**jQuery**

**Introdução**

Biblioteca JavaScript com a proposta de: escreva menos e faça mais;

Recursos:

Seleção e manipulação de elementos HTML,

Navegação pelo DOM,

Manipulação de CSS,

Eventos: click sobre botões, entrada de texto no input,

Efeitos e animações: efeitos já prontos; ex.: ocultar ou exibir elemento HTML,

Ajax: realização de requisições assíncronas,

Plugins: podem ser reaproveitados dentro das aplicações; por exemplo: ao invés de criar uma galeria de imagens, pode ser utilizado um plugin que já tem essa finalidade.

**Configurando e testando o jQuery**

No terminal: **npm i jquery;**

**jquery.com > https://code.jquery.com/jquery-3.7.1.min.js(salvar link no projeto em que o jquery será utilizado).**

**CDN** – conectar com o link do j Query**:**

[https://releases.jquery.com](https://releases.jquery.com/) > minified > CTRL + C > Copia na tag script

**Seletores – Selecionando elementos HTML parte 1**

Dependendo do objetivo para modificar elemento, eles precisam ser selecionados.

Selecionando elementos por tag - **$(tag)**

// // // classe - **$(.classe)**

// // // id - **$(#id)**

**$** - permite selecionar elementos, acesso aos recursos da biblioteca jQuery;

- permite executar ações da biblioteca jQuery;

- permite o acesso aos recursos da biblioteca jQuery.

**Seletores – Selecionando elementos HTML parte 2**

Praticando a seleção de elementos de um modo um pouco mais avançado.

**-** Combinação de seletores do jQuery;

**-** Como atuar sobre o array de retorno da seleção de elemento.

  //selecionar elementos por classe de tag específica

      console.log($("li.exemplo1"));

retorna a classe **exemplo1** quando ela estiver dentro de um elemento **li**, caso exista **mais** **de** **uma li** com a mesma classe, **retorna** também;

      //selecionar elementos por classe de classe específica

      console.log($(".lista1 .exemplo2"));

retorna os elementos de **qualquer tag** **com** a classe **lista1** que **possui elementos** **com** a **classe** **exemplo2**

      console.log($(".lista1 div.exemplo2"));

retorna os elementos de **qualquer tag com** a classe **lista1** que **possui** elementos **div com** a **classe exemplo2**

//atuação sobre o array de retorno dos elementos selecionados

**:first 🡪** retorna primeiro elemento da seleção

**:last 🡪** retorna último elemento da seleção

**:even 🡪** retorna elementos contidos nos índices pares

**:odd 🡪** retorna elementos contidos nos índices ímpares

Alguns elementos que permite **reaplicar filtro** **no retorno de elementos já filtrados**.

**Sequência de execução de código e $(document).ready()**

*Formas tradicionais com JS puro*

<body onload =”funcao()” />

windows.onload = function() { ... }

*Utilizando jQuery*

$(document).ready(function() { ... })

$(function() { ... })

- Essas, são 2 formas de implementar o controle do momento em que o código JS entra em ação.

$(document).ready(function () {

        //'construi' código javascript no momento em que a página está carregada

        function teste() {

          console.log($("#exemplo"));

        }});

      $(teste); //nao foi colocado o parênteses após a função, pois não será passado nenhum parâmetro, a função está apenas sendo referenciada

$(function () {

        //passando direto a função como parâmetro

        console.log($("#exemplo"));

      });

**Recuperando e manipulando os atributos dos elementos HTML**

*<tag attr1=”alguma coisa” attr2=”alguma coisa”></tag>*

**attr – atributos**

Padrão do plano de fundo

Descrição gerada automaticamente com confiança baixa

recuperando atributo ‘**src**’ da tag **img.**

- Um segundo valor, é um valor que iremos atribuir para aquele atributo

Forma

Descrição gerada automaticamente com confiança média

Nesse exemplo, estava sendo exibida a imagem1 e mudamos para imagem2.

        $("div").attr("style", "background-color:blue", "width:100");//recuperando atributo style da tag div e atribuindo 2 valores a ela

**Recuperando e manipulando o conteúdo interno de elementos HTML**

- Elemento interno, diferente de value

- .html 🡪 método que define ou retorna o conteúdo dos elementos selecionados

*<tag attr=”alguma\_coisa”></tag>*

*<tag attr=”alguma\_coisa”>... conteúdo inerno ...</tag>*

Padrão do plano de fundo

Descrição gerada automaticamente com confiança baixa

Modificando ***valor* *interno***da div1.

Padrão do plano de fundo

Descrição gerada automaticamente com confiança baixa

**Tags** também **podem ser adicionadas**, nesse exemplo é a tag strong.

Forma

Descrição gerada automaticamente

Adicionando style.

Padrão do plano de fundo

Descrição gerada automaticamente com confiança baixa

Método **text** recebe string e mesmo que ela contenha uma tag HTML válida, **atribui o texto puro**.

**Recuperando e manipulando valores de inputs (text) e selects**

.val() – método que definir ou obtém valor de elementos de formulário

Padrão do plano de fundo

Descrição gerada automaticamente com confiança baixa

Recuperando valor por id

Padrão do plano de fundo

Descrição gerada automaticamente com confiança baixa

Modificando valor pelo Id

**Recuperando e manipulando valores checkboxs/radios e Loop com $.each()**

Forma

Descrição gerada automaticamente

Recuperando classe ‘sexo’ que está dentro do input 🡪 ‘Feminino’;

Recuperando valor checked (verificado, que foi selecionado) da classe ‘sexo’.

Texto

Descrição gerada automaticamenterecuperando valor contido dentro do elemento ‘radio’ que está checked.

Para um melhor entendimento sobreo valor (value) e nome (name) do elemento:

Tela preta com letras brancas

Descrição gerada automaticamente

Recuperando elementos que contém a classe ‘interesse’:

Texto

Descrição gerada automaticamente

Recuperando apenas os elementos que contém a classe ‘interesse’ que estão selecionados:

Texto

Descrição gerada automaticamente

A diferença é que enquanto o **radio só permite** a seleçãode **uma opção** do conjunto, o **checkbox permite** a seleção **de 1 ou mais opções**.

Caso haja a tentativa de retornar **valores do checkbox**, **só retorna o primeiro**:



Método $.each() retorna 1 array e 1 função de call-back e o índice e o valor de cada elemento do array

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamenterecuperando diretamente o atributo value.

**Inserindo e removendo elementos HTML**

Feito dentro de:

      $(document).ready(() => {});

**Inserindo** como último item da ‘lista1’ 🡪 ‘li – Item 4’

      $("#lista1").append("<li>Item 4</li>");

**Inserindo** como último item da ‘lista1’ 🡪 ‘li – Item 0’

        $("#lista1").prepend("<li>Item 0</li>");

**Inserindo** como 1° item da lista 2 uma li com o conteúdo interno ‘Item novo’

          $("#lista2").prepend("<li>Item novo</ li>");

A diferença entre after() e prepend(), é que os elementos adicionados com o **.after** estão no **mesmo nível** hierárquico **que o elemento alvo**, já os elementos adicionados com **.prepend** é ‘**filho’ do elemento alvo**.

Ex.:

É adicionado um elemento HTML com o **.after**;

Esse elemento será adicionado após a tag que contém a classe utilizada.

Se esse elemento é adicionado com o **.prepend**;

Esse elemento ficará após os elementos já existentes, porém, dentro da tag que contém a classe.

Diagrama

Descrição gerada automaticamente

        $("#lista2").remove(); //removendo item

**Navegando entre elementos HTML**

**parent ()** 🡪 elemento pai

Texto

Descrição gerada automaticamenterecuperando elemento pai da tag que contém a classe ‘secao1’:

<div id="pagina">

**closest ()** 🡪 procura por elementos pais

seleciona o primeiro elemento que ele encontra debaixo para cima

Texto

Descrição gerada automaticamenteretorna 1° elemento pai de ‘.subSecao’ que contém o Id ‘#pagina’.

Texto

Descrição gerada automaticamenteacessando elementos pai de ‘.item’ que tem a tag ‘ul’.

Diferença entre **parent()** e **closest()**, é que enquanto closest procura até achar a referência solicitada, parent vai direto no elemento pai.

**find ()** 🡪 procura por elementos filhos

Texto

Descrição gerada automaticamenterecupera elementos pais de ‘#topo’ que contém a tag ‘li’.

Texto

Descrição gerada automaticamente recupera elementos pais de ‘#rodape’ que contém a tag ‘p’.

Texto

Descrição gerada automaticamente’secao1’ sobe um nível e recupera elemento filho ‘h1’

Texto

Descrição gerada automaticamenteseleciona elemento pai de ’secao1’ que contém ‘#pagina’ e seleciona elemento filho ‘h1’

**Manipulando CSS**

- Manipular estilos visuais dos elementos HTML

- .css() 🡪 adiciona propriedades de estilo inline:

**$(‘<elemento-estilizado>’).css(‘atributo-estilo’, ‘valor’)**

Essa **estilização pode ser feita** através do ***id****,* ***class*** e até mesmo a***tag.***

Ex.: estilizando topo



Adicionando mais de um atributo de estilo:

Texto

Descrição gerada automaticamente

Adicionando mais de um valor no mesmo método:

Basta adicionar um objeto literal: $(‘’).css(**{**‘ ’**:**’ ’**,** ‘ ’**:**’ ’**}**);

Texto

Descrição gerada automaticamente

modificando item através da tag ‘h1’

.addClass – adiciona uma classe

adicionando classe ‘.campo’ aos inputs

Texto

Descrição gerada automaticamenteatribuindo 2 classes de uma só vez

hasClass() 🡪 verifica a existência de uma classe no elemento html

Interface gráfica do usuário, Texto

Descrição gerada automaticamenteverificando se dentro de ‘textarea’, existe as classes ‘campo’ e ‘erro’

removeClass() 🡪 remove classes

Texto

Descrição gerada automaticamente

**Eventos**

- Envio de formulário;

- Entrada de texto em algum campo;

- Scrow (rolagem) sobre a janela;

- Movimento do mouse;

- Qualquer interação do usuário com a página web.

**Eventos – browser (navegador)**

**onload** – executa código JS assim que o carregamento da página é feito;

**Associando eventos a janela do navegador (window):**

        $("#onload").html("Página carregada");

Assim que página for carregada, texto será exibido no elemento que contém o “#onload”

$(window).scroll(() => {

          $("#scroll").html("Scroll acionado");

        });

Scroll associado ao window (objeto que guarda o estado da janela do navegador);

Através da seleção da janela, executa o método scroll passando uma ação que é a seleção do elemento que contém o id ‘scroll’, passando um texto

$(window).resize(() => {

          $("#xyz").html("Página redimensional");

        });

Resize associado ao window (objeto que guarda o estado da janela do navegador);

Texto passado para o elemento que contém o id ‘xyz’ é executado cada vez que a janela é redimensionada

$("#xyz")

            .html("Página redimensionada")

            .css("background-color", "red");

        });

Adicionando um css ao elemento do id a cada vez que ele for exibido (quando janela é redimensionada)

**Associando eventos a elementos HTML:**

$("#div1").scroll(function () {

          $(this).css("background-color", "blue");

        });

**this** – referência objeto em questão, passado acima, assim, não é necessário colocar novamente seu nome;

Atuando sobre próprio elemento

Com arrow function:

$("#div1").scroll(() => {

          $('#div1').css("background-color", "blue");

OU

 $("#div1").scroll((e) => {

          $(e.target).css("background-color", "blue");

        });

Atuando sobre elemento alvo com ‘e’

**Eventos do mouse**

**mousedown()** – quando o ponteiro do mouse está sobre o elemento e o botão do mouse é pressionado;

**mouseup()** – quando o ponteiro do mouse está sobre o elemento e o botão do mouse é liberado;

**click()** – quando o ponteiro do mouse está sobre o elemento e o botão do mouse é pressionado e liberado.

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente com confiança média**click** e **mouseup** tem quase a **mesma função**, pode ser utilizado qualquer um, pois são **executados no mesmo momento**.

**dblclick()** – double click, dois clicks no mouse.

Interface gráfica do usuário, Texto

Descrição gerada automaticamente com confiança média

Adiciona background amarelo na ‘div2’, quando ‘btn2’ for clicada duas vezes

**mousemove()** – movimento do mouse sobre um determinado elemento HTML;

base de efeitos em que clicamos no elemento HTML e o realocamos em outros lugares;

nos permite capturar as coordenadas do elemento conforme o movimento feito dentro desse elemento, fazendo com que possamos criar efeitos interessantes.





**mouseenter()** e **mouseover()** – ocorre quando um elemento HTML é sobreposto pelo cursor do mouse, quando entramos na região visual de um elemento.

 - mesma função do mouseover

**mouseleave()** e **mouseout()** – ocorre na saída do cursor do mouse sobre a região do elemento HTML renderizado no navegador(basicamente o contrário de **mouseenter** e **mouseover**).

Texto

Descrição gerada automaticamente

Esses eventos se comportam de formas diferentes, quando estão em um cenário em que esses eventos são capturados dentro de uma hierarquia de elementos HTML.

mouseover / mouseout (antigos)

mouseenter / mouseleave (novos)

Um exemplo da confusão que pode haver:

Texto

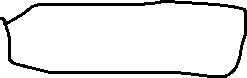
Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo, Word

Descrição gerada automaticamente



Esse mesmo processo com os métodos **mouseenter** e **mouseleave**, ocorre normalmente:

Interface gráfica do usuário, Aplicativo, Word

Descrição gerada automaticamente



**Eventos do teclado**

**keydown()** – pressão da tecla

**keyup()** – liberação da tecla

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Caso ocorra da tecla ficar pressionada por um tempo, o número de registro do keydown aumenta, mas o registro do keyup só ocorre quando a tecla é liberada de fato;

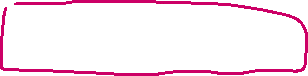
Isso, pode influenciar o momento em que o evento deve ser disparado, momento em que é executada uma ação em função da pressão de uma tecla.

Uma informação útil em que o evento captura é o **keyCode** (código da tecla que foi pressionada).

Fazendo testes:

Texto

Descrição gerada automaticamente executando ação de acordo com o n° keyCode.



Interface gráfica do usuário, Texto

Descrição gerada automaticamente assim que enter for pressionado, texto que estiver dentro do input, vai para o html do span.



**Eventos de formulário**

**focus()** – o foco sobre determinado elemento html;

podemos afetar visualmente algum campo quando ele recebe ou perde o foco

**blur()** – retirada do foco de um determinado elemento html;

**change()** – modificação de um valor (select, radio, checkbox), podemos capturar os eventos de mudança que ocorrem nesse elemento e trabalhar com eles;

**submit()** – captura o envio de um formulário.

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

**Registrando/removendo eventos com on/off**

**on () / off ()** – registra e/ou remove eventos para os elementos HTML mesmo que eles sejam criados posteriormente a chamada função.

O problema de associação de eventos a elementos HTML que são criados posteriormente a execução dessa associação, tentou ser resolvidos de algumas formas diferentes:

**live ()** – até versão 1.7;

**delegate ()** – descontinuado na versão 3;

**bind ()** – descontinuado na versão 3;

$("body").on("focus", "input", (e) => {

          //qualquer elemento do tipo input dentro de body, recebe essas definições, mesmo que seja, criados de forma programatica

          $(e.target).removeClass("desfocado");

          $(e.target).addClass("focado");

        });

Pode ser feita a alteração diretamente ao elemento ou passando o elemento pai.

removendo o efeito passado ‘blur’ de todos os inputs dentro do body