JAVASCRIPT

Manipulando a

DOM

Guia Prático

We Santos

Agradecimentos

A Deus, pela minha vida, e por me ajudar a ultrapassar todos os obstáculos que tenho encontrado ao longo do meu caminho.

Ao meu pai que sempre me mostrou o caminho certo, por mais difícil que deva ter sido.

A meus amigos, familiares e professores que direta ou indiretamente auxiliaram no desenvolvimento deste livro.

E por fim a Igor, Tiago e Bené, sem eles, nem este livro ou muito menos o desenvolvedor Waliston iriam existir.

Porque para mim tenho por certo que as aflições deste tempo presente não são para comparar com a glória que em nós há de ser revelada.
Romanos 8:18

Sumário

Sobre este Livro:

Do que se trata ?

Javascript Manipulando o DOM guia prático, é um condensado com o básico do que é mais utilizado por desenvolvedores web quando o assunto é manipular a dom.

A quem se destina ?

Este livro foi pensado para auxiliar desenvolvedores com alguma ou pouca experiência no desenvolvimento web, servindo como um índex para consulta e referência de pesquisa para assim dar seus primeiros passos na manipulação do DOM.

Preciso de algum Pré-requisito ?

È aconselhavél ter ciência do básico de html e css para assim você conseguir executar todo o código referente aos cápitulos e modifica-los a sua maneira

Os códigos utilizados nos exemplos estão disponíveis neste repositório

https://github.com/wal-wizard/javascrip-DOM

Introdução ao DOM

O DOM (Document Object Model) é a representação de dados dos objetos que compõem a estrutura e o conteúdo de um documento na Web. Neste guia, apresentaremos brevemente o DOM. Veremos como o DOM representa um documento HTML ou XML na memória e como você usa APIs para criar aplicativos e conteúdo da Web.

O que é o DOM ?

O Document Object Model (DOM) é uma interface de programação para os documentos HTML e XML. Representa a página de forma que os programas possam alterar a estrutura do documento, alterar o estilo e conteúdo. O DOM representa o documento com nós e objetos, dessa forma, as linguagens de programação podem se conectar à página.

Uma página da Web é um documento. Este documento pode ser exibido na janela do navegador ou como a fonte HTML. Mas é o mesmo documento nos dois casos. O DOM (Document Object Model) representa o mesmo documento para que possa ser manipulado. O DOM é uma representação orientada a objetos da página da web, que pode ser modificada com uma linguagem de script como JavaScript.

Os padrões W3C DOM e WHATWG DOM são implementados na maioria dos navegadores modernos. Muitos navegadores estendem o padrão; portanto, é necessário ter cuidado ao usá-los na Web, onde os documentos podem ser acessados por vários navegadores com diferentes DOMs.

Todas as propriedades, métodos e eventos disponíveis para manipular e criar páginas da Web são organizados em objetos (por exemplo, o objeto de document que representa o próprio documento, o objeto de table que implementa a Interface especial DOM HTMLTableElement para acessar tabelas HTML e assim por diante)

O DOM moderno é construído usando várias APIs que trabalham juntas. O DOM principal define os objetos que descrevem fundamentalmente um documento e os objetos dentro dele. Isso é expandido conforme necessário por outras APIs que adicionam novos recursos e capacidades ao DOM. Por exemplo, a HTML DOM API adiciona suporte para representar documentos HTML no DOM principal.

Chega de teoria certo ?

Então bora meter a mão no código, eu espero que você já esteja com o VScode ou seu editor de códigos favorito aberto com seu index.html rodando em um live server juntamente com um arquivo script.js já incorporado ao html, pois temos muita coisa para ver.

Capítulo 1

Capiturando elementos no DOM

Vamos começar pelo básico, a capitura de elementos no dom, por exemplo dentro de seu documento HTML você tenha alguma tag com Classes ou ID's e você que ter acesso a mesma utilizando o Javascript, existem diversas maneiras de fazer isso, mas 9 em cada 10 desenvolvedores utilizam essas duas formas aqui:

```
let elementName = document.getElementById()

let elementName = document.querySelector()
```

A Primeira opção ira selecionar os ID's presentes em nossos elementos, a segunda opção dará a liberdade de você "pegar" todos as tags do HTML, além de Classes e ID's

Eu quando vou desenvolver algum projeto, prefiro a segunda opção, pois dá a liberdade de utilizarmos não somente um ID, mas também seletores, Classes e tags.

Dito isso, utilizando a segunda opção, vamos nos aprofundar um pouco na captura de elementos.

Eis um problema, o querySelector irá selecionar apenas a primeira ocorrência de um elemento, se tivermos diversos itens com um mesmo nome de classe, por exemplo uma navbar com diversas tags Li com a classe .bg-red, como podemos selecionar todos esses elementos? Para isso temos uma maneira muito simples utilizando o próprio querySelector e aqui está:

```
let element = querySelectorAll(".bg-red");
```

Podemos ir mais além caso nosso intuito seja capturar atributos no HTML, podemos utilizar:

```
let element = querySelectorAll("[data-snackz]");
```

Que ira selecionar todos os elementos com o atributo [data-snackz]

Também podemos captuarar elementos que ainda não existem, isso mesmo você leu certo elementos que não existem. Se liga no exemplo abaixo:

```
let title = document.createElement("h1");
title.classList.add("bg-orange");
title.innerText = "Selecionado Classes que ainda não existem dentro do html"
```

Aqui eu acabei de criar um elemento dentro do JS que ainda não existe no HTML, porem isso não nos impede de captuar o nome da classe, então podemos fazer isso:

```
let elemNone = document.querySelector('.bg-orange');
```

Pronto, podemos estilizar ele com CSS, mas isso é assunto para outro Capítulo, vamos focar em selecionar os elementos.

Seria interessante para nós se pudessemos validar essas informações para ter certeza se tal emento existe certo?, não se preocupe podemos utilizar o match para isso, certo mas do que se trata o match você pode estar se perguntando bom ele é o seguinte:

Você utiliza ele para verificar se um elemento seria selecionado por um determinado seletor ou conjunto de seletores. Retorna true se o elemento for compatível e false quando não for

```
const element = document.querySelector('#sandwich');

if ( element.matches('.turkey') {
        console.log("deu match!");
} eles {
        console.log("não deu match :(");
}
```

Bom isso conclui nosso primeiro capítulo sobre seleção de elementos utilizando o JavaScript, O que eu posso deixar de conselho neste primeiro capítulo seria: foque em dominar a captura de elementos, isso fará toda a diferença para você no futuro, no começo será difícil lembrar de tudo, então sempre tenha uma "colinha" com as propriedades, lembrando que você não precisa decorar tudo, apenas saiba como cada coisa funciona, caso as dúvidas sejam muito grandes, o Mister G (Google) existe para ajudar você. Bons estudos, nos vemos no próximo capítulo.

Capítulo 2

Manipulando Arrays e Objetos

Este capítulo e mespecial irá mostrar a vocês, aplicações práticas de como iterar em um array ou objeto utilizando o FOR, FOR-IN, FOR-EACH. Fechando com BREAK, utilizado para dar um "stop" em nossos laços e o CONTINUE que como o nome sugere, irá fazer noss código prosseguir

Para dar início aos trabalhos, temos a seguinte estrutura:

```
let sandWiches = [
        "tuna",
        "ham",
        "turkey",
        "pb&j",
];
```

Começaremos utilizando o laço mais facil de todos o clássico FOR:

```
 for (let i = 0;  i < sandWiches.lenght; i++) {
        console.log(i)
        console.log(`${ i } - ${sandWiches[i]}`)
}
```

Podemos ir além e "aninhar" nossos laços for (útil quando trabalhamos com matrizes)

```
for (let i = 0;  i < sandWiches.lenght; i++) {
        for (let j = 0;  j < drinks.lenght; j++) {
              //do Something.....
        }
}
```

Para dar continuidade em nossos exemplos temos outra estrutura:

```
let lunch = {
        sandwich: "ham",
        snack: "chips",
        drink: "soda",
        desert: "cookie",
        guests: 3,
        alcohol: false,
}
```

Para esse cara iremos utilizar o FOR-IN, mas o que faz o for-in? Bom O laço interage sobre propriedades enumeradas de um objeto, na ordem original de inserção, ele pode ser executado para cada propriedade distinta do objeto.

```
for(let key  in lunch) {
        console.log(key)
}
```

Simples não?, mas ele é feito para de fato ser simples. Seguindo para o próximo exemplo, agora vamos entrar no BREAK e CONTINUE.

O CONTINUE, é util quando temos a intenção de "pular" certa propriedade, por exemplo:

```
let skippingLoop = ['turkey', 'tuna', 'ham', 'chiken salad', '&j'];
```

com a ajuda do FOR e uma dose de IF podemos fazer isso:

```
for (let index = 0; index < skippingLoop.length; index++) {
   if(skippingLoop[index] === 'ham') {
```

```
        continue;
    }
    console.log(skippingLoop[index]);
}
```

Então quando o laço iterar sobre a estrutura e a correspndência for encontrada, ele irá "pular" a mesma e seguir até o final. E se nosso intuito for parar de vez a execuçãoi do programa? Bom, aqui o BREAK entra na brincadeira desta forma:

```
const luch3 = {
    sandwich: 'ham',
    snack: 'chips',
    drink: 'soda',
    desert: 'cookie',
    guests: 3,
    alcohol: false,
};

for (const key in luch3) {
    if (luch3.hasOwnProperty.call(luch3, key)) {
        if (key === 'drink') {
            break;
        }
        console.log(luch3[key]);

    }
}
```

Neste exemplo, assim que a correspondência for encontrada, o programa irá interromper sua execução e mostrar os resultados no console.

Como os últimos serão os primeiros, agora falaremos daquele que é o terror dos Devs, o ForEach, mas o que é o ForEach ? Bom, O método executa uma dada função em cada elemento de um array. Como assim? , certo eis alguns exemplos:

```
const sandwiches2 = [
    'tuna',
    'ham',
    'turkey',
    'pb&j'
];

sandwiches2.forEach(function (sandwich, index) {
    console.log(index + " - " + sandwich);
    // console.log();
})


const lunch = {
    sandwich: 'ham',
    snack: 'chips',
    drink: 'soda',
    desert: 'cookie',
    guests: 3,
    alcohol: false,
};

Object.keys(lunch).forEach(function (item){
    console.log(`${item} - ${lunch[item]}`);
})
```

Nos dois exempolos deu para repar que o foreach executa o callback fornecido uma vez para cada elemento da ordem com um valor atribuido. Ele não é invocado para propriedades de índices que foram deletados ou que não foram inicializados (por ex. em arrays esparsos).

Bom cgegamos ao fim de mais um capítulo, espero que com muitas dúvidas, o que poss deixar aqui é o seguinte, eu sei que em primeiro contato o foreach pode assustar um pouco, mas acredite em mim e não desista, com o tempo você irá colher todos os frutos de seu esforço, até a próxima!

Capítulo 3

Utilizando get, set e remove

Dando continuidade aos nossos estudos, dígamos que um cliente aleatório nos contatou por e-mail para darmos manutenção em seu site focado em delivery, sejam comidas ou bebidas. Nosso trabalho como desenvolvedores é melhorar a visibilidade das promoções que vão aparecer no site, sendo assim foi nos passado uma lista das coisas que o cliente quer remover e outras a serem adicionadas. Com o problema em mãos e metade do pagamento na conta, podemos ir atrás da solução, que, na verdade, é mais simples do que se pode imaginar, para isso nossos amigos:

getAttribute(), setAttribute(), removeAttribute() e hasAttribute()

vão entrar em campo.

Começando pela nossa estrutura básica (que eu espero que vjá esteja pronta, pois eu quero que você teste, crie, destrua e crie novamente todos os exemplos do livro)

```
let elemento = document.querySelector('#lunch');
```

getAttribute() → retorna o valor de um argumento específico do elemento. Se o atributo não existir, o valor retornado será null ou "" (string vazia).

```
let sandwich = elemento.getAttribute(”data-sandwich'');
```

setAttribute() → Adiciona um novo atributo ou modifica o valor de um atributo existente num elemento específico.

```
elemento.setAttribute(”data-sandwich'', “turkey”);
```

removeAttribute() → remove um atributo de um elemento específico.

```
elemento.removeAttribute('data-chips');
```

hasAttribute() → Retorna um boolean indicando quando determinado elemento tem determinado atributo ou não.

```
if(elem.hasAttribute('data-drink')) {
    console.log('Add a drink!');
}
```

Lembrando que esses metodos também pode ser utilizados para manipular outros tipos de atributos como por exemplo: id, tabindex, name etc.

Bom chegamos ao final de mais um capítulo, este foi bem rápido não é mesmo?, esse conceito é bem simples de se entender, espero que você tenha entendido o que fora mostrado, e caso existam dúvidas, Mister G

Capítulo 4

Manipulando Estilos

Neste capítulo iremos tratar de estilo com JS, mas antes temos algumas coisas a tratar. Bom saiba que sim, é possível criar estilos com o Javascript, ele é muito bom nisso também, porém para estilizar nossos componentes já existe o CSS, ele é a unanimidade neste quesito, como a tarefa do javascript dentro da web e gerar interações com os usuários, o certo seria utilizar esses estilos no que diz respeito a animações, interações etc

E é nisso que esse capítulo se baseia, iremos manipular estilos visando interações com os usuários, então sempre tenha em mente que:

HTML → marcação
CSS → estilização

JS → interações

Primeiro de tudo vamos a algumas explicações, após isso vou mostrar um exemplo, como de costume já tenha seu editor de códigos aberto.

Nota: se seu exemplo for uma div, tenha certeza de ter dado uma altura e largura a ela.

```
let elemento = document.querySelector("#sandwish");
```

Vamos do básico, você tem um elemento e quer dar um fundo a ele, o que você utilizará sera:

```
elemento.style.backgroundColor = "putple";
```

Reparou algo diferente?, dentro do CSS utilizamos essa propriedade de oura forma, dentro do JS, utilizamos o padrão Camel Case na nomenclatura das propriedades.

Dentro do HTML DOM style Object existem diversas propriedades que podem ser utilizadas, vou deixar um link externo incrivel da W3 Schools com mais alguns estilos pra vocês testarem.

https://www.w3schools.com/jsref/dom_obj_style.asp

Agora vamos ao nosso exemplo, primeiro crie um documento index.html, copie e cole o conteúdo daqui: (eu recomendo fortemente que você digite todo ele):

```
<!DOCTYPE html>
<html>
    <head>
        <title>JavaScript Styles</title>
        <link rel="stylesheet" type="text/css" href="style.css" />
        <script type="text/javascript" src="script.js"></script>
    </head>

<body>
    <h1>JavaScript sample</h1>
    <div id="square"></div>
    <button type="button" onclick="doDemo(this);">Click Me</button>
</body>
</html>
```

Crie um novo arquivo, agora com o nome de styles.css, copie e cole o conteúdo daqui:

```
#square {
  width: 20em;
  height: 20em;
  border: 2px inset gray;
```

```
  margin-bottom: 1em;
}

button {
  padding: .5em 2em;
}
```

Agora por fim crie um novo documento com o nome de script.js, Copie e cole o conteúdo daqui:

```
function doDemo (button) {
  let square = document.getElementById("square");
  square.style.backgroundColor = "#fa4";
  button.setAttribute("disabled", "true");
  setTimeout(clearDemo, 2000, button);
}

function clearDemo (button) {
  let square = document.getElementById("square");
  square.style.backgroundColor = "transparent";
  button.removeAttribute("disabled");
}
```

Abra o documento no brawser e pressione o botão e veja a "mágica" acontecer, legal não é?.

Bom com esse exemplo, damos fim ao capítulo de estilos com JS, O que deixo de conselho ao final deste capítulo é o seguinte, esta parte de estilos você com certeza irá utilizar muito em sua vida como Dev, então perca bastante tempo indo atrás de coisas novas e criando muito, pois só assim você irá de fato cristalizar todos os conceitos e lembre-se, o uso dele estará restrito apenas a interações com seus usuários, divirta-se criando coisas novas e compartilhando com seus amigos. Até a próxima!

Capítulo 5

Utilizando add, remove, toggle em atributos

Neste capítulo, entraremos na santa trindade da manipulação do DOM, quando se trata de interações com usuários esses três tem seu lugar reservado no hall da fama. Add, Remove e Toggle, antes dos exemplos vamos ver o que cada um faz:

```
let elem = document.querySelector("#sandwish");
```

Add → Este método acrescenta um novo elemento com o valor especificado no final de um objeto Set

```
elem.classList.add("turkey");
```

Remove → Este método remove um elemento do DOM

```
elem.classList.remove("tuna");
```

Antes de entrarmos na parte mais importante deste capítulo, pode se fazer nescessário checarmos se certo elemento contem cerca classe, para isso nosso colega contains entra na batalha desta forma:

```
if(elem.classList.contains('mayo')) {
    console.log('add mayo!');
}
```

Agora partindo para o ponto alto deste capítulo, o top 1 de mais utilizados em operações no DOM, o Toggle, este método do DOMTokenList, é responsável por remover um elemento de nosso documento caso ele exista, e adicionar caso ele não exista, ele é muito utilizado em interações com menus quando, por exemplo, estamos criando um menu responsivo para dispositivos moveis e queremos um menu lateral que aparece quando o usuário clica em algum botão. Então estude bem este método, pois você usará ele muito em sua vida.

Sua sintaxe é simples podendo haver algumas variações e validações

```
elem.classList.toggle("tuna");
```

Agora seguindo para nosso exemplo final. Primeiro crie um documento index.html, copie e cole o conteúdo daqui: (eu recomendo fortemente que você digite todo ele):

```
<!DOCTYPE html>
<html>
    <head>
        <meta name="viewport" content="width=device-width, initial-scale=1">
    </head>
```

```
<body>

    <p>Clique no Botão</p>

    <button onclick="myFunction()">Aqui!</button>

    <div id="myDIV">
        O efeito ocorrerá aqui.
    </div>
</body>
</html>
```

Crie um novo arquivo, agora com o nome de styles.css, copie e cole o conteúdo daqui:

```
  width: 100%;

  padding: 25px;

  background-color: coral;

  color: white;

  font-size: 25px;

  box-sizing: border-box;
```

Agora por fim crie um novo documento com o nome de script.js, Copie e cole o conteúdo daqui:

```
function myFunction() {

  let element = document.getElementById("myDIV");

  element.classList.toggle("mystyle");

}
```

Então clique algumas vezes no botão para ver a “mágica” acontecer.

Bom com este exemplo chegamos ao fim a mais um capítulo, e com ele chegamos quase ao final do livro, o próximo capítulo será o último, porém isso é o próximo, agora estamos aqui, e trataremos da nossa trindade do DOM, aqui como nos outros capítulos, vou deixar como conselho, teste muito, faça muitos códigos e "respire" esses métodos, pois com certeza eles vão te acompanhar por muito tempo. Até a próxima!

Capítulo 6

Utilizando Event Listeners

Bom aqui estamos com mais um capítulo e outro tema extremamente importante em sua jornada como desenvolvedor, aqui trataremos dos event listeners, antes do código , um piquinho de teoria, o que fáz o Event Listener? Bom:

O addEventListener() registra uma única espera de evento em um único alvo. **O Alvo do Evento** *pode ser um único* **elemento** *em um documento, o* **Documento** *em si, uma* **Janela** *,ou um* **XMLHttpRequest**.

Para registrar mais de uma espera de evento como alvo, chame addEventListener() para o mesmo alvo mas com diferentes tipos de evento ou captura de parâmetros.

Existem diversos tipos de eventos disponíveis para usarmos, deixarei alguns links com uma coletânea para que você se divertir testando todos e aprendendo, nos próximos exemplos darei enfase no evento de **Click,** *que segue sendo um dos mais utilizados. Junto com: onchange, onmouseover, onmouseout, onkeydown, onload.*

https://www.tutorialspoint.com/javascript/javascript_events.htm

https://www.w3schools.com/js/js_events.asp

Sintaxe

```
alvo.addEventListener(type, listener[,options]);
```

ou seja:

```
let element = document.querySelector("body");
element.addEventlistener("click", function(){
        //seu código
})
```

No exemplo acima, já fica comprovado o quão simples é utilizar um event listener, nos próximos exemplos mostrarei ainda mais, vamos começar por capturar o elemento que ira escutar o evento (geralmente um botão).

```
const  btn = document.querySelector("#click-me");

btn.addEventListener('click', function(event) {
        console.log(event);
        //Mostrará os detalhes do event.
        console.log(event.target)
```

```
    //O elemento que fora clicado.

})
```

Sim, eu usei uma function, você pode utilizar uma arrow function sem problemas também, porém tenha em mente que fazendo isso, o debug pode ser meio complicado para ser feito, mas vai da escolha, eu geralmente uso arrow functions, mas para os exemplos decidi variar um pouco.

Seguindo, pode passar por sua cabeça, "E se eu tiver muitos eventos no meu documento eu posso utilizar um laço para "invocar" todos?", sim, você pode, porem por motivos de desempenho não é aconselhável, veja o exemplo a seguir:

```
const btns2 = document.querySelector(".click-me");

Array.from(btns2).forEach(function(btn) {

    btn.addEventListener('click', function(event) {
        console.log(event);
        console.log(event.target);
    })
})
```

O exemplo acima irá funcionar sem nenhum problema, porem como já fora dito, por questões de desempenho, é extremamente desacoselhado o seu uso. Felizmente, existe uma maneira muito fácil e eficiente de fazer um loop sobre cada elemento, para isso temos: delegação de eventos ou bubbling de eventos.

Também é possivél adicionar eventos em todo nosso documento HTML, capturando a window por exemplo:

```
window.addEventListener('click', function(event) {
```

```
        if(event.target.matches('click-me')) {
            //seu código
        }
})
```

Isso é útil para quando temos a necessidade de impedir que a janela faça reload após alguma ação ou forçar um reload por exemplo.

Também podemos ao invés de ter a função declarada dentro evento, chamar uma função que já existem dentro do nosso código javascript, isso nos dá a possibilidade de reutilizar essa função e esse código em diversos locais diferentes sem ter a necessidade estar sempre criando códigos e mais códigos para executar a mesma tarefa. Dentro da programação existe um termo muito utilizado, o DRY do inglês Don't repeat yourself, ou seja, não se repita, porque você iria escrever múltiplos códigos em funções diferentes que fazem o mesmo, quando você pode ter apenas uma função, e chamar a mesma em diversos eventos diferentes? Veja no segue o exemplo abaixo:

```
let formContact = document.querySelector("#form-control");

const logTheEvent = function(event) {
        console.log(`O seguionte evento está acontecendo ${event.target}`)
}

document.addEventListener('click',  logTheEvent);
window. addEventListener('scroll',  logTheEvent);
formContact.addEventListener('focus', logTheEvent);
```

Abra seu console e veja todos os respectivos eventos que estão ocorrendo dentro da nossa página ser mostrado, agora vamos ao nosso exemplo final:

Primeiro dentro do seu documento index.html copie e cole o seguinte código(sempre vou aconselhar você a digitar todo o código, vai por mim, você irá me agradecer futuramente por isso).

```
<div class="wrapper">
        <div id="message" class="hide">
                Mensagem oculta, que a força esteja com você jovem padawan!
        </div>
        <div id="button">Click me!</div>
</div>
```

Agora precisamos de dar algunm "toque de requinte" em nosso exemplo, e o resposnsável por isso sera nosso bom e velho CSS, eis aqui o código que você irá precisar:

```
.wrapper{
  position: relative;
  top: 40vh;
}
#button{
  border: 5px solid #FFAB33;
  width: 200px;
  text-align: center;
  padding: 10px;
  font-size: 30px;
  cursor: pointer;
  margin: auto;
  background: #FF5733;
}

#button:hover{
  background: #DFF30B;
```

```
}

#message{
  font-size: 26px;
  margin-bottom: 30px;
  text-align: center;
}

.hide{
  visibility: hidden;
}

.reveal{
  visibility: visible;
}
```

Para finalizar, precisamos de fato adicionar os eventos a nossa página, e como já sabemos, o JS faz isso muito bem, então vamos aos scripts:

```
let button = document.querySelector('#button');
let msg = document.querySelector('#message');

button.addEventListener('click', ()=>{
  msg.classList.toggle('reveal');
})
```

Agora basta recarregar sua página e clicar no botão, note que com algumas linhas, você já executou algo bem interessante.

Com esse exemplo, damos fim ao capítulo de Event Listeners e com isso o livro também, o que deixo de conselhos para fecharmos esse pequeno tour pelos eventos do javascript é o seguinte:

Em sua vida como desenvolvedor você irá passar por vários altos e baixos, tempos em que você será extremamente produtivo, outros que você ira ficar

procrastinando e às vezes até se perguntando se isso é mesmo para você, acredite é. Programação não tem que ser um bicho de 7 cabeças como a maioria passa em cursos ou vídeos, também sabia que não sera uma área facial com salários estrondosos com pouco tempo de estudo, é necessária muita dedicação e uma superação diária, não é você contra outros Devs, e sim você contra você, tenha foco e se prepare, pois o futuro pertence àqueles que se preparam para isso hoje. Até a próxima.

Referencias Bibliograficas

https://www.devmedia.com.br/trabalhando-com-eventos-em-javascript/28521

http://www.linhadecodigo.com.br/artigo/3617/eventos-em-javascript-tratando-eventos.aspx

https://desenvolvimentoparaweb.com/javascript/eventos-javascript/

https://www.computersciencemaster.com.br/eventos-com-javascript/

https://www.freecodecamp.org/portuguese/news/tutorial-sobre-button-onclick-em-html-e-evento-de-clique-em-javascript/

https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks/Events

https://www.ic.unicamp.br/~celio/inf533/docs/eventos_javascript.html

https://cfbcursos.com.br/javascript-32-eventos/

https://www.javascriptprogressivo.net/2019/01/Eventos-em-JS-O-que-Sao-Para-que-servem.html

https://www.javatpoint.com/javascript-events

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events

https://www.digitalocean.com/community/tutorials/understanding-events-in-javascript

https://eloquentjavascript.net/15_event.html

https://www.freecodecamp.org/news/event-handling-in-javascript-with-examples-f6bc1e2fff57/