar os input de um form por meio da propriedade \_name\_.
* A adicionar elementos na página e dentro de outros elementos com a função appendChild()

**Aula 05 - Consolidando o seu conhecimento**

Neste capítulo vamos organizar nosso código para que ele fique mais fácil de manter e para que sigamos as boas práticas.

1- Nosso arquivo principal.js estava começando a ficar muito grande, logo vamos separá-lo em arquivo Javascript individuais, aonde cada arquivo ficará responsável por uma funcionalidade do sistema. Primeiro, crie um novo arquivo chamado form.js, e importe-o no fim no seu HTML:

// Restante do código HTML

</section>

</main>

<!-- Importação dos Javascripts AQUI -->

<script src="js/principal.js"></script>

<script src="js/form.js"></script>

</body>

</html>

2- Com o arquivo criado e importado, mova tudo que estiver relacionado com nosso formulário de adicionar paciente para lá. Basicamente tudo dentro do event listener do click do botão:

//form.js

var botaoAdicionar = document.querySelector("#adicionar-paciente");

botaoAdicionar.addEventListener("click", function(event) {

// Todo o código da função.

});

3- Aproveite e renomeie o seu arquivo principal.js para um nome mais semântico, como calcula-imc.js , que diz melhor o que aquele código faz.

4- Agora que já fizemos esta separação em arquivos, vamos começar a melhorar mais ainda o código. É a hora de exportarmos certas partes do código para funções, assim fica mais fácil reutilizá-las. Um bom exemplo é o código responsável por calcular um imc, que deve ser usado tanto na função cálcula IMC quanto quando criamos um novo paciente através do formulário. Crie a função calculaImc em seu arquivo calcula-imc.js como abaixo:

//calcula-imc.js

// restante do código

function calculaImc(peso, altura){

var imc = 0;

imc = peso / (altura \* altura);

return imc.toFixed(2);

}

5- Agora substitua o antigo cálculo na mão que fazíamos dentro do *if* por uma chamada a nossa recém criada função calculaImc, passando os parâmetros de acordo:

//calcula-imc.js

// Restante do código

if (pesoEhValido && alturaEhValida) {

var imc = calculaImc(peso,altura);

imcTd.textContent = imc; }

// Restante do código

6- Vamos também chamar a função calculaImc no nosso form.js, para que o IMC do paciente também seja calculado quando ele for inserido na tabela. Adicione junto dos outros <td's> para que o contéudo do tdImc seja o retorno da função calculaImc

//form.js

nomeTd.textContent = nome;

pesoTd.textContent = peso;

alturaTd.textContent = altura;

gorduraTd.textContent = gordura;

//Adicionar aqui:

imcTd.textContent = calculaImc(peso,altura);

7- E não vamos esquecer de colocar o tdImc também dentro do <tr> paciente:

//form.js

pacienteTr.appendChild(nomeTd);

pacienteTr.appendChild(pesoTd);

pacienteTr.appendChild(alturaTd);

pacienteTr.appendChild(gorduraTd);

//Adicionar aqui:

pacienteTr.appendChild(imcTd);

Agora quando o paciente for adicionado na tabela, seu IMC também será calculado e inserido automaticamente!

Graças as boas práticas de reuso de código e de uma melhor organização não precisamos escrever o código responsável de calcular o IMC duas vezes, e ainda separamos nosso código Javascript em arquivos diferentes, para que fique mais fácil dar manutenção ao mesmo.

Dando continuidade as boas práticas de organização de código, uma outra que devemos atacar é que nossa função anônima do form.js está com muitas responsabilidades, ela sozinha está fazendo muitas coisas. Ela obtém um paciente do formulário, cria a <tr> paciente, cria diversos <td>, coloca um dentro do outro e depois ainda adiciona-os na tabela!

São muitas funcionalidades para uma única função, vamos quebrá-la em funções menores para melhorar a legibilidade de nosso código:

1- O primeiro passo é extrair a responsabilidade de obter os dados do formulário para uma nova função. Crie a função obtemPacienteDoFormulario, que irá cuidar disto. Esta função deve receber o formulário e retornar todos os dados do paciente, e para isto vamos salvar estes dados dentro de um objeto do Javascript :

// form.js

function obtemPacienteDoFormulario(form) {

var paciente = {

nome: form.nome.value,

peso: form.peso.value,

altura: form.altura.value,

gordura: form.gordura.value,

imc: calculaImc(form.peso.value, form.altura.value)

}

return paciente;

}

2- Agora vamos chamar está função no local aonde criávamos várias variáveis com peso, altura,etc..:

// form.js

botaoAdicionar.addEventListener("click", function(event) {

event.preventDefault();

var form = document.querySelector("#form-adiciona");

// Remova a criação das variaveis individuais e deixe apenas o objeto paciente

var paciente = obtemPacienteDoFormulario(form);

// Restante do código

...

});

3- A próxima parte que iremos atacar é a criação da <tr> paciente. Vamos criar uma função para criar a <tr> e uma para as <td>. Vamos começar pela função montaTd, que deve receber o dado que vai ser colocado dentro da td e a classe, e nos retornar o objeto montado:

function montaTd(dado, classe) {

var td = document.createElement("td");

td.classList.add(classe);

td.textContent = dado;

return td;

}

4- Agora aproveitando a função montaTd, vamos criar a montaTr, que vai receber um objeto paciente, criar cada uma das td, e colocar dentro da tr :

function montaTr(paciente) {

//Cria TR

var pacienteTr = document.createElement("tr");

pacienteTr.classList.add("paciente");

//Cria as TD's e a adiciona dentro da TR

pacienteTr.appendChild(montaTd(paciente.nome, "info-nome"));

pacienteTr.appendChild(montaTd(paciente.peso, "info-peso"));

pacienteTr.appendChild(montaTd(paciente.altura, "info-altura"));

pacienteTr.appendChild(montaTd(paciente.gordura, "info-gordura"));

pacienteTr.appendChild(montaTd(paciente.imc, "info-imc"));

// retorna a TR

return pacienteTr;

}

5- Agora vamos fazer as substituições no event listener:

var botaoAdicionar = document.querySelector("#adicionar-paciente");

botaoAdicionar.addEventListener("click", function(event) {

event.preventDefault();

var form = document.querySelector("#form-adiciona");

var paciente = obtemPacienteDoFormulario(form);

var pacienteTr = montaTr(paciente);

var tabela = document.querySelector("#tabela-pacientes");

tabela.appendChild(pacienteTr);

});

6- O código ficou bem mais limpo e organizado! Por último , vamos fazer que o form seja limpo após adicionar um paciente. Utilize a função .reset() como último comando:

var botaoAdicionar = document.querySelector("#adicionar-paciente");

botaoAdicionar.addEventListener("click", function(event) {

// restante do código

tabela.appendChild(pacienteTr);

form.reset();

});

Veja que agora temos um código muito mais organizado e fácil de ler!

Neste capítulo vimos

* Dividir o código, separando por funcionalidades do sistema
* Os objetos em Javascript
* A quebrar funções grandes em funções menores ,com cada uma com sua responsabilidade
* A função form.reset() para limpar o formulário

**Aula 06 - Consolidando o seu conhecimento**

Vamos aproveitar as regras de validação que tínhamos no calcula-imc.js para poder validar os pacientes que adicionamos pelo formulário e poder exibir as mensagens de erro.

1- Vamos começar exportando a lógica de validação de peso e altura para duas funções externas, a validaPeso e a validaAltura. Em seu arquivo calcula-imc.js crie as seguintes funções:

function validaPeso(peso) {

if (peso > 0 && peso < 1000) {

return true;

} else {

return false;

}

}

function validaAltura(altura) {

if (altura > 0 && altura <= 3.00) {

return true;

} else {

return false;

}

}

2- Agora faça as seguintes alterações no seu for para usarmos nossas funções validadoras:

for (var i = 0; i < pacientes.length; i++) {

//Restante do código

...

// Alteração

var alturaEhValida = validaAltura(altura);

var pesoEhValido = validaPeso(peso);

// Alteração

if (!pesoEhValido) {

...

}

// Alteração

if (!alturaEhValida) {

...

}

// Alteração

if (pesoEhValido && alturaEhValida) {

var imc = calculaImc(peso, altura);

tdImc.textContent = imc;

}

}

3- Vamos agora criar a função que irá validar o usuário quando submetermos o form. O nome desta será validaPaciente, que receberá o objeto paciente e retornará um array de erros:

// form.js

function validaPaciente(paciente) {

var erros = [];

if (paciente.nome.length == 0) {

erros.push("O nome não pode ser em branco");

}

if (paciente.gordura.length == 0) {

erros.push("A gordura não pode ser em branco");

}

if (paciente.peso.length == 0) {

erros.push("O peso não pode ser em branco");

}

if (paciente.altura.length == 0) {

erros.push("A altura não pode ser em branco");

}

if (!validaPeso(paciente.peso)) {

erros.push("Peso é inválido");

}

if (!validaAltura(paciente.altura)) {

erros.push("Altura é inválida");

}

return erros;

}

Esta função cria um array e caso encontre algum erro em alguma propriedade do paciente, adiciona ao array de erros a mensagem que será exibida depois.

4- Agora vamos chamar nossa função validaPaciente quando clicarmos no botão adiciona e tratar de exibir a mensagem de erro caso haja alguma:

botaoAdicionar.addEventListener("click", function(event) {

event.preventDefault();

var form = document.querySelector("#form-adiciona");

var paciente = obtemPacienteDoFormulario(form);

var pacienteTr = montaTr(paciente);

//Adição aqui

var erros = validaPaciente(paciente);

if (erros.length > 0) {

// Aqui vai entrar o código de exibir erros.

}

// Resto do código

...

});

5- Com toda a lógica preparada, basta apenas exibirmos as mensagens de erro de dentro do Array. Para isto, vamos criar uma <ul> acima do formulário de adicionar pacientes para exibir as mensagens:

//principal.html

...

<ul id="mensagens-erro"></ul>

<form>

...

6- Crie a função exibeMensagensDeErro, que recebe o array de erros e cria uma <li> para cada mensagem de erro:

function exibeMensagensDeErro(erros) {

var ul = document.querySelector("#mensagens-erro");

ul.innerHTML = "";

erros.forEach(function(erro) {

var li = document.createElement("li");

li.textContent = erro;

ul.appendChild(li);

});

}

7- Agora basta chamar a função exibeMensagemErro() dentro do if que verifica se existe algum erro no array:

//form.js

botaoAdicionar.addEventListener("click", function(event) {

event.preventDefault();

var form = document.querySelector("#form-adiciona");

var paciente = obtemPacienteDoFormulario(form);

var erros = validaPaciente(paciente);

if (erros.length > 0) {

//Adição aqui

exibeMensagensDeErro(erros);

return;

}

// Resto do código

var pacienteTr = montaTr(paciente);

...

});

8- Para que as mensagens de erro fiquem vermelhas, adicione o seguinte estilo abaixo em seu estilos.css:

// estilos.css

// Restante do CSS

...

#mensagens-erro{

color: red;

}

9- Por último, vamos adicionar nosso paciente na tabela, além disso, precisamos remover as mensagens de erro caso seja adicionado um paciente válido.

O código para isso é:

var tabela = document.querySelector("#tabela-pacientes");

tabela.appendChild(pacienteTr);

form.reset();

var mensagensErro = document.querySelector("#mensagens-erro");

mensagensErro.innerHTML = "";

Nosso código completo será:

//form.js

botaoAdicionar.addEventListener("click", function(event) {

event.preventDefault();

var form = document.querySelector("#form-adiciona");

var paciente = obtemPacienteDoFormulario(form);

var erros = validaPaciente(paciente);

if (erros.length > 0) {

//Adição aqui

exibeMensagensDeErro(erros);

return;

}

var pacienteTr = montaTr(paciente);

var tabela = document.querySelector("#tabela-pacientes");

tabela.appendChild(pacienteTr);

form.reset();

var mensagensErro = document.querySelector("#mensagens-erro");

mensagensErro.innerHTML = "";

});

Neste capítulo vimos:

* Mais organização de código, exportando as funções de validação.
* O operador de negação NOT (!)
* Como validar um formulário
* A função push para colocar elementos dentro de um array
* A propriedade innerHTML dos elementos, que foi usada para apagar os itens da <ul>
* O método forEach para percorrer arrays

**Aula 07 - Consolidando o seu conhecimento**

Mãos na Massa: Removendo os pacientes

Neste capítulos aprendemos mais sobre **eventos** e vimos que através da delegação de eventos conseguimos reduzir o número de eventListeners na página e ainda assim executar as funções que desejamos. Vamos implementar a remoção de pacientes colocando um escutador na tabela:

1- Nosso primeiro passo é criar um novo arquivo /js, já que se trata de uma nova funcionalidade vamos chamá-lo de remove-paciente.js:

introducao-javascript

└── js

├── calcula-imc.js

├── form.js

└── remove-paciente.js

2- Não vamos esquecer de importar o nosso novo arquivo .js ao fim da tag <body>:

//index.html

</section>

<script src="js/calcula-imc.js" ></script>

<script src="js/form.js" ></script>

<script src="js/remove-paciente.js" ></script>

</body>

3- Agora vamos selecionar o corpo da tabela para adicionarmos um eventListener de duplo clique. Dentro do remove-paciente.js faça:

// remove-paciente.js

var tabela = document.querySelector("#tabela-pacientes");

tabela.addEventListener("dblclick", function() {

});

4- Agora devemos remover a linha que foi clicada, porém como o eventListener está na tabela, devemos perguntar primeiro qual das <td> que foi clicada, em seguida buscar pelo seu pai <tr> e removê-lo:

// remove-paciente.js

var tabela = document.querySelector("#tabela-pacientes");

tabela.addEventListener("dblclick", function(event) {

event.target.parentNode.remove();

});

Não esqueça de adicionar o parâmetro event na função anônima do evento!

Agora ao dar um duplo clique no elemento, o evento sobe até a tabela, e dentro da função conseguimos perguntar qual td foi clicado (event.target) e partir disto subir para o seu pai (parentNode) que é a <tr> do paciente e removê-lá.

### Dica

Não sei se você notou, mas se der um duplo clique no cabeçalho da nossa tabela, esta linha também será removida.

Podemos resolver isto de uma maneira simples. Pense um pouco... que elemento desejamos remover da tabela?

Sim, a tr pai do elemento em quem clicamos. Mas em que elemento estamos clicando? Estamos clicando dentro da "célula" da tabela, ou seja, no td.

Mas quando fazemos isto no cabeçalho, na realidade estamos clicando em um th.

Acho que você já pegou a lógica. Não queremos que o pai do th seja removido, somente o pai do td. Ou para tornar mais simples: somente remova o pai se o elemento clicado for um td.

Os elementos de nossa página possuem o atributo tagName que guarda uma string com o nome da tag do elemento. Faça um teste na página da Aparecida Nutrição, abra o console e execute document.querySelector('h1').tagName. O console te retornará o nome da tag do elemento em maiúsculo ("H1").

Em nosso caso queremos saber o nome da tag do elemento clicado, portanto segue a implementação:

var tabela = document.querySelector('#tabela-pacientes');

tabela.addEventListener('dblclick', function(event) {

// Somente executará nosso código caso o elemento em que clicamos seja um <td>

if (event.target.tagName == 'TD') {

event.target.parentNode.remove()

}

});

Pronto, agora a nutricionista não vai mais apagar o cabeçalho sem querer. Bom trabalho!

Agora que estamos removendo o paciente, vamos no próximo exercício fazer a animação da remoção, para que paciente não suma instantaneamente da página.

Vamos agora adicionar uma pequena animação na remoçao de cada paciente, fazendo uma mistura de técnicas de CSS e Javascript.

1- Vamos adicionar um CSS que será responsável por animar a remoção. Em seu index.css faça:

//index.css

.fadeOut {

opacity: 0;

transition: 0.5s;

}

Essa classe que deveremos adicionar ao elementos que vamos remover.

2- Vamos utilizar a técnica vista no vídeo, de aplicar a classe, chamar a função setTimeout para aguardar a animação terminar e ao fim dela, remover o elemento dá página. Primeiro adicione a classe na <tr>:

var tabela = document.querySelector("#tabela-pacientes");

tabela.addEventListener("dblclick", function(event) {

event.target.parentNode.classList.add("fadeOut");

event.target.parentNode.remove();

});

3- Agora vamos utilizar a setTimeout para aguardar antes de remover:

var tabela = document.querySelector("#tabela-pacientes");

tabela.addEventListener("dblclick", function(event) {

event.target.parentNode.classList.add("fadeOut");

setTimeout(function() {

event.target.parentNode.remove();

}, 500);

});

Deste modo, quando o usuário clicar para remover um paciente, ele vai esmaecer por meio segundo antes de sumir!

Neste capítulo aprendemos:

* O novo evento de dblclick
* Como os eventos propagam atráves da página
* A delegação de eventos
* Como animar uma remoção com o setTimeout

**Aula 08 - Consolidando o seu conhecimento**

Neste capítulo nós vamos implementar um campo de texto para filtrar os pacientes de nossa tabela pelo nome. Vamos lá:

1- Comece criando o input e label do campo aonde usuário irá digitar. No seu index.html, entre o título "Meus Pacientes" e a tabela, adicione:

//index.html

...

<h2>Meus pacientes</h2>

<label for="filtrar-tabela">Filtre:</label>

<input type="text" name="filtro" id="filtrar-tabela" placeholder="Digite o nome do paciente">

<table>

...

2- Vamos aproveitar e estilizar um pouco este campo, para que ele fique com uma letra um pouco maior. No seu estilos.css adicione o estilo abaixo:

//estilos.css

...

#filtrar-tabela {

width: 200px;

height: 35px;

margin-bottom: 10px;

}

3- Agora vamos para o Javascript, como se trata de uma nova funcionalidade, vamos criar um novo arquivo na pasta /js chamado filtra.js:

introducao-javascript

└── js

├── calcula-imc.js

├── form.js

├── remover-paciente.js

└── filtra.js

4- E vamos importá-lo em nosso HTML ao fim da tab <body>:

</section>

<script src="js/calcula-imc.js" ></script>

<script src="js/form.js" ></script>

<script src="js/remover-paciente.js" ></script>

<script src="js/filtra.js" ></script>

</body>

5- Vamos começar selecionando o campo de input, pois temos o interesse de pegar o seu valor para realizar a busca:

//filtra.js

var campoFiltro = document.querySelector("#filtrar-tabela");

6- Vamos adicionar um novo evento nele, o evento de input que é responsável por detectar quando o usuário começar a digitar no campo:

//filtra.js

var campoFiltro = document.querySelector("#filtrar-tabela");

campoFiltro.addEventListener("input", function() {

});

7- Para realizar a busca, vamos utilizar a seguinte técnica: Toda vez que o usuário começar a digitar alguma coisa, vamos esconder todos os itens da tabela, e só exibir aquele que batem com a busca. E quando o usuário deixar o campo em branco, vamos voltar a exibir os itens da tabela. Para esconder os itens, vamos criar uma nova classe chamada de invisivel no estilos.css:

//estilos.cs

....

.invisivel {

display: none;

}

Um simples display:none irá ocultar as linhas da tabela. Vamos aplicá-la quando usuário digitar algo.

8- Sabemos que o que foi digitado pelo usuário pode ser acessado através do this.value, afinal o *this* neste caso é o <input> e podemos ter acesso ao seu contéudo com a propriedade .value. Se detectarmos algo escrito no campo, vamos esconder todos os pacientes da tabela colocando a classe .invisivel, caso contrário, vamos remover a classe invisivel e exibi-los novamente:

var campoFiltro = document.querySelector("#filtrar-tabela");

campoFiltro.addEventListener("input", function() {

var pacientes = document.querySelectorAll(".paciente");

if (this.value.length > 0) {

for (var i = 0; i < pacientes.length; i++) {

var paciente = pacientes[i];

paciente.classList.add("invisivel");

}

} else {

for (var i = 0; i < pacientes.length; i++) {

var paciente = pacientes[i];

paciente.classList.remove("invisivel");

}

}

});

Experimente digitar algo na tabela, verá que todos os pacientes somem, mas ao deixar o campo em branco, ou seja, com o .length de tamanho zero, os pacientes voltam a aparecer.

Por enquanto só estamos escondendo todos os pacientes, vamos realizar a busca individual por cada um deles no próximo exercício!

Vamos agora fazer a busca pelo o que o usuário digitou no campoFiltro. Se bater com o nome de algum paciente, vamos exibí-lo, caso contrário vamos continuar deixando-o escondido:

1- O primeiro passo é conseguir extrair o nome de cada paciente para comparar o valor do campoFiltro (this.value). Vamos buscar o nome de dentro da <tr> como já fizemos diversas vezes anteriormente:

var campoFiltro = document.querySelector("#filtrar-tabela");

campoFiltro.addEventListener("input", function() {

var pacientes = document.querySelectorAll(".paciente");

if (this.value.length > 0) {

for (var i = 0; i < pacientes.length; i++) {

var paciente = pacientes[i];

//Adição aqui

var tdNome = paciente.querySelector(".info-nome");

var nome = tdNome.textContent;

paciente.classList.add("invisivel");

}

} else {

for (var i = 0; i < pacientes.length; i++) {

var paciente = pacientes[i];

paciente.classList.remove("invisivel");

} } });

2- Agora precisamos **comparar** o que o usuário digitou com o nome de cada um dos pacientes. Vimos uma ferramenta muito poderosa para comparar textos que é Expressão Regular, que nos permite comparar até mesmo parte de textos, com por exemplo achar a palavra "Pedro", buscando apenas por "Ped".

Como queremos que a nossa Expressão Regular busque pelo o que o usuário digitou , vamos criar um nova expressão com este parâmetro:

var campoFiltro = document.querySelector("#filtrar-tabela");

campoFiltro.addEventListener("input", function() {

var pacientes = document.querySelectorAll(".paciente");

if (this.value.length > 0) {

for (var i = 0; i < pacientes.length; i++) {

var paciente = pacientes[i];

var tdNome = paciente.querySelector(".info-nome");

var nome = tdNome.textContent;

//Adição aqui

var expressao = new RegExp(this.value, "i");

paciente.classList.add("invisivel");

}

} else {

for (var i = 0; i < pacientes.length; i++) {

var paciente = pacientes[i];

paciente.classList.remove("invisivel");

} } });

O modificador "i" é para indicar que estamos buscando por case-insensitive, ou seja tanto "Pa" quanto "pa" achariam a palavra "Paulo", ele não liga para a diferença entre maísuculas e minúsculas.

3- Com a expressão regular em mãos, só precisamos pedir para ela testar se o nome do usuário bate com o que foi digitado. Para isto, vamos utilizar sua função .test, que **em caso positivo** deve exibir o paciente **removendo a sua classe invisivel**, e em caso **negativo** deve esconder o paciente **adicionando a classe invisivel** :

var campoFiltro = document.querySelector("#filtrar-tabela");

campoFiltro.addEventListener("input", function() {

var pacientes = document.querySelectorAll(".paciente");

if (this.value.length > 0) {

for (var i = 0; i < pacientes.length; i++) {

var paciente = pacientes[i];

var tdNome = paciente.querySelector(".info-nome");

var nome = tdNome.textContent;

var expressao = new RegExp(this.value, "i");

// Adição aqui

if (expressao.test(nome)) {

paciente.classList.remove("invisivel");

} else {

paciente.classList.add("invisivel");

}

}

} else {

for (var i = 0; i < pacientes.length; i++) {

var paciente = pacientes[i];

paciente.classList.remove("invisivel");

}

}

});

Faça o teste agora e veja que é possível encontrar pacientes buscando apenas por uma parte de seus nomes!

Neste capítulo aprendemos:

* O novo evento de input
* O truque de deixar invisível adicionando uma classe com display:none
* Como realizar uma busca, escondendo todos os elementos e mostrando apenas aqueles que nos interessam
* Como buscar por apenas um pedaço de uma palavra com as Expressões Regulares em Javascript

# Para saber mais: filtrando sem regex?

Aprendemos como filtrar, comparar letra a letra, utilizando expressões regulares, representado pelo fragmento de código abaixo:

var expressao = new RegExp(this.value,"i");

if (!expressao.test(nome)) {

paciente.classList.add("invisivel");

} else{

paciente.classList.remove("invisivel");

}

Mas há um modo de fazer essa comparação sem a necessidade de utilizar expressões regulares! Podemos utilizar a função **substring**, que recebe dois parâmetros, fazendo com que ela devolva parte da string, com o tamanho definido nos parâmetros. O primeiro parâmetro é o início, começando do 0 (que representa o primeiro caractere). O segundo parâmetro define o fim (exclusivo, mostramos até o penúltimo caractere). Por exemplo:

var string = "Alura";

var resultado = string.substring(1, 4);

Extraímos uma string que começa no segundo caractere (número 1) e termina no quarto caractere (número 3, índice anterior ao número 4). O valor da variável **resultado** é:

lur

Conhecendo essa função, pense em um modo de comparar o valor digitado com parte do nome, sem utilizar expressões regulares. Veja em seguida a resposta do instrutor.

Como o primeiro parâmetro é o inicio, e queremos comparar desde o início da string **nome**, vamos utilizar como início o valor **0**, ou seja, sempre a partir do primeiro caractere. Mas qual é o fim? O fim é justamente o tamanho do valor digitado:

nome.substr(0, this.value.length);

Podemos guardar essa string em uma variável, e compará-la com o que está sendo digitado. Caso seja falso, adicionamos a classe **invisivel**, se não for, removemos-a:

var comparavel = nome.substr(0, this.value.length);

if (!(this.value == comparavel)) {

paciente.classList.add("invisivel");

} else{

paciente.classList.remove("invisivel");

}

Mas e a distinção entre letras maiúsculas e minúsculas? Nesse caso não temos distinção entre letras maiúsculas e minúsculas, mas para contornar isso, antes de compará-las, podemos colocar as duas strings em letras minúsculas, para efetuar a comparação entre elas em seguida:

var comparavel = nome.substr(0, this.value.length);

var comparavelMinusculo = comparavel.toLowerCase();

var valorDigitadoMinusculo = this.value.toLowerCase();

if (!(valorDigitadoMinusculo == comparavelMinusculo)) {

paciente.classList.add("invisivel");

} else{

paciente.classList.remove("invisivel");

}

Esta é uma alternativa de implementar a mesma funcionalidade sem expressão regular, porém temos que escrever mais e nos preocupar com mais detalhes! Fica ai esta opção para você guardar nos seus conhecimentos.

**Aula 09 - Consolidando o seu conhecimento**

Neste capítulo nós implementaremos o botão buscar, que como vimos no vídeo, deve ir em um servidor externo através de uma requisição GET e trazer os dados de novos pacientes.

1- Como de praxe , já que se trata de uma nova funcionalidade vamos começar criando um novo arquivo na pasta /js, chamado de buscar-pacientes.js:

introducao-javascript

└── js

├── calcula-imc.js

├── form.js

├── remover-paciente.js

├── filtra.js

└── buscar-pacientes.js

2- Vamos importá-lo ao fim da tag <body>:

</section>

<script src="js/calcula-imc.js" ></script>

<script src="js/form.js" ></script>

<script src="js/remover-paciente.js" ></script>

<script src="js/filtra.js" ></script>

<script src="js/buscar-pacientes.js" ></script>

</body>

3- Agora vamos criar o botão Buscar Pacientes no HTML. Abaixo da <table> e acima do fechamento da <section> adicione o botão abaixo:

</table>

// Adicionar aqui!

<button id="buscar-pacientes" class="botao bto-principal">Buscar Pacientes</button>

</section>

</main>

4- Agora podemos no buscar-pacientes.js selecionar o botão e colocar um eventListener de click nele:

var botaoAdicionar = document.querySelector("#buscar-pacientes");

botaoAdicionar.addEventListener("click", function() {

});

5- Para executarmos requisições com o Javascript, precisamos de um objeto especialista nisso, que é o XMLHttpRequest. Vamos criar um novo objeto deste tipo:

var botaoAdicionar = document.querySelector("#buscar-pacientes");

botaoAdicionar.addEventListener("click", function() {

var xhr = new XMLHttpRequest();

});

6- O XMLHttpRequest precisa ser configurado, para dizer qual método HTTP queremos utilizar na requisição, e para qual servidor vamos enviá-la. Para configurar o XMLHttpRequest utilizamos a função .open():

var botaoAdicionar = document.querySelector("#buscar-pacientes");

botaoAdicionar.addEventListener("click", function() {

var xhr = new XMLHttpRequest();

xhr.open("GET", "https://api-pacientes.herokuapp.com/pacientes");

});

7- Por último , para enviar a requisição precisamos chamar o método .send():

var botaoAdicionar = document.querySelector("#buscar-pacientes");

botaoAdicionar.addEventListener("click", function() {

var xhr = new XMLHttpRequest();

xhr.open("GET", "https://api-pacientes.herokuapp.com/pacientes");

xhr.send();

});

Se você clicar no botão neste momento, a requisição será enviada, porém não veremos nenhuma alteração na tela ou resultado no console, pois não estamos pegando a resposta. Vamos tratar disto.

8- Para pegarmos a resposta quando a requisiçao HTTP voltar precisamos colocar um escutador de evento no próprio XMLHttpRequest, escutando o evento de load:

var botaoAdicionar = document.querySelector("#buscar-pacientes");

botaoAdicionar.addEventListener("click", function() {

var xhr = new XMLHttpRequest();

xhr.open("GET", "https://api-pacientes.herokuapp.com/pacientes");

xhr.addEventListener("load", function() {

console.log(xhr.responseText);

});

xhr.send();

});

Clique no botão agora e observe o seu console, você verá que agora conseguimos exibir os resultados!

Por enquanto estamos apenas exibindo o retorno de nossa requisição no console, mas no próximo exercício vamos adicionar estes pacientes na tabela!

Já conseguimos realizar a requisição HTTP, ir ao servidor e trazer os dados dos pacientes, que até mesmo imprimimos no console e vimos que a resposta é um grande texto formatado. Agora neste exercício vamos transformar este texto em um objeto Javascript e adicionar cada um destes pacientes na tabela.

1- Se você observar o que é impresso no console, você vai ver que o servidor nos retorna um JSON, um formato de texto muito comum hoje em dia na web. Como não queremos trabalhar com texto e sim com objetos Javascript, vamos parsear este texto para um objeto Javascript equivalente. Use a função JSON.parse() na resposta para transformar:

var botaoAdicionar = document.querySelector("#buscar-pacientes");

botaoAdicionar.addEventListener("click", function() {

var xhr = new XMLHttpRequest();

xhr.open("GET", "https://api-pacientes.herokuapp.com/pacientes");

xhr.addEventListener("load", function() {

var resposta = xhr.responseText;

var pacientes = JSON.parse(resposta);

});

xhr.send();

});

2- Nossa variável pacientes agora é um array de objetos paciente, com nome, peso, altura, gordura e IMC. Agora precisamos iterar por este array e adicionar cada um destes pacientes na tabela, mas antes de começar a implementar isto na mão, vamos refatorar um pedaço de código que já havíamos feito para facilitar nossa vida.

3- Vamos refatorar um pedaço do código do form.js para aproveitar o código que adiciona uma paciente na tabela, exportando este código para uma função externa que aproveitaremos no nosso buscar-pacientes.js, afinal também queremos adicionar um paciente lá.

Em seu form.js crie a função adicionaPacienteNaTabela, que recebe um paciente. Ela vai fazer o que já fazíamos depois de montar um paciente do form. Só que agora, portado para uma função externa.

// Crie esta função em seu form.js

function adicionaPacienteNaTabela(paciente) {

var pacienteTr = montaTr(paciente);

var tabela = document.querySelector("#tabela-pacientes");

tabela.appendChild(pacienteTr);

}

4- Agora no evento de click do form.js, faça as substuições e chame a função adicionaPacienteNaTabela aqui também. Seu código final do evento deve ficar assim:

botaoAdicionar.addEventListener("click", function(event) {

event.preventDefault();

var form = document.querySelector("#form-adiciona");

var paciente = obtemPacienteDoFormulario(form);

var erros = validaPaciente(paciente);

if (erros.length > 0) {

exibeMensagensDeErro(erros);

return;

}

adicionaPacienteNaTabela(paciente);

form.reset();

var mensagensErro = document.querySelector("#mensagens-erro");

mensagensErro.innerHTML = "";

});

Cuidado ao fazer esta refatoração! Se for necessário copie e cole este código para que não haja erros!

5- Voltando para o buscar-pacientes.js, agora que temos uma função especialista em adicionar pacientes na tabela , vamos aproveitá-la para percorrer o array de pacientes e adicionar cada um deles:

var botaoAdicionar = document.querySelector("#buscar-pacientes");

botaoAdicionar.addEventListener("click", function() {

var xhr = new XMLHttpRequest();

xhr.open("GET", "https://api-pacientes.herokuapp.com/pacientes");

xhr.addEventListener("load", function() {

var resposta = xhr.responseText;

var pacientes = JSON.parse(resposta);

pacientes.forEach(function(paciente) {

adicionaPacienteNaTabela(paciente);

});

});

xhr.send();

});

Clique no botão de buscar pacientes e veja que agora os pacientes são carregados!

Estamos carregando os pacientes com sucesso, mas não estamos ainda validando nenhum possível erro na requisição, trataremos disto no próximo exercício.

Nossa requisição AJAX funciona, porém não estamos levando em conta a possibilidade de acontecer um erro na requisição. Neste exercício vamos detectar este problema e exibir o erro para o usuário caso aconteça algo:

1- Para detectarmos se ocorreu algo, devemos utilizar o código de status da requisição HTTP, que pode ser obtido através da propriedade .status do XMLHttpRequest. Vamos testar se o código é 200, que significa que a requisição foi OK, caso contrário, vamos exibir os erros para os usuários:

var botaoAdicionar = document.querySelector("#buscar-pacientes");

botaoAdicionar.addEventListener("click", function() {

var xhr = new XMLHttpRequest();

xhr.open("GET", "https://api-pacientes.herokuapp.com/pacientes");

xhr.addEventListener("load", function() {

if (xhr.status == 200) {

var resposta = xhr.responseText;

var pacientes = JSON.parse(resposta);

pacientes.forEach(function(paciente) {

adicionaPacienteNaTabela(paciente);

});

} else {

//Exibiremos o erro aqui!

}

});

xhr.send();

});

2- Caso a requisição mostre qualquer outro código que não o 200, devemos exibir uma pequena mensagem de erro para o usuário. Esta mensagem erro será uma tag <span> que começará com a classe invisível, e que caso dê algum problema ela ficará visível. Crie a tag <span> que conterá a mensagem de erro acima da tag button:

</table>

<span id="erro-ajax" class="invisivel">Erro ao buscar os pacientes</span>

<button id="buscar-pacientes" class="botao bto-principal">Buscar Pacientes</button>

</section>

</main>

3- O primeiro passo é selecionar o <span> dentro do nosso código:

var botaoAdicionar = document.querySelector("#buscar-pacientes");

botaoAdicionar.addEventListener("click", function() {

var xhr = new XMLHttpRequest();

xhr.open("GET", "https://api-pacientes.herokuapp.com/pacientes");

xhr.addEventListener("load", function() {

//Adição aqui

var erroAjax = document.querySelector("#erro-ajax");

if (xhr.status == 200) {

erroAjax.classList.add("invisivel");

var resposta = xhr.responseText;

var pacientes = JSON.parse(resposta);

pacientes.forEach(function(paciente) {

adicionaPacienteNaTabela(paciente);

});

} else {

erroAjax.classList.remove("invisivel");

}

});

xhr.send();

});

4- Se a requisição retornar qualquer erro, devemos remover a classe invisivel do <span>, tornando a mensagem de erro visível para o usuário. Remova a classe invisivel caso o código não seja 200, e volte a adicionar caso a requisição não dê nenhum erro:

var botaoAdicionar = document.querySelector("#buscar-pacientes");

botaoAdicionar.addEventListener("click", function() {

var xhr = new XMLHttpRequest();

xhr.open("GET", "https://api-pacientes.herokuapp.com/pacientes");

xhr.addEventListener("load", function() {

var erroAjax = document.querySelector("#erro-ajax");

if (xhr.status == 200) {

erroAjax.classList.add("invisivel");

var resposta = xhr.responseText;

var pacientes = JSON.parse(resposta);

pacientes.forEach(function(paciente) {

adicionaPacienteNaTabela(paciente);

});

} else {

erroAjax.classList.remove("invisivel");

}

});

xhr.send();

});

Simule um erro, por exemplo removendo um pedaço da URL do servidor, e verifique se a mensagem de erro da tag *<span>*irá aparecer.

Neste capítulo aprendemos:

* Como fazer uma requisição assíncrona com o Javascript
* O objeto XMLHttpRequest
* Como configurar e enviar uma requisição
* O evento de load
* O que é a técnica AJAX.
* O formato JSON
* Como converter JSON para um objeto Javascript com a função JSON.parse()
* Como lidar com erros durante o AJAX