Desenvolvimento de Sistemas

Bibliotecas de *front-end*: tecnologias, aplicabilidade e jQuery

Em *front-end*, é possível utilizar diversas bibliotecas, que são coleções de recursos usados por programa de computador. Pode-se dizer que, quando se está utilizando uma linguagem de programação e aplica-se uma biblioteca, tem-se acesso a um conjunto de funções que já foram escritas por desenvolvedores.

As principais vantagens de utilizar biblioteca são: funções predefinidas; mais agilidade, pois é possível usar programas distintos; códigos menores e mais organizados, o que reduz as chances de erros de códigos; e facilidade de atualização de programas diversos.

Merecem destaque as bibliotecas Bootstrap, jQuery, Angular e React, porém existem diversas outras.

Tecnologias

Bootstrap

Figura 1 – Logo do Bootstrap
Fonte: Bootstrap ([s. d.])

Bootstrap é uma coleção gratuita e *open-source* de ferramentas para criação de *websites* e aplicações *web*. Ele contém modelos de *design* baseados em CSS (*cascading style sheets*) e HTML (*hyper text markup language*) para tipografia, formulários, botões, navegação e outros componentes de interface, bem como extensões opcionais JavaScript. Seu objetivo é facilitar o desenvolvimento de *websites* dinâmicos e aplicações *web*.

O Bootstrap, originalmente chamado Twitter Blueprint, foi desenvolvido por Mark Otto e Jacob Thornton no Twitter, como um *framework* para encorajar consistência entre os projetos internos da empresa. Antes do Bootstrap, muitas bibliotecas foram usadas no desenvolvimento da interface, o que levou a inconsistências e alto custo de manutenção.

O Bootstrap é compatível com as últimas versões dos navegadores Google Chrome, Firefox, Internet Explorer, Opera e Safari. Desde a versão 2.0 tem *design* responsivo, e a partir da versão 3.0 adotou a filosofia *mobile-first*, que prioriza a construção de aplicações *web* e *sites* com boa aparência mesmo em dispositivos móveis.

jQuery

Figura 2 – Logo do jQuery
Fonte: jQuery (c2023)

O jQuery é uma biblioteca gratuita que oferece um conjunto de funcionalidades que simplificam bastante a programação JavaScript, deixando-a mais intuitiva, clara e fácil. É importante destacar que o jQuery não é uma linguagem, na realidade ele é um conjunto de funções em JavaScript.

O jQuery conta com uma API (*application programming interface*) intuitiva e de fácil leitura (o que torna a curva de aprendizado da linguagem mais suave), que executa em qualquer *browser* sem a necessidade de *hacks* em JavaScript para adaptar o código para cada navegador. Além disso, os usuários da API se beneficiam também com a liberação de novas versões do jQuery, pois a cada novo *release*, seu *site* é automaticamente atualizado, bastando apenas a substituição do arquivo de importação do jQuery.

Antes de seguir adiante, veja um comparativo simples de uma funcionalidade em JavaScript puro e em jQuery. O código a seguir captura um elemento de uma página que tenha o atributo **id** igual a **"menu"**.

```
// JavaScript
var menu = document.getElementById("menu");
// jQuery
var menu = $("#menu");
```

Como é possível ver no exemplo, o jQuery consegue executar a mesma funcionalidade do JavaScript com menos código e ainda consegue deixar mais claro e legível o código. Em resumo, o jQuery possibilita:

- Manipular facilmente o conteúdo de uma página *web*.
- Trabalhar com eventos de maneira bastante clara e intuitiva.
- Adicionar efeitos especiais.
- Manipular um conjunto de elementos de uma página com apenas uma linha de código.
- Utilizar uma grande quantidade de *plugins* disponíveis para diversos tipos de necessidades de desenvolvimento *web*, tais como: *menus*, *slides*, animações, gráficos etc.

A filosofia do jQuery sintetiza o que se observa nos exemplos anteriores, ou seja: “escreva menos, faça mais”. A economia de linhas de código é ainda mais significativa em *sites* profissionais, quando arquivos *scripts* chegam à casa dos milhares de linhas de código.

Realizando o *download* da biblioteca do jQuery

O primeiro passo é fazer *download* da biblioteca do jQuery no *site* oficial (para isso, basta digitar *“Download* jQuery” em seu buscador). Aqui, será usada a versão *uncompressed* do jQuery. Essa versão é geralmente utilizada para a fase de desenvolvimento de *sites*, pois permite ler o código do jQuery caso haja necessidade. A versão *compressed* é uma versão compactada, com remoção dos espaços em branco. Deve ser utilizada quando o *site* é publicado, pois seu tamanho é menor, o que torna o *download* da página mais rápido.

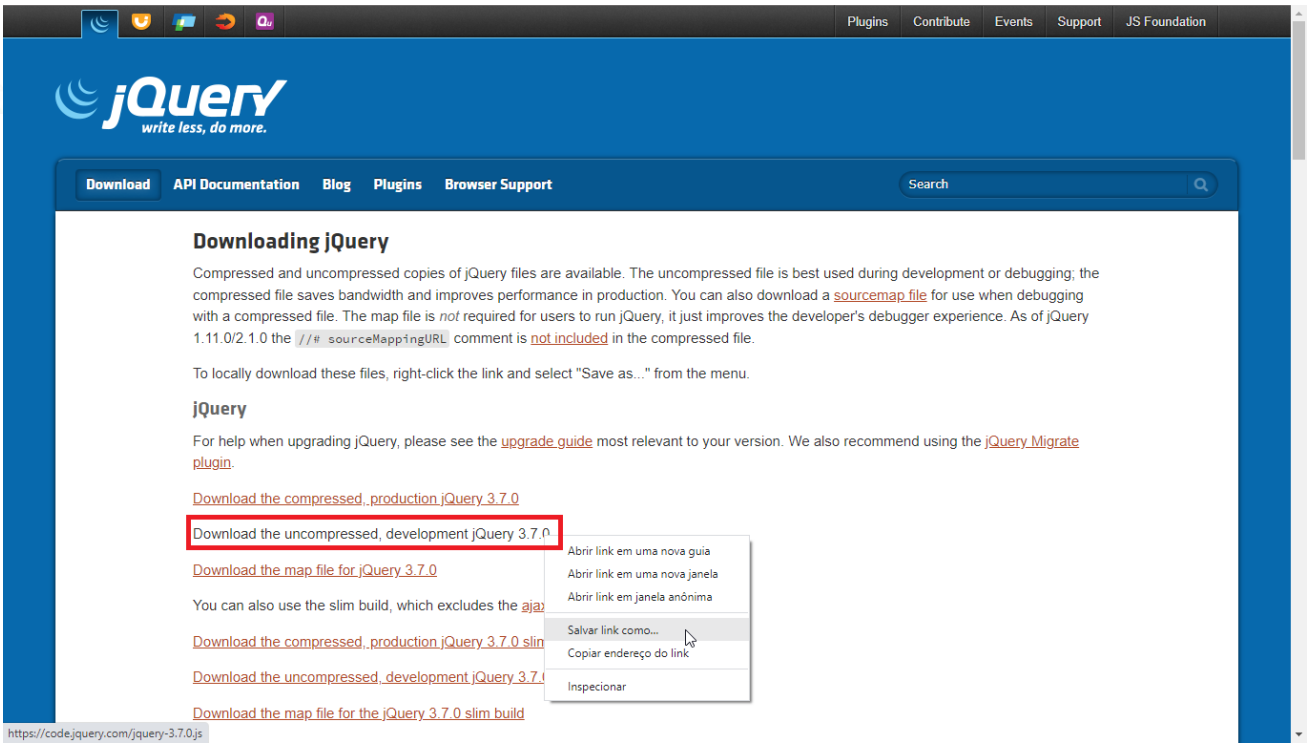

Figura 3 – Página de *downloads* do jQuery

Fonte: Senac EAD (2023)

Para baixar o arquivo, clique com o botão direto no *link* e escolha **Salvar conteúdo do link como**.

Com o arquivo baixado (**jquery-3.x.x.js**, por exemplo), renomeie ele para **jquery.js** para manter a simplicidade dos exemplos. Para incorporar o jQuery em sua página HTML, é preciso realizar a importação da biblioteca **jquery.js** com o seguinte comando:

```
<script src="jquery.js"> >/script>
```

Observação: é importante salientar que o arquivo **jquery.js** deve estar na mesma pasta onde está o seu arquivo HTML. Caso contrário, o comando acima precisará ser adaptado para conter o caminho do arquivo. Sugere-se que siga esse passo a passo até conseguir a confirmação de que o jQuery está sendo executado corretamente.

Outra alternativa ao *download* da biblioteca é usar um servidor CDN (*content discovery network*). Com o CDN, é possível carregar a biblioteca do jQuery diretamente da Internet por meio dos servidores que hospedam o *script*, sem a necessidade de baixar o arquivo **JS (.js)** para a pasta do seu projeto ou *site*. Observe o código a seguir:

```
<script src="https://code.jquery.com/jquery-3.6.3.js" integrity="sha256-nQLuAZGRRcILA+6dMBOvcRh5Pe310sBpanc6+QBmyVM=" crossorigin="anonymous"> </script>
```

Nesse caso, a biblioteca do jQuery será baixada do *site* <code.jquery.com>. A biblioteca é apenas um arquivo **JS (.js)**. A vantagem do uso do CDN é que os servidores estão distribuídos por várias regiões do mundo, o que torna mais eficiente o seu *download* por usuários de diferentes países.

O código acima pode ser copiado em <thttps://releases.jquery.com>, clicando na versão do jQuery que deseja em *uncompressed*.

É recomendado colocar os *scripts* antes do fechamento da *tag* **<body>**, conforme o exemplo:

jQuery via CDN	jQuery com arquivo local

`<script src="https://code.jquery.com/jquery-3.6.3.js" integrity="sha256-nQLuAZGRRcILA+6dMBOvcRh5Pe310sBpanc6+QBmyVM=" crossorigin="anonymous"> </script>`
`</body>`
`</html>`
	`<script src="jquery.js">`
	`</script>`
	`</body>`
	`</html>`

Aplicabilidade: jQuery na prática

O jQuery tem apenas uma função de entrada que você precisará chamar toda vez que necessitar algum recurso da API. O nome dessa função é jQuery. Porém, em uma página *web*, é comum chamar essa função dezenas ou centenas de vezes. Pensando nisso, foi criado um **álias** para essa função, por meio do símbolo **$**, a fim de economizar código. Observe o código a seguir:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
```

```
          <meta charset="UTF-8">
          <meta http-equiv="X-UA-Compatible" content="IE=edge">
          <meta name="viewport" content="width=device-width, initial-scale=1.0">
          <title>Alerta da Versão do jQuery</title>
        </head>
        <body>
          <h3>Aparecerão dois pop-ups no topo da tela informando a versão do jQuery! </h
3>

          <script src="jquery.js"></script>
          <!--
            As duas linhas abaixo são equivalentes,
            a única diferença é que a primeira usa a função jQuery
            e a segunda usa o seu álias.
          -->
          <script>
            alert('versão do jQuery: ' + jQuery().jquery)
            alert('versão do jQuery: ' + $().jquery)
          </script>
        </body>
      </html>
```

As duas chamadas – **jQuery().jquery** e **$().jquery** – são equivalentes, a única diferença é que a primeira usa a função jQuery e a segunda usa o seu álias.

Uma típica declaração, ou comando, do jQuery é composta de quatro partes:

1. A função jQuery ou seu álias
2. O(s) seletor(es)
3. A(s) ação(ões)
4. O(s) parâmetro(s)

Exemplo:

```
$("p").css("background-color","#FFFFFF");
```

É possível dividir nas seguintes partes:

Função jQuery	Seletor	Ação	Parâmetros

$	(“p”)	.css()	"background-color","#FFFFFF"

- A função jQuery é necessária para as funcionalidades.
- O seletor permite definir quais elementos serão selecionados.
- A ação aplica algum método ou função do jQuery sobre os elementos selecionados.
- Os parâmetros definem as propriedades que serão aplicadas na ação.

A partir de agora, sempre que você usar o símbolo **$()**, estará, na realidade, chamando a principal função da API, que é a função **jQuery()**.

Na maioria das vezes em que se usa jQuery, elementos HTML serão manipulados. Para garantir que todo o código HTML e CSS tenha sido completamente carregado antes de executar uma função jQuery, é uma boa prática usar a função **ready()** do jQuery para garantir que o código poderá manipular qualquer elemento HTML carregado na árvore DOM (*document object model*). Caso uma manipulação de elementos seja executada antes do carregamento total, é possível que se depare com erros.

No exemplo a seguir, depois de carregada a página, invoca-se a função **alert()** para imprimir uma mensagem assim que a página estiver carregada:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Funcionamento do jQuery</title>
  </head>
  <body>

    <script src="jquery.js"></script>
    <script>
      $(document).ready(function () {
        alert('jQuery funcionando corretamente!')
      })
    </script>
  </body>
</html>
```

Se uma janela de alerta apareceu no seu navegador, então você executou seu exemplo jQuery. Caso não apareça nenhuma janela, algo está errado com o arquivo **jquery.js**.

Em vários exemplos presentes neste conteúdo será usado um recurso do JavaScript chamado **função anônima**. Observe quando se usa a função **ready()**. O que foi passado como parâmetro? Veja, no exemplo a seguir, o trecho destacado:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Função anônima</title>
  </head>
  <body>
    <script src="jquery.js"></script>
    <script>
      $(document).ready(function () {
        alert('Esta é uma função anônima!!')
      })
    </script>
  </body>
</html>
```

A função **ready()** está recebendo uma função como parâmetro. O escopo dessa função existe apenas dentro da chamada da função **ready()**. Essa função não tem um nome ou identificador. Seu propósito é existir apenas quando a função **ready()** for chamada. Esse tipo de função é chamado de função anônima. O uso de função anônima é bem comum no jQuery, mas não necessariamente obrigatório. O mesmo código acima poderia ser escrito como:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Função anônima</title>
  </head>
  <body>
    <script src="jquery.js"></script>
    <script>
      $(document).ready(teste)
```

```
      function teste() {
        alert('Esta é uma função normal do JavaScript!')
      }
    </script>
  </body>
</html>
```

Seletores

Antes de realizar alguma operação no jQuery, é preciso selecionar quais são os elementos ou as *tags* alvos, aos quais serão aplicadas as funções do jQuery. Para selecionar essas *tags* ou esses elementos, usam-se seletores que têm uma sintaxe semelhante ao CSS. Passa-se como argumento à função **$()** o nome da *tag*, da classe ou do **id** que se deseja selecionar.

Veja alguns exemplos:

1. Como selecionar **todas as *tags* <p>** de uma página? Basta passar como parâmetro para a função **$()** o nome da *tag*, sem os sinais **< >**:

   ```
   $("p")
   ```

2. Como selecionar todas as ***tags* cuja classe** é **"destaque"**? Passa-se como parâmetro o nome da classe CSS, lembrando que toda classe começa com um ponto (.).

   ```
   $(".destaque")
   ```

3. Como selecionar a ***tag* cujo id** é **"menu"**? Passando o nome do **id** da *tag* como parâmetro. Todo **id** inicia com o sinal de *hash* ou cerquilha (#). Pela especificação do HTML, o **id** não pode se repetir; espera-se que a chamada retorne apenas um elemento. É muito importante que essa regra seja respeitada para o funcionamento correto do jQuery e do CSS.

   ```
   $("#menu")
   ```

4. Como selecionar apenas os ***links* que estão dentro do menu cujo id** é **"menu"**? Usando o mesmo conceito do CSS, definindo primeiro o elemento **menu**, seguido de um espaço e do nome da *tag*. A leitura que se faz do seletor a seguir é: retorna todas as *tags* **<>** contidas no elemento cujo **id** é **"menu"**.

```
$("#menu a")
```

O exemplo anterior pode ser ampliado para selecionar qualquer elemento na árvore DOM. Por exemplo, suponha que você precisa selecionar apenas as **spans** que estão contidas em um parágrafo localizado dentro de uma **div** que está localizada na *tag* **header**:

```
$("header div p span")
```

Exemplo: usando seletor de classe para esconder elementos assim que a página é carregada.

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Seletores</title>
  </head>
  <body>
    <h2 class="testeClasse">Este é um título</h2>
    <p class="testeClasse">Este é um parágrafo</p>
    <p id="">Este é outro parágrafo</p>

    <script src="jquery.js"></script>
    <script>
      $(document).ready(function () {
          $('.testeClasse').hide();
      });
    </script>
  </body>
</html>
```

Note que, no código dentro do *tag* **<script>**, estão sendo selecionando os elementos **<h2>** e **<p>** que têm classe **testeClasse**. A ação **hide()** será aplicada a cada um desses elementos. Ao abrir a página, o resultado deve ser apenas a frase “Este é outro parágrafo” na tela.

Exemplo: usando seletor de **id** para esconder elementos específicos ao clicar em um botão.

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Seletores</title>
  </head>
  <body>
    <h2>Este é um título</h2>
    <p>Este é um parágrafo</p>
    <p id="testeId">Este é outro parágrafo</p>

    <button>Clique aqui para ocultar segundo parágrafo</button>
    <p>Selecionamos o segundo parágrafo (tag p) através do seletor, usando o id.</
p>

    <script src="jquery.js"></script>
    <script>
    $(document).ready(function(){
      $("button").click(function(){
          $("#testeId").hide();
      });
    });
    </script>
  </body>
</html>
```

Note, pelo código entre os *tags* **<script>**, alguns pontos:

1. O código será executado apenas após o carregamento da página, já que se está usando uma função anônima para **ready()**.
2. O código está definindo um evento para o botão presente na tela. Mais adiante, serão abordados os eventos, mas agora observe que se está selecionando o elemento por meio do nome de *tag* **($("button"))**.
3. Na função anônima passada ao **click()**, define-se o que será executado quando o clique no botão acontecer. Nesse trecho, seleciona-se o parágrafo com **id “testeId” ($("#testeId"))** e aplica-se a ação de escondê-lo **(hide())**.

Exemplo: usando seletor de *tag* para esconder todos os parágrafos da página.

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
```

```
          <meta name="viewport" content="width=device-width, initial-scale=1.0" />
          <title>Seletores</title>
        </head>
        <body>
          <h2>Este é um título</h2>
          <p>Este é um parágrafo</p>
          <p>Este é outro parágrafo</p>

          <button>Clique aqui os parágrafos</button>
          <span>(note que até a observação abaixo irá ser ocultada, devido a estar em um
a tag p)</span>
          <p>Selecionamos os dois parágrafos (tag P) e ocultamos diretamente pela tag.</
p>

          <script src="jquery.js"></script>
          <script>
          $(document).ready(function(){
            $("button").click(function(){
                $("p").hide();
            });
          });
          </script>
        </body>
      </html>
```

O código é semelhante ao exemplo anterior, com a diferença de que a chamada **$("p")** é usada para retornar todos as *tags* de parágrafo.

Manipulando estilos

Função CSS()

A função **css()** permite definir uma ou mais propriedades CSS para um elemento. No exemplo a seguir, define-se a propriedade **background-color** com valor **#CCC** para todos os parágrafos (**p**).

```
      $(document).ready(function(){
        $("p").css("background-color", "#CCC");
      })
```

Além de definir uma propriedade CSS, é possível obter o valor da propriedade CSS que já foi aplicada ao elemento. No exemplo a seguir, quando se usa a função **css()** passando apenas um parâmetro, ela funcionará como um método **get()** e retornará o valor daquela propriedade, neste caso, o tamanho da fonte:

```
$(document).ready(function(){
  var fontSize = $("p").css("font-size");
  alert(fontSize);
});
```

Também é possível alterar mais de uma propriedade CSS em uma só chamada da função **css()**. Basta passar como parâmetro para a função um conjunto de propriedades-valor no formato:

```
{"propriedade":"valor", "propriedade":"valor", ...}
```

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <title></title>
    <style>
      fundo-e-borda {
        background-color: red;
        border: 5px solid yellow;
      }
    </style>
  </head>
  <body>
    <p class="fundo-e-borda">
      Este texto terá um background da cor "Aqua" e borda de 2px, sólida e
      vermelha.
    </p>
    <p>
      Foi definido um estilo para a classe "fundo-e-borda", porém, a mesma foi alt
erada usando a função .CSS
    </p>

    <script src="jquery.js"></script>
    <script>
      $(document).ready(function () {
        $('.fundo-e-borda').css({ 'background-color': 'aqua', border: '2px solid r
ed' })
      })
    </script>
  </body>
</html>
```

Função addClass()

A função **addClass** permite adicionar uma classe CSS a um determinado elemento. É importante destacar que essa classe deve existir no arquivo CSS ou na seção **style**, caso contrário a função não terá nenhum efeito. No exemplo a seguir, todos os parágrafos da página terão associação à classe **"destaque"**:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <title>Função addClass()</title>
  </head>
  <body>
    <!--CSS-->
    <style>
      .destaque {
        color: blue;
        background-color: #f6d8ce;
      }
    </style>

    <p>
      Esse trecho terá uma cor clara de fundo e cor da letra azul, devido a função
addClass() adicionar a tag "p" para a classe destaque.
    </p>
    <!--Arquivo com os códigos padrão do jQuery-->
    <script src="jquery.js"></script>
    <!--Função addClass-->
    <script>
      $(document).ready(function(){
        $('p').addClass('destaque');
      })
    </script>
  </body>
</html>
```

Observe que a função **addClass** recebe o nome da classe, sem o sinal de ponto na frente.

Função hasClass()

Essa função permite verificar se determinado elemento tem uma classe CSS:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Função hasClass()</title>
    <style>
      .destaque {
        color: red;
        background-color: #f6d8ce;
      }
    </style>
  </head>
  <body>
    <p class="destaque">Tag com a classe "destaque"</p>
    <script src="jquery.js"></script>
    <script>
      $(document).ready(function () {
        var possuiDestaque = $('p').hasClass('destaque');
        if (possuiDestaque) alert('parágrafo possui destaque');
        else alert('classe destaque não detectada');
      })
    </script>
  </body>
</html>
```

Resultado:

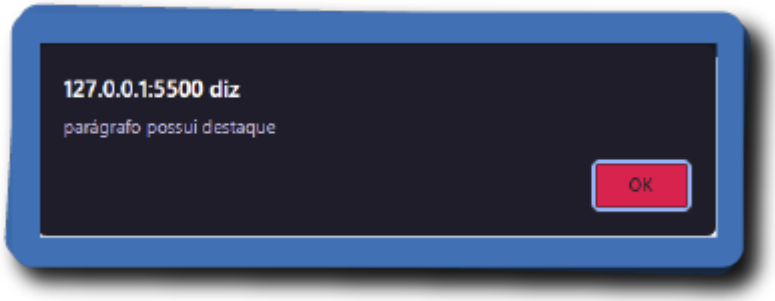

Figura 4 – Mensagem do resultado esperado do exemplo

Fonte: Senac EAD (2023)

Aparecerá a mensagem “parágrafo possui destaque” em razão da classe **destaque** e da utilização dela na *tag* **<p>**.

Função removeClass()

A função **removeClass()** faz exatamente o oposto da função **addClass()**, ou seja, remove uma classe de determinado elemento:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Função removeClass()</title>
    <style>
      .destaque {
        color: red;
        background-color: #f6d8ce;
      }
    </style>
  </head>
  <body>
    <p class="destaque">Tag com a classe "destaque"</p>
    <script src="jquery.js"></script>
    <script>
      $(document).ready(function () {
        alert("A seguir vamos remover a classe 'destaque'.");
        $('p').removeClass('destaque');
      })
    </script>
  </body>
</html>
```

No exemplo, remove-se a classe **destaque** do elemento **<p>** após o usuário dispensar a caixa de alerta que aparece ao carregar a página. Se a classe não estiver associada a **<p>**, então o código acima não terá nenhum efeito.

Figura 5 – Resultado do exemplo antes de executar **removeClass()**

Fonte: Senac EAD (2023)

Figura 6 – Resultado após executar **removeClass()**

Fonte: Senac EAD (2023)

Função toggleClass()

A operação de adicionar e remover classes CSS é algo bem comum no jQuery. Um exemplo típico é quando o usuário clica em um elemento e necessita dar um destaque a ele, como mudar a cor de fundo desse elemento. Nesse caso, seria possível adicionar uma classe CSS que realiza a formatação de destaque, e, quando o usuário clica novamente no elemento, teria de remover essa classe CSS. Porém isso pode ser redundante e trabalhoso, pois seria preciso fazer um **if** para testar se a classe existe ou não, para decidir antes se usará o método **addClass()** ou o **removeClass()**. A função **toggleClass()** faz esse controle sem a necessidade de **if**, em uma só linha. Por exemplo:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <title>Função toggleClass()</title>
    <style>
      .destaque {
        color: aquamarine;
        background-color: #561702;
      }
    </style>
```

```
  </head>
  <body>
    <p id="manchete">
      Clique na linha para adicionar ou remover a sua classe!
    </p>

    <script src="jquery.js"></script>
    <script>
      $(document).ready(function () {
        $('#manchete').click(function () {
          $(this).toggleClass('destaque');
        })
      })
    </script>
  </body>
</html>
```

Referência this

No exemplo do **toggleClass**, tem-se a palavra **this** e agora você entenderá o funcionamento dela. Considere outro exemplo que associa uma função que monitora eventos do tipo **click** em todos os parágrafos da página:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <title>Referência this</title>
    <style>
      .destaque {
        color: aquamarine;
        background-color: #561702;
      }
    </style>
  </head>
  <body>
    <p id="manchete">
      Clique na linha para adicionar ou remover a sua classe!
    </p>

    <script src="jquery.js"></script>
    <script>
      $(document).ready(function () {
        $('p').click(function () {
          $(this).toggleClass('destaque');
        })
```

```
      })
    </script>
  </body>
</html>
```

Nesse exemplo, toda vez que houver um clique em um parágrafo, a classe **destaque** será adicionada ao parágrafo, caso ainda não exista, ou removida, caso já esteja presente. Mas e o **this** nessa história toda? Observe que, no momento em que se escreve o código jQuery, não há como saber quantos parágrafos existem na página. É provável que exista mais de um parágrafo, ou até dezenas ou centenas. Portanto, se o código for rodado com o *script* da maneira como se mostra a seguir, estaria sendo feito um método que, quando recebesse um clique em **p**, aplicaria a classe **destaque** em todos os parágrafos!

```
<script>
  $(document).ready(function () {
    $('p').click(function () {
      $('p').toggleClass('destaque');
    })
  })
</script>
```

Contudo, não é isso que se quer. O objetivo é aplicar apenas no **p** que recebeu o clique. E é exatamente isso que o comando **this** faz: definir que o **<p>** que terá o **toggleClass** aplicado é apenas aquele que recebeu o clique.

A referência **this** deixa o código mais genérico, pois permite trabalhar com uma grande quantidade de elementos, sem que se saiba, a princípio, determinar qual é o elemento.

Monte uma página HTML com elementos de texto, duas classes css, uma chamada **“tema-claro”**, definindo fundo branco e letras pretas, e outra chamada **“tema-escuro”**, definindo fundo preto e letras brancas. Inclua na página dois botões, um para acionar via jQuery o tema branco e outro para o tema escuro. A classe deve ser aplicada a **<body>**.

Função **hide()**	Essa função esconde/oculta um elemento. No exemplo ao lado, quando o usuário clicar no parágrafo, este é ocultado. O usuário não enxergará mais esse parágrafo, entretanto ele continua presente na página e poderá tornar-se visível usando a função **show()**.	`$('p').click(function(){ $(this).hide(); });`
Função **show()**	Essa função faz o oposto de **hide()**. Se um elemento está escondido, é possível torná-lo visível chamando a função **show()**.	`$('#botao').click(function(){ $("p").show(); });`
Função **attr()**	Essa função permite alterar um atributo de um elemento. No exemplo ao lado, usou-se a função **attr** para alterar a largura da imagem toda vez que o *mouse* passa sobre ela. Quando sai de cima da imagem, altera-se a largura para o tamanho original.	`$(document).ready(function () { $('#photo').mouseover(function () { $(this).attr('width', 650) }) $('#photo').mouseout(function () { $(this).attr('width', 620) }) })`

Tabela 1 – Funções

Fonte: Senac EAD (2023)

Animações e efeitos especiais

O jQuery tem diversas funções que permitem realizar animações básicas com elementos HTML. Essas animações operam sobre propriedades do CSS variando seus valores de modo a criar um efeito de transição entre dois estados, por exemplo, entre um estado visível e outro oculto.

fadeIn() e **fadeOut()**

Essas funções permitem efetuar a animação entre dois estados: 1 – totalmente visível; e 2 – totalmente invisível. Ambas as funções operam sobre a propriedade **opacity** do CSS. A diferença é que a função **fadeIn()** ajusta o nível de opacidade de 0 para 1, de modo gradual, o que simula o efeito de aparecimento. Já a função **fadeOut()** faz exatamente o oposto: ela ajusta a opacidade de 1 para 0, de modo gradual, simulando o efeito de desaparecimento.

```
<!DOCTYPE html>
<html lang="pt">
  <head>
    <meta charset="UTF-8" />
    <script src="jquery.js"></script>
    <script>
      $(document).ready(function () {
        $('#desaparecer').click(function () {
          $('#foto').fadeOut()
        })
        $('#aparecer').click(function () {
          $('#foto').fadeIn()
        })
      })
    </script>
  </head>
  <body>
    <button id="desaparecer">Desaparecer</button>
    <button id="aparecer">Aparecer</button>
    <br />
    <img id="foto" src="carro01.jpg" />
  </body>
</html>
```

slideUp() e **slideDown()**

A função **slideUp()** esconde um elemento diminuindo o seu tamanho até desaparecer. A função **slideDown()** faz o contrário, torna o elemento visível aumentando seu tamanho progressivamente até atingir o tamanho original. No exemplo, a imagem **#foto** é submetida à função **slideUp()** ou **slideDown()** sempre que um dos botões é pressionado:

```
<!DOCTYPE html>
<html lang="pt">
  <head>
    <meta charset="UTF-8" />
    <script src="jquery.js"></script>
    <style>
```

```
      #foto {
        border: 1px solid black;
      }
    </style>
    <script>
      $(document).ready(function () {
        // reduz em 50% a opacidade:
        $('#slideup').click(function () {
          $('#foto').slideUp('slow')
        })
        $('#slidedown').click(function () {
          $('#foto').slideDown('slow')
        })
      })
    </script>
  </head>
  <body>
    <button id="slideup">SlideUp</button>
    <button id="slidedown">SlideDown</button>
    <br />
    <img id="foto" src="carro02.jpg" />
  </body>
</html>
```

Função de busca: each()

A função **each** permite navegar por cada um dos elementos do DOM, verificando ou alterando suas propriedades de modo individual, se assim necessário.

Considere o código a seguir, que permite ao usuário digitar um nome, e, à medida que as teclas são digitadas, a função **each** é chamada para percorrer cada **<li>** verificando se o conteúdo dessa **<li>** tem o texto informado pelo usuário:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Função Each</title>
    <style>
      .off {
        background-color: #ccc;
      }
      .on {
        background-color: red;
      }
      li {
        border-radius: 4px;
```

```
        padding: 8px;
        background-color: navy;
        color: white;
        display: inline;
      }
      ul {
        list-style: none;
      }
      input.busca {
        margin-left: 40px;
        width: 350px;
        height: 25px;
        font-size: 16px;
        padding: 4px;
      }
    </style>
    <script src="jquery.js"></script>
    <script>
      $(document).ready(function () {
        $('#filtro').keyup(function () {
          var termo = $('#filtro').val().toLowerCase()
          $('ul.nomes li').each(function (index) {
            $(this).addClass('off')
            $(this).removeClass('on')
            var li_texto = $(this).text().toLowerCase()
            if (li_texto.search(termo) != -1 && termo.length > 1) {
              $(this).removeClass('off')
              $(this).addClass('on')
            }
          })
        })
      })
    </script>
  </head>
  <body>
    <input type="text" id="filtro" class="busca" />
    <ul class="nomes">
      <li>Adriana</li>
      <li>Alice</li>
      <li>Daltron</li>
      <li>Fabiano</li>
      <li>Fernando</li>
      <li>Fabrício</li>
      <li>Lucas</li>
      <li>Gustavo</li>
      <li>Mauro</li>
      <li>Ronald</li>
    </ul>
  </body>
</html>
```

Modificando o conteúdo

As funções **text()**, **html()** e **val()** permitem obter e alterar o conteúdo de um elemento. A seguir, você verá a diferença entre cada um.

Partindo do exemplo anterior, inclua o seguinte parágrafo (**<p>**) após a *tag* **<input>**.

```
<p> Olá <span id=”spnNome”> [seu nome] <span>, tudo bem?</p>
```

Ao clicar no botão, substitua o conteúdo do **span** com **id "spnNome"** pelo conteúdo digitado pelo usuário.

Tratamento de eventos

Existem diversos tipos de eventos que ocorrem em uma página *web* durante a interação do usuário. Por exemplo, quando o usuário move o *mouse*, clica sobre um elemento, seleciona um texto, digita um texto, seleciona uma opção etc. O jQuery oferece várias funções para lidar com esses eventos, possibilitando capturá-los para disparar rotinas de tratamento de eventos associados a uma ação desejada.

Por exemplo, toda vez que o usuário passar o *mouse* sobre uma imagem, esta poderia ser aumentada para facilitar a visualização de seus detalhes. Para tornar isso possível, seriam programadas algumas algumas etapas: 1) associar o evento do *“mouse* sobre” a aquela imagem específica; 2) monitorar quando esse evento é disparado; 3) tomar uma ação para realizar o resultado desejado. Todas essas etapas são realizadas com o tratamento de eventos que o jQuery oferece em sua API.

Evento click

Talvez um dos eventos mais comuns. Ocorre quando o usuário efetua um clique do *mouse* sobre algum elemento. Veja um exemplo de como associar um evento **click** a um botão HTML:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
  </head>
```

```
  <body>
    <button id="botao">Clique aqui!</button>

    <script src="jquery.js"></script>
    <script>
    $(document).ready(function(){
        $('#botao').click(function () {
          alert('O botão foi clicado!')
        })
      })
    </script>
  </body>
</html>
```

Evento mouseover

Ocorre quando o usuário passa o cursor do *mouse* sobre determinado elemento. Veja o exemplo a seguir, que modifica a cor de fundo de uma **div** com **id #texto** toda vez que o *mouse* passa sobre ela:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
  </head>
  <body>
    <div id="texto">
      <p>Ao passar o cursor sobre esse texto, a cor de fundo ficará vermelha!</p>
    </div>

    <script src="jquery.js"></script>
    <script>
      $(document).ready(function(){
        $('#texto').mouseover(function () {
          $(this).css('background-color', 'red')
        })
      })
    </script>
  </body>
</html>
```

Observe o uso do **this**. É comum o seu uso quando se trabalha com eventos, pois geralmente quando se captura um evento sobre um determinado elemento, o próprio elemento é o alvo da ação que se realizará logo a seguir. Portanto, nesse caso, o **this** se refere ao elemento **#texto**, que é a **div**.

Avalie com atenção quando é mais vantajoso usar um evento **mouseover()** ou quando usar um seletor **:hover** de CSS. O evento é útil quando há mais ações a serem realizadas no evento, como a ativação de um elemento HTML, por exemplo, enquanto o **:hover** é preferível se a questão for apenas uma alteração de estilo. Eventos em CSS são bem menos custosos computacionalmente do que código JavaScript.

Evento change

Esse evento pode ser bastante útil para capturar quando o estado de um elemento é alterado. Considere que é preciso alterar o atributo **src** de uma imagem de acordo com a opção selecionada em um elemento **select**. Usando o evento **change**, fica fácil fazer isso:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <title>Evento Change</title>
  </head>
  <body>
    Selecione o modelo de carro:
    <select id="modelos">
      <option></option>

      <option value="carro01.jpg">Carro 01</option>
      <option value="carro02.jpg">Carro 02</option>
      <option value="carro03.jpg">Carro 03</option>
    </select>
    <img src="" id="foto" />

    <script src="jquery.js"></script>
    <script>
      $(document).ready(function () {
        $('#foto').hide();
        $('#modelos').change(function () {
          $('#foto').attr('src', $(this).val());
          $('#foto').show();
        });
      })
    </script>
```

```
  </body>
</html>
```

Nesse exemplo, a função **change** só é disparada quando o **select** tem seu valor modificado. Assim, basta usar a função **attr()** para modificar o atributo **src** da imagem.

Evento keypress

O evento **keypress** pertence à outra categoria de eventos associados a ações que o usuário realiza por meio do uso do teclado. A função **keypress** permite monitorar quando uma tecla é pressionada e, assim, realizar algum tipo de ação. Considere o código a seguir:

```
<!DOCTYPE html>
<html lang="pt">
  <head>
    <meta charset="UTF-8" />
    <title>Evento Keypress</title>
  </head>
  <body>
    Digite seu nome:
    <input type="text" id="nome" />
    <div id="exibe"></div>

    <script src="jquery.js"></script>
    <script>
      $(document).ready(function () {
        $('#nome').keypress(function (evento) {
          $('#exibe').append(
            'Código ascii da tecla digitada: ' + evento.keyCode + '<br>',
          )
        })
      })
    </script>
  </body>
</html>
```

Nesse exemplo, é associado um tratador de eventos do tipo **"tecla pressionada"** ao input **#nome**. Assim, toda vez que a tecla é pressionada, esse tratador é chamado. Nesse caso, a função de tratamento recebe como parâmetro um objeto chamado **"evento"**, que

tem informações sobre o evento da tecla pressionada. Com isso, é possível saber exatamente qual foi a tecla pressionada por meio do atributo **keyCode**.

Lista de eventos

A tabela a seguir apresenta uma lista resumida de eventos separados por tipos de eventos: (1) eventos de *mouse*; (2) eventos de teclado e (3) eventos do *browser*. Todos esses eventos podem ser mapeados no jQuery e os devidos tratamentos podem ser realizados.

Função	Descrição
Eventos de *mouse*	
.change()	Ocorre toda vez que o valor do elemento é alterado.
.click()	Ocorre quando um *click* do *mouse* é detectado em um elemento.
.dbclick()	É um evento de clique duplo sobre um elemento.
.focus()	Ocorre quando um elemento recebe o foco.
.blur()	Ocorre quando um elemento perde o foco.
.hover()	Ocorre quando o *mouse* passa sobre um elemento.
.mousedown()	É disparado quando o botão do *mouse* é apertado.
.mousemove()	É acionado toda vez que o *mouse* se movimento dentro de um elemento.
.mouseout()	É acionado toda vez que o *mouse* sai de cima de um elemento.
.mouseover()	É acionado toda vez que o *mouse* passa sobre o elemento.
.mouseup()	É acionado toda vez que o botão do *mouse* é clicado e liberado.
Eventos de teclado	
.keydown()	É acionado quando uma tecla é apertada.
.keyup()	É acionado quando uma tecla é liberada.
.keypress()	É acionado quando uma tecla é pressionada.
Eventos do browser	
.ready()	É acionado indicando que todo o código HTML foi lido e a árvore DOM está pronta.

.resize()	É acionado quando o tamanho da janela do *browser* é alterado.

Tabela 2 – Lista resumida de eventos
Fonte: Senac EAD (2023)

Exemplo: página para definição de senha com um campo para a nova senha e outro para a confirmação da nova senha. Quando se altera o valor do primeiro campo, a página mostra uma dica informando que ela deve ter seis caracteres caso o usuário tenha digitado menos que isso. Ao clicar no botão, valida se a senha tem tamanho mínimo e se está idêntica nos dois campos.

```
<!DOCTYPE html>
<html lang="pt">
  <head>
    <meta charset="UTF-8" />
    <title>Evento Keypress</title>
  </head>
  <body>
    <div>
        Digite a senha: <input type="password" id="txtSenha" />
        <span id="spnDica" style="font-size: smaller; color: red;">A senha deve te
r ao menos 6 caracteres</span>
    </div>
    <div>
        Repita a senha: <input type="password" id="txtSenhaRepete" />
    </div>
    <div>
        <button id="btnOk">Definir Senha</button>
        <span id="spnResultado"></span>
    </div>

    <script src="jquery.js"></script>
    <script>
      $(document).ready(function () {
        //ao iniciar a página, esconte a dica
        $("#spnDica").hide();

        //evento dispara ao alterar valor e sair do campo
        $("#txtSenha").change(function(){
            let senha = $("#txtSenha").val();
            if(senha.length < 6){
                $("#spnDica").show();
            }else{
                $("#spnDica").hide();
            }
        });

        //evento dispara ao clique do botão
        $("#btnOk").click(function(){
            let senha = $("#txtSenha").val(),
```

```
                senha2 = $("#txtSenhaRepete").val();

            if(senha.length >= 6 && senha == senha2){
                //ações encadeadas: primeiro altera conteúdo depois altera cor
                $("#spnResultado").text("Senha Válida!").css("color", "green");
            }else{
                $("#spnResultado").text("Senha Inválida!").css("color", "red");
            }
        });

    })
  </script>
 </body>
</html>
```

Monte uma página que contenha um campo de texto (**<input>**) e implemente eventos para:

- Tornar o texto digitado no campo sempre maiúsculo.
- Quando o usuário digitar **Enter** (código 13), mostrar uma caixa *alert* com o conteúdo do campo de texto e apagar o texto presente nesse campo.

Manipulação dinâmica do DOM

Veja, a seguir, funções para a alteração dinâmica do DOM, seja removendo, seja adicionando ou alterando a hierarquia de elementos da árvore DOM.

insertBefore e **insertAfter**

A função **insertBefore** permite anexar um elemento no DOM, exatamente antes de um determinado elemento. A função **insertAfter** faz o oposto, ela permite anexar um elemento logo depois de um determinado elemento. No exemplo a seguir, existem dois botões. O primeiro insere um elemento **p**, criado dinamicamente antes do elemento **#menu**. O segundo botão insere um elemento **p** depois do elemento **#menu**:

```
<!DOCTYPE html>
<html lang="pt">
  <head>
    <meta charset="UTF-8" />
    <style>
```

```
      div,
      p {
        border: 1px solid black;
        margin: 10px;
      }
      #menu {
        background-color: lightgreen;
      }
    </style>
    <script src="jquery.js"></script>
    <script>
      $(document).ready(function () {
        $('#insertB').click(function () {
          $('<p>elemento inserido antes</p>').insertBefore('#menu')
        })
        $('#insertA').click(function () {
          $('<p>elemento inserido depois</p>').insertAfter('#menu')
        })
      })
    </script>
  </head>
  <body>
    <button id="insertB">InsertBefore P (Inserir antes)</button>
    <button id="insertA">InsertAfter P(Inserir depois)</button>
    <h1>Título principal</h1>
    <div id="menu">Porções: arroz, salada e carne.</div>
    <div id="artigos">Conheça mais sobre o restaurante</div>
  </body>
</html>
```

parent()

Essa função retorna o elemento pai de um dado elemento.

```
<!DOCTYPE html>
<html lang="pt">
  <head>
    <meta charset="UTF-8" />
    <script src="jquery.js"></script>
    <style>
      div.destaque {
        background-color: lightblue;
      }
    </style>
    <script>
      $(document).ready(function () {
        $('.seleciona').click(function () {
          $(this).parent().toggleClass('destaque')
        })
      })
```

```
    </script>
  </head>
  <body>
    <h4>Escolha seu destino</h4>

    <div class="destino">
      <input class="seleciona" type="checkbox" />
      <span>Porto Alegre</span>
    </div>
    <div class="destino">
      <input class="seleciona" type="checkbox" />
      <span>Florianópolis</span>
    </div>
    <div class="destino">
      <input class="seleciona" type="checkbox" />
      <span>Curitiba</span>
    </div>
  </body>
</html>
```

Toda vez que um *click* ocorrer na **input "seleciona"**, a função **parent()** busca o ancestral direto da **input**, que, no exemplo, é a div **"destino"**.

children()

Essa função retorna todos os nodos filhos de um elemento.

```
<!DOCTYPE html>
<html lang="pt">
  <head>
    <meta charset="UTF-8" />
    <script src="jquery.js"></script>
    <style>
      span.destaque {
        color: red;
      }
    </style>
    <script>
      $(document).ready(function () {
        $('.destino').bind('mouseover mouseout', function () {
          $(this).children().toggleClass('destaque')
        })
      })
    </script>
  </head>
  <body>
    <h4>Escolha seu destino</h4>
    <div class="destino">
      <span>São Paulo</span>
```

```
            <span><b>|+| Mais informações</b></span>
          </div>
          <div class="destino">
            <span>Rio de Janeiro</span>
            <span><b>|+| Mais informações</b></span>
          </div>
        </body>
      </html>
```

Nesse exemplo, foram associados dois eventos, **mouseover** e **mouseout**, às **divs** com classe **".destino"**. Quando o evento é disparado, todos os nodos filhos, no caso ****, **** e **<b>**, são formatados com a classe **"destaque"**. Essa função permite, em uma só linha de comando, atingir todos os nodos filhos de uma só vez, o que pode ser bastante útil quando não se sabe exatamente quantos nodos filhos existem.

AJAX

AJAX (acrônimo em inglês de *asynchronous JavaScript and XML*, em português JavaScript e XML assíncronos) é o uso metodológico de tecnologias como JavaScript e XML (*extensible markup language*), providas por navegadores, para tornar páginas *web* mais interativas com o usuário, utilizando-se de solicitações assíncronas de informações.

XML foi inicialmente desenvolvida pelo estudioso Jessé James Garret e em seguida por diversas associações. Apesar do nome, a sua utilização não é obrigatória (JSON, ou *JavaScript object notation*, é frequentemente utilizado), e as solicitações também não precisam, obrigatoriamente, ser assíncronas.

Quando se submete um formulário ao servidor, seja via GET, seja via POST, você acaba enviando junto muito mais informações do que gostaria (geralmente a página inteira é enviada), bem como direcionando a resposta do usuário para longe da página atual. Muitas vezes, você gostaria de consumir conteúdo do servidor sem ter de trocar de página, e, mesmo que a tecnologia de *back-end* permita redirecionar o fluxo novamente para a mesma tela, há aquelas “piscadas” inconvenientes no navegador, que fazem com que todos os dados que estavam na página se percam. Essa é a motivação por trás do AJAX: economia de dados trafegados e melhor experiência para o usuário.

A respeito da função **ajax()**, esta permite gerar requisições do tipo GET ou POST para um servidor de forma assíncrona, isto é, sem a necessidade de bloquear a interface do usuário até o retorno da resposta. Para fazer uma chamada assíncrona, é preciso definir alguns parâmetros, os quais você verá a seguir.

URL

Define o endereço da página ou o arquivo a ser requisitado.

Contexto

Define o elemento que será vinculado à requisição, geralmente aquele que receberá os dados retornados pelo servidor.

Type

Define o método de requisição que será utilizado, geralmente GET ou POST.

Success

Define a função **callback**, que é executada após a resposta do servidor.

A seguir, veja um exemplo de requisição assíncrona que solicita o arquivo **tabela.txt**, sem bloquear a navegação do usuário. Caso a resposta seja confirmada (com sucesso), uma função anônima é declarada, cujo parâmetro **data** encapsula os dados retornados pelo servidor. Nesse caso, a operação se completa fazendo um **append()** desses dados ao elemento **#lista**.

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
```

```
    <script src="jquery.js"></script>
    <style>
      ul {
        font-family: monospace;
      }
      button {
        display: block;
        margin: 20px 0px;
      }
    </style>
    <script>
      $(document).ready(function () {
        $('#carregar').click(function () {
          $.ajax({
            url: 'tabela.txt',
            context: $('#lista'),
            type: 'GET',
            success: function (data) {
              this.append(data)
            },
          })
        })
      })
    </script>
  </head>
  <body>
    <h4>Melhores destinos</h4>
    <button id="carregar">Carregar lista</button>
    <ul id="lista"></ul>
  </body>
</html>
```

Para conseguir testar essa função, é necessário que ela esteja dentro de um servidor *web* como o Apache, ou executar a página HTML por meio da extensão “Live Server”, do Visual Studio Code. Você também deverá criar um arquivo no bloco de notas com o nome **tabela.txt** e colocar na mesma pasta em que está o arquivo HTML.

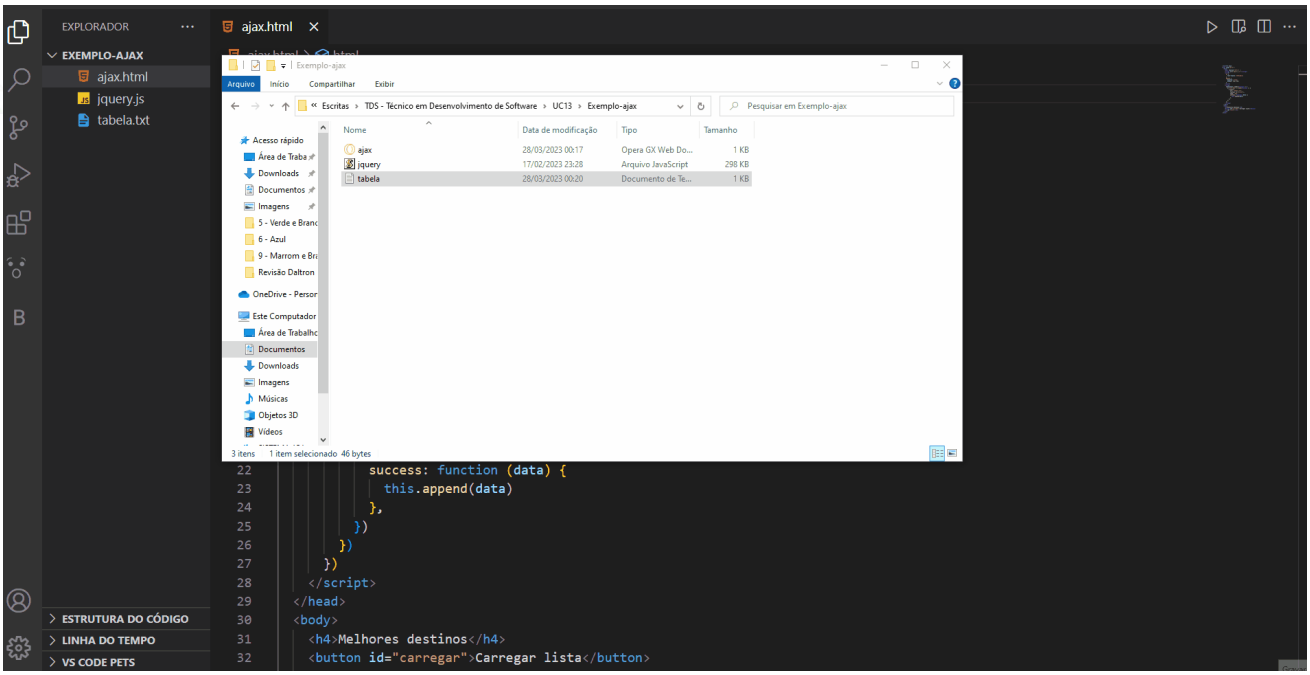

Figura 7 – Estrutura da pasta com os arquivos para testar o exemplo

Fonte: Senac EAD (2023)

Outras tecnologias

É possível citar outras bibliotecas, como React e Angular, as quais você conhecerá a seguir.

React

O React é uma biblioteca do JavaScript para o desenvolvimento de interfaces de usuário (UIs). Ele é gerenciado pela equipe do Facebook e por uma comunidade de desenvolvedores de código aberto. O *framework* veio a público em maio de 2013.

O React é uma biblioteca que auxilia na construção de interfaces. Se comparado com um modelo MVC (*Model* – *View* – *Controller*), ele é o V, responsável apenas pela *view*.

É possível destacar algumas vantagens, como: facilidade de aprendizado, reaproveitamento de código, comunidade ativa em fóruns, sintaxe amigável na construção de componentes, utilização por grandes empresas e biblioteca rápida.

Para acessar o *site* oficial, pesquise em seu buscador “React, uma biblioteca JavaScript para criar interfaces de usuário”.

Angular

O Angular é um *framework* de JavaScript em código aberto de desenvolvimento para *web* e dispositivos móveis. Ele tem como base o TypeScript e é gerenciado pela equipe do Angular do Google, assim como pela comunidade de desenvolvedores do Angular.

Lançado em setembro de 2016, o Angular (também conhecido como Angular 2.0) é uma reinterpretação completa do AngularJS (Angular 1.0), que foi lançado em 2010.

Também usa a arquitetura MVC, encaixando-se como o “M” (*model*/modelo). O HTML se encarrega da *view* (visão), e o controle (*controller*) fica por conta do JavaScript.

É possível destacar algumas vantagens, como: bastante tempo de mercado, ferramenta potente e TypeScript nativo. Para acessar o *site* oficial, digite “Angular” em seu buscador.

Encerramento

O uso de bibliotecas *front-end* é essencial em qualquer projeto, pois elas auxiliam de diversas formas, por exemplo: redução no tamanho do código e melhor organização, uso de funções predefinidas e maior agilidade na hora do desenvolvimento dos códigos.

Existe uma gama muito grande de bibliotecas disponíveis, cabendo ao programador selecionar a que melhor se encaixa em seu projeto e pesquisar como é o seu uso em sua documentação, que sempre é disponibilizada no *site* oficial.