Instituto Nacional de Telecomunicações – INATEL

Pós-graduação em Desenvolvimento de Aplicações para Dispositivos Móveis e Cloud Computing

DM121 – Introdução ao desenvolvimento Web: HTML5, CSS3 e Javascript

Sobre a apostila

Este é um material de apoio para o curso de Pós-graduação em Desenvolvimento de Aplicações para Dispositivos Móveis e Cloud Computing do Instituto Nacional de Telecomunicações – INATEL, referente a disciplina DM121 – Introdução ao desenvolvimento Web: HTML5, CSS3 e Javascript.

Sumário

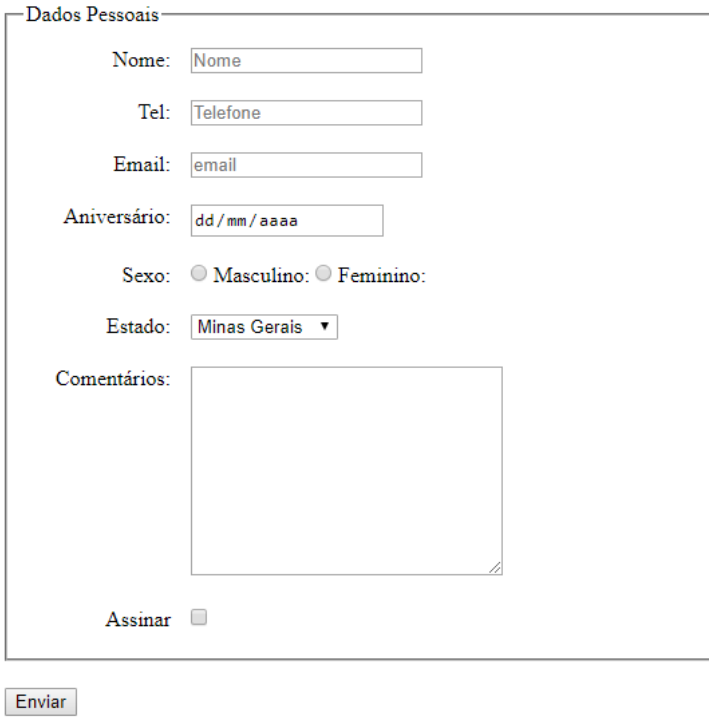
1. Formulários e validações

Formulários em HTML são utilizados para permitir que usuários enviem dados para o servidor. Considere o seguinte HTML:

```
<form action="" method="">

        <fieldset>

            <legend>Dados Pessoais</legend>

            <div class="fields">

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

                <input type="text" placeholder="Nome" name="name" id="name">

            </div>

            <div class="fields">

                <label for="tel">Tel:</label>

                <input type="tel" placeholder="Telefone" name="tel" id="tel">

            </div>

            <div class="fields">

                <label for="email">Email:</label>

                <input type="email" placeholder="email" name="email" id="email">

            </div>

            <div class="fields">

                <label for="aniversario">Aniversário:</label>

                <input type="date" name="aniversario" id="aniversario">

            </div>

            <div id="field-sexo">

                <label id="label_sexo" for="opt_sexo">Sexo:</label>

                <div id="opt_sexo">

                    <input type="radio" name="sexo" id="sexo_masc">

                    <label for="sexo_masc">Masculino:</label>

                    <input type="radio" name="sexo" id="sexo_fem">

                    <label for="sexo_fem">Feminino:</label>

                </div>

            </div>

            <div class="fields">

                <label for="lista">Estado:</label>
```

```
            <select name="lista" id="lista">
                <option value="MG">Minas Gerais</option>
                <option value="RJ">Rio de Janeiro</option>
                <option value="SP">São Paulo</option>
            </select>
        </div>
        <div class="fields">
            <label for="comentarios">Comentários:</label>
            <textarea name="comentarios" id="comentarios" cols="30" rows="10"></textarea>
        </div>
        <div class="fields">
            <label for="newsletter">Assinar</label>
            <input type="checkbox" name="newsletter" id="newsletter">
        </div>
    </fieldset>
    <button type="submit">Enviar</button>
</form>
```

Como resultado temos:



Observe o seguinte trecho de HTML

```
<form action="" method="">
```

O atributo **action** especifica para onde os dados do formulário serão enviados quando forem submetidos e o **method** é utilizado para especificar como os dados serão enviados e pode assumir o valor POST ou GET que são métodos do HTTP.

Formulários não são novidades no HTML e já são utilizados há algum tempo, porém, quando chegou o HTML 5 além das melhorias semânticas também foram adicionadas melhorias especificas para formulários como: novos atributos, tipos específicos para cada tipo de entrada de entrada de dados, validações entre outros.

Em cada **input** observe que existe um tipo especifico, em versões anteriores do HTML isso não era comum, mas com a chegada do HTML5 novos tipos foram adicionados.

color, date , datetime , datetime-local , email , month , number , range , search, tel, time , url, week.

```
<input type="email" placeholder="email" name="email" id="email">
```

```
<input type="date" name="aniversario" id="aniversario">
```

Alguns dos novos atributos exclusivos do HTML 5 são:

autocomplete, autofocos, formnovalidate, min and max, pattern (regexp), placeholder, required, step

O atribuito **required**

```
<input type="text" placeholder="Nome" name="name" id="name" required>
```

Esse atributo adiciona uma validação simples ao campo, ou seja, quando tentar submeter o formulário sem o campo preenchido os dados não serão enviados pelo fato do campo ser **required**

O atributo **autocomplete**

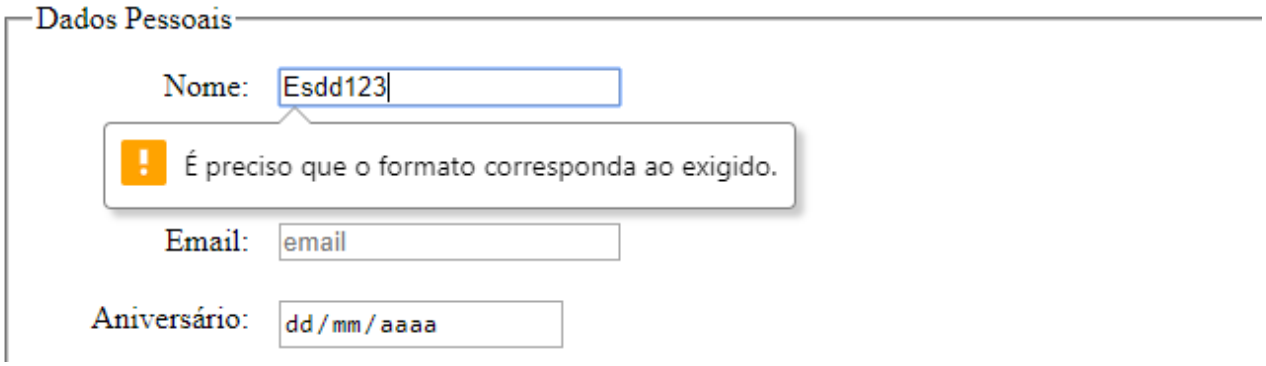
```
<input type="tel" placeholder="Telefone" name="tel" id="tel" autocomplete="on">
```

O atributo **autocomplete** permite que baseados em valores digitados anteriormente o navegador exiba opções para preencher o campo.

O atributo **pattern**

```
<input type="text" placeholder="Nome" name="name" id="name" required
pattern="[a-zA-Z\s]+">
```

O atributo **pattern** permite adicionar um padrão de entrada baseado em expressão regular, neste exemplo é esperado que somente letras sejam digitadas no campo.



Observe que se digitado algo fora do **pattern** ocorre uma validação.

O HTML oferece um conjunto básico de validações do lado cliente que pode ser utilizado para melhorar a interação com o usuário, porém, precisa ser lembrado que esse tipo de validação é do lado cliente (navegador) ou seja, ainda é preciso adicionar validação do lado servidor quando necessário.

2. Seletores avançados em CSS

Em algumas situações é necessário agrupar, combinar seletores CSS a fim de conseguir aplicar regras mais complexas e é para esses casos que entram alternativas mais avançadas em CSS para alcançar tal objetivo.

Considere o seguinte HTML:

```
<header>
        <p>Header</p>
    </header>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
        Illum cupiditate ex molestiae doloribus placeat, ipsa,
        rem dolore consequuntur nulla modi sint assumenda ]
        aliquam harum culpa atque repellendus non iste necessitatibus?
    </p>
    <p>Lorem ipsum dolor sit, amet consectetur adipisicing elit.
       Placeat perspiciatis voluptatum praesentium deserunt quia quam,
       perferendis impedit, esse eveniet iusto temporibus et possimus
       architecto qui eos vitae ratione accusamus ut.
    </p>
    <secion class="main">
        <article>
            <h1>Título</h1>
            <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
                Officia doloremque quia ipsam nobis quas mollitia rem consectetur
                minima error quaerat!Cupiditate nemo quos aliquam quae magnam
                quam perspiciatis et officiis.
            </p>
        </article>
    </secion>
    <section class="portfolio">
        <h2>Portfólio</h2>
        <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
            Odio officia repellat ut voluptate ducimus quae perspiciatis
            libero harum vel. Itaque unde similique, rem adipisci incidunt voluptas
```

```
          ipsam aut quia quo!
      </p>
   </section>
```

Agrupando seletores

ELEMENTO1, ELEMENTO2

```
h1,p{
    color: red
}
```

Neste exemplo tanto o elemento **h1** quanto o elemento **p** serão vermelhos.

Seletores descendentes

ELEMENTO1 ELEMENTO2

```
.portfolio P{
    text-transform: uppercase
}
```

Neste exemplo todo elemento **p** que descende do elemento que tenha a classe **portfolio** será maisculo.

Seletores filho

ELEMENTO 1 > ELEMENTO 2

```
article > p{
    border: 5px double green
}
```

Neste exemplo todo elemento **p** que é filho direto de um elemento **article** terá bordas duplas, verdes e com 5px.

Seletores irmãos adjacentes

ELEMENTO1 + ELEMENTO2

```
header + p{
   font-style: italic
}
```

Neste exemplo todo elemento **p** que é o irmão mais próximo do elemento **header (**onde o elemento **header** deve ter o mesmo pai do elemento **p)** terá a fonte em itálico.

Seletores irmãos

ELEMENTO1 ~ ELEMENTO2

```
header ~ p{
    color: blueviolet
}
```

Neste exemplo todo elemento **p** que é irmão do elemento **header** terá a cor azul-violeta.

Seletores de atrbuto

Considere o seguinte HTML:

```
<ul id="lista">
        <li value="Item-1" disabled class="item highlights" id="item1">Item 1</li>

        <li value="Item-2" class="item" id="item2">Item 2</li>
        <li value="Item-3" class="item highlights" id="item3">Item 3</li>

        <li value="Item-4" class="item" id="item4">Item 4</li>
        <li value="Item-5" disabled class="item" id="item5">Item 5</li>
        <li value="6" disabled class="item" id="item6">Item 6</li>
    </ul>
```

É um tipo de seletor que considera o valor do atributo de determinado elemento.

[attr] – Seleciona todos os elementos que tem o atributo.

```
[disabled]{
    color: red
}
```

Neste exemplo os elementos Item 1 e Item 5 serão vermelhos pois possuem o atributo **disabled**

[attr=valor] – Seleciona todos os elementos que tem o atributo e o valor.

```
li[value="Item-3"]{
    text-decoration: underline
}
```

Neste exemplo o elemento Item 3 será sublinhado pois é o único que possui o valor “Item 3”

[attr~=valor] – Seleciona todos os elementos que tem o atributo e se contem o valor.

```
li[class~="highlights"]{
    border: 3px dotted blue
}
```

Neste exemplo todos os elementos que possuem a classe “highlights” no valor terão bordas pontilhadas azuis de 3px.

Também é possível usar seletores de atributos baseados em padrões (semelhantes a expressões regulares mas não com todos os recursos).

[attr|=val] : Seleciona todos os elementos que tem o atributo com valor exato ou se o valor inicia com o padrão.

[attr^=val] : Seleciona todos os elementos que tem o atributo que inicia com o valor passado.

[attr$=val] : Seleciona todos os elementos que tem o atributo que termina com o valor passado.

[attr*=val] : Seleciona todos os elementos que tem o atributo que contem o valor passado.

Pseudo-classes

São classes que permitem selecionar elementos baseado em seu estado. Existe várias possibilidades de pseudo-classes, por exemplo:

:active	:hover	:only-child
:any	:indeterminate	:only-of-type
:checked	:in-range	:optional
:default	:invalid	:out-of-range
:dir()	:lang()	:read-only
:disabled	:last-child	:read-write
:empty	:last-of-type	:required
:enabled	:left	:right
:first	:link	:root
:first-child	:not()	:scope
:first-of-type	:nth-child()	:target
:fullscreen	:nth-last-child()	:valid
:focus	:nth-last-of-type()	:visited
:focus-within	:nth-of-type()	

Não será tratato todas as possibilidades, no entanto você pode encontrar mais detalhes aqui.

Considere o seguinte HTML:

```
<a href="#">Clique aqui</a>
<ul id="lista">
        <li value="Item-1" disabled class="item highlights" id="item1">Item 1</li>
        <li value="Item-2" class="item" id="item2">Item 2</li>
        <li value="Item-3" class="item highlights" id="item3">Item 3</li>
        <li value="Item-4" class="item" id="item4">Item 4</li>
        <li value="Item-5" disabled class="item" id="item5">Item 5</li>
        <li value="6" disabled class="item" id="item6">Item 6</li>
    </ul>
```

A sintaxe de pseudo-classes é: **elemento : <estado>** por exemplo:

```
li:hover{
    font-size: 2em;
    cursor: pointer
}
```

Neste exemplo todo elemento **li** que estiver no estado **hover (**evento ao passar o mouse sobre o elemento) irá aumentar o tamanho da fonte e alterar o tipo do cursor do mouse.

```
a:visited{
    color: green
}
```

Neste exemplo o link ficará na cor verde quando já tenha sido visitado.

Pseudo-elementos

São utilizados para selecionar partes do elemento. As possibilidades de pseudo-elementos são:

`::after`

`::before`

`::first-letter`

`::first-line`

`::selection`

`::backdrop`

A sintaxe de pseudo-elementos é: **elemento :: <pseudo-elemento>**

```
li[class*="item"]::after{
    content: "↑"
}
```

Neste exemplo, todo elemento **li** que possuir alguma classe **item** será adicionado o caracter ↑ após seu conteúdo.

```
li::first-letter{
    font-size: 2em
}
```

Neste exemplo, toda primeira letra do elemento **li** terá a fonte aumentada em 2em.

Exemplo 1 – Estilizando tabelas

O objetivo é adicionar estilo em uma tabela HTML, para isso é necessário utilizar alguns seletores mais avançados. O estilo da tabela deve ser dessa forma:

Nome	Email	telefone
Contato 1	c1@contato.com	359123456
Contato 1	c1@contato.com	359123456
Contato 2	c2@contato.com	359223456
Contato 3	c3@contato.com	359323456
Contato 4	c4@contato.com	359423456
Contato 5	c5@contato.com	359523456
Contato 6	c6@contato.com	359623456
Contato 7	c7@contato.com	359723456
Contato 8	c8@contato.com	359823456
Contato 9	c9@contato.com	359923456

Crie um arquivo chamado **index.html** com o seguinte conteúdo:

```
<table id="contatos">
        <tr>
          <th>Nome</th>
          <th>Email</th>
          <th>telefone</th>
        </tr>
        <tr>
          <td>Contato 1</td>
```

```
        <td>c1@contato.com</td>
        <td>359123451</td>
    </tr>
    <tr>
        <td>Contato 2</td>
        <td>c2@contato.com</td>
        <td>359123452</td>
    </tr>
    <tr>
        <td>Contato 3</td>
        <td>c3@contato.com</td>
        <td>359223453</td>
    </tr>
    <tr>
        <td>Contato 4</td>
        <td>c4@contato.com</td>
        <td>359323454</td>
    </tr>
    <tr>
        <td>Contato 5</td>
        <td>c5@contato.com</td>
        <td>359423455</td>
    </tr>
    <tr>
        <td>Contato 6</td>
        <td>c6@contato.com</td>
        <td>359523456</td>
    </tr>
    <tr>
        <td>Contato 7</td>
        <td>c7@contato.com</td>
        <td>359623457</td>
    </tr>
    <tr>
```

```
            <td>Contato 8</td>
            <td>c8@contato.com</td>
            <td>359723458</td>
        </tr>
        <tr>
            <td>Contato 9</td>
            <td>c9@contato.com</td>
            <td>359823459</td>
        </tr>
        <tr>
            <td>Contato 10</td>
            <td>c10@contato.com</td>
            <td>359923410</td>
        </tr>
    </table>
```

Crie um arquivo chamado **style.css** e implemente as seguintes regras CSS.

A tabela deve ser estilizada com base nas seguintes definições:

Tabela

- Fontes: Arial, Helvetica, sans-serif;
- Tipo da borda da tabela: colapse;
- Largura de 600px

Cabeçalho da tabela

- Cor de fundo: #1e2e01;
- Cor da fonte: #ffffff;
- Texto alinhado a esquerda;
- Preenchimento de 10px.

Estilo das células

- Bordas de 1px solid #dbebe3;
- Preenchimento de 10px.

Estilo das linhas

- As linhas pares da tabela devem ter cor de fundo: #8f757549.

  Dica: Use esta pseudo-classe https://developer.mozilla.org/en-US/docs/Web/CSS/%3Anth-child.

Efeito de mouse hover

Quando o usuário passar o mouse sobre a linha da tabela o seguinte estilo deve ser aplicado:

- Cursor do mouse dever ser “pointer”;
- A cor de fundo deve ser #887480a2;
- Cor da fonte deve ser #ffffff.

Um exemplo de solução para estilizar a tabela seria:

```
#contatos{
    font-family: Arial, Helvetica, sans-serif;
    border-collapse: collapse;
    width: 600px
}

#contatos th{
    background-color: #1e2e01;
    color: #ffffff;
    text-align: left;
    padding: 10px
}

#contatos td{
    border: 1px solid #dbebe3;
    padding: 10px
}

#contatos tr:nth-child(even){
    background-color: #8f757549
}

#contatos tr:hover{
    cursor: pointer;
    background-color: #887480a2;
    color: #ffffff
}
```

3. Introdução ao layout responsivo

A fim de propiciar uma experiência satisfatória tanto para usuários desktop quanto para usuários mobile e suas diferentes resoluções e qualidade de tela é preciso repensar a maneira que as aplicações web, web sites são desenvolvidas.
Em 2010 foi publicado um artigo cujo título era: “Responsive Web Design”, este artigo públicado por Ethan Marcotte iria mudar para sempre a maneira de desenvolver design para web. Devido a repercussão positiva desse artigo, mais tarde seria lançado um livro, também intitulado de: Responsive Web Design.

3.1. Definição

Web Design Responsivo é basicamente um conceito, onde somos responsáveis em permitir que determinada informação seja apresentada de forma clara e acessível independente de dispositivo.
O design responsivo é fundamentado em três vertentes: **Layout Fluído**, **Recursos Flexíveis** e **Media Queries**. Essas três tecnologias juntas possibilitam o desenvolvimento de uma web mais acessível e independente de dispositivo.

3.2. Layout Fluído

É a criação de layout sem utilização de medidas fixas (pixels, pontos, centímetros, etc), ou seja, é utilizar medidas flexíveis. Para desenvolvimento de um layout fluído são utilizadas as seguintes medidas: **PORCENTAGEM**, **EM**, **REM**.
PORCENTAGEM: Especifica uma medida em relação ao elemento “pai”. Por exemplo:

```
body{
   width: 100% /*Ocupará a largura da página toda*/
}
section{
  width: 50% /*Ocupa 50% da página em relação ao elemento pai*/
}
```

EM: Especifica uma medida em relação ao font-size do elemento pai. Por padrão, caso não exista nenhuma regra CSS o font-size adotado pelos navegadores é de 16px.

```
p{
  font-size: 2em /*Parágrafos serão 16px * 2 = 32px */
}
```

REM: Pode ser utilizado em qualquer propriedade, porém, temos que ter em mente que o mesmo faz relação com o tamanho da fonte do elemento pai. Na maioria dos casos será utilizado o **EM**.

Ao utilizar medidas flexíveis nota-se o efeito colateral (no bom sentido) nos elementos filhos, ou seja, os elementos filhos são alterados de forma proporcional ao elemento pai e de maneira flexível por exemplo:

HTML

```
<body>
    <section class="main-content">
            <p>HTML</p>
            <p>CSS</p>
            <p>Javascript</p>
    </section>
</body>
```

CSS

```
section {font-size: 1.125em} p {font-size: 0.7em}
```

A tag html e body ainda terão 16px de font-size como referência (valor default do navegador).

A tag section terá: 16px * 1.125 = 18px

A tag p terá: 18px * 0.7 = 12.6px

Neste caso a tag p irá assumir o valor base do elemento pai, que nesse caso é a tag section.

rem: Específica uma medida relativa e proporcional ao elemento root (<html>). Isto implica que o tamanho será relativo ao tamanho da fonte base do documento (Caso não seja definida uma fonte, irá assumir os 16px que é padrão entre os navegadores).

Por exemplo:

HTML

```
<body>
    <section class="main-content">
            <p>HTML</p>
```

```
        <p>CSS</p>
        <p>Javascript</p>
    </section>
</body>
```

CSS

```
section {font-size: 1.125rem}
p {font-size: 0.7rem}
```

A tag html e body ainda terão 16px como referência (valor default do navegador).

A tag section terá: 16px * 1.125 = 18px

A tag p terá: 16px * 0.7 = 11.2px

Até a escrita desse material a única restrição com relação a medida **rem** é o uso no Internet Explorer 8. Ainda existem outras medidas flexíveis que não estão sendo abordadas nesse documento.

3.2.1. Aplicando medidas flexíveis

A principal ação para a concepção de um layout fluído é: Não utilizar medidas fixas.

Conhecendo a tag Viewport

Meta tag viewport <meta name="viewport" content="">, em nosso contexto, viewport é a parte visível da página renderizada pelos navegadores. A maioria dos navegadores móveis tentam renderizar a página web ajustando o zoom do display automaticamente e isso é um problema quando a página já está ajustada para telas pequenas, ou seja, pode causar um efeito indesejado ao abrir a página em um dispositivo móvel.

Pense no viewport como uma meta tag que possibilita configurar resoluções customizadas ao abrir a página em determinado dispositivo.

Existem várias configurações que podemos declarar no atributo “content” da tag viewport, porém, os principais são:

width: define a largura da viewport.

height: define a altura da viewport.

initial-scale: define o zoom inicial da viewport.

Esses parâmetros podem ser utilizados de forma individual ou em conjunto no atributo “content” da tag viewport.

Essa meta tag foi uma solução para o Iphone na época e copiada pelos outros navegadores posteriormente, até a elaboração deste material a meta tag viewport não fazia parte da especificação oficial, ou seja, pode ser que alguns navegadores não suportem essa meta tag.

Para resolver esse problema adicione também no CSS usando a regra @viewport que é a forma oficial de utilizá-la sem grandes problemas.

Considere o seguinte cenário, em nosso html com layout fixo. Existe uma página chamada **index.html** e em sua folha de estilo **style.css** existe a seguinte declaração CSS:

```
body {
    width: 980px;
    margin: 0 auto;
}
```

O trecho de código CSS acima apenas define uma largura para o elemento body e o centraliza na tela. Pensando em acessibilidade considere abrir este HTML no IPhone 5 que tem uma resolução de width 320px. Qual é o efeito em abrir essa página em um IPhone 5 por exemplo? Certamente o navegador irá abrir, porém, temos que lembrar que foi definido no CSS um width 980px.

As combinações de configuração para viewport podem ser inúmeras pelo fato que existem vários dispositivos móveis com características diferentes e diante deste fato, já existe uma configuração mais indicada e que atende bem a maioria dos casos. O mais indicado para a maioria dos sites mobile é:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

A configuração width=device-width assumirá que o valor de width será o mais indicado para o aparelho com base na definição do fabricante, enquanto que o initial-scale=1 define que a página deve ser aberta no tamanho especificado.

Ajustando as medidas fixas para medidas flexíveis.

Para auxiliar na conversão de medidas fixas para medidas flexíveis existe uma formula simples, com base nesta formula, será garantido que quando convertermos uma font-size de 18px para **EM** a aparência da fonte ainda seja próxima de 18px.

A formula é: **Resultado = Elemento alvo / Contexto**

Resultado: O valor procurado

Elemento alvo: Elemento alvo com a medida atual

Contexto: Onde se enquadra o elemento alvo (com base no elemento pai)

Por exemplo:

Qual a equivalência em em de 18px em uma fonte que está configurada com 16px?

Resultado: Valor que queremos encontrar na unidade **EM** relativo a 18px.

Elemento alvo: font para 18px

Contexto: Temos já configurado como padrão uma fonte de 16px (configuração para todas as fontes).

Resultado = 18 / 16

Resultado = 1,125em

4. Media Queries

Já foi possível melhorar a responsividade, porém, ainda falta um item importante: **O layout deve se ajustar conforme a necessidade**. E é neste item que entram as Media Queries.

Antes de entrar no contexto de Media Queries é preciso se atentar aos tipos de mídias (Medias Types). Media Types, foi uma das primeiras iniciativas para permitir direcionar formatação CSS para determinado tipo de mídia e estão nas recomendações do W3C desde o CSS2. Com o auxílio das Medias Types é possível apresentar o site de maneira customizada de acordo com o tipo de mídia.

Relationship between media groups and media types

Media Types	Media Groups			
	continuous/paged	visual/audio/speech/tactile	grid/bitmap	interactive/static
braille	continuous	tactile	grid	both
embossed	paged	tactile	grid	static
handheld	both	visual, audio, speech	both	both
print	paged	visual	bitmap	static
projection	paged	visual	bitmap	interactive
screen	continuous	visual, audio	bitmap	both
speech	continuous	speech	N/A	both
tty	continuous	visual	grid	both
tv	both	visual, audio	bitmap	both

Figura 1 - Media types

Fonte: W3C

Exemplo de uso de Media Types com links para folhas de estilo específicas:

```
<link rel="stylesheet" type="text/css" media="print, handheld"
href="printHandheld.css">
```

No exemplo acima, o estilo CSS seria aplicado apenas se o tipo de dispositivo fosse um handheld ou se a página for impressa.

Exemplo de Media Types sendo aplicadas dentro do arquivo CSS:

```
@media screen, print {
	body {
		line-height: 1.2;
	}
}
```

Neste caso, o estilo CSS seria aplicado apenas se o tipo de dispositivo fosse screen ou se a página for impressa.

A estratégia de uso de Media Types, embora tenha ajudado ainda não resolveu o problema, pois atualmente a diferença entre os vários dispositivos tem diminuído. Um exemplo são os smarphones que atualmente possuem navegadores capazes de renderizar páginas web semelhante a desktops,

porém, a forma de navegar é completamente diferente que um desktop. Diante deste fato temos um cenário onde não se deve implementar um CSS especialista para smartphones e um CSS especialista para desktop, ou seja, olhando para os smartphones é possível identificar que tem características próprias de smartphones, porém, possuem algumas características também de desktop, temos uma zona mediana a considerar. Para cobrir essa lacuna a solução é a utilização de Media Queries.

Media Queries, diferente de Media Types, possuem a capacidade de separar dispositivos pelo **tamanho da tela** e não apenas por tipo de dispositivo. No contexto de web design responsivo o que devemos considerar realmente é o tamanho da tela. As Media Queries possuem recomendações desde 2012 e atualmente já se encontram no quarto nível desde sua padronização.

Exemplo de uso de Media Queries linkando folhas de estilos específicas:

```
<link rel="stylesheet" media="screen and (max-width: 800px)" href="style-800.css">
```

No exemplo acima foi indicado que um CSS será aplicado para dispositivos com tela até 800px.

Exemplo de Media Queries sendo aplicadas dentro do arquivo CSS:

```
@media screen and (max-width: 800px) {
	body {
		font-size: 70%;
	}
}
```

O número de parâmetros de Media Queries é extenso: **width, height, device-width, device-height, orientation, aspect-ratio, device-aspect-ratio, color, color-index, monochrome, resolution, scan, grid**.

É mais comum utilizar o Medias Queries dentro do código CSS, fica mais organizado que utilizar um CSS específico para cada situação. Em geral os parâmetros podem ser fixados com **max-** ou **min-** que representam respectivamente: **maior ou igual e menor ou igual a.**

width: Largura da janela útil do navegador. Para meios contínuos é o valor viewport considerando o tamanho do scroll, para meios páginados o tamanho considerado é o page box.

valor: <tamanho>

Media onde se aplica: visual e tátil

Aceita min/max: sim

Exemplo: @media screen and (min-width: 400px) and (max-width: 700px) { … }

height: Altura da janela útil do navegador. Para meios contínuos é o valor viewport considerando o tamanho do scroll, para meios páginados o tamanho considerado é o page box.

valor: <tamanho>

Media onde se aplica: visual e tátil

Aceita min/max: sim

Exemplo: @media screen and (min-height: 400px) and (max-height: 700px) { … }

device-width: Largura em pixel do dispositivo. Para meios contínuos é a largura da tela, para meios páginados é o tamanho da página.

valor: <tamanho>

Media onde se aplica: visual e tátil

Aceita min/max: sim

O exemplo abaixo aplicará determinado estilo somente para telas cujo a largura corrente está exatamente em 800px.

Exemplo: `@media screen and (device-width: 800px) { … }`

device-height: Altura em pixel do dispositivo. Para meios contínuos é a largura da tela, para meios páginados é o tamanho da página.

valor: <tamanho>

Media onde se aplica: visual e tátil

Aceita min/max: sim

O exemplo abaixo aplicará determinado estilo somente para telas cujo a altura corrente está exatamente em 800px.

Exemplo: `@media screen and (device-height: 800px) { … }`

orientation:

valor: <portrait | landscape>

Media onde se aplica: bitmap

Aceita min/max: não

O exemplo abaixo aplicará determinado estilo para dispositivos que estão em orietation portrait.

Exemplo: `@media all and (orientation: portrait) { … }`

aspect-ratio: Proporção da área visível do browser. Dois valores inteiros e positivos que representam a proporção em pixels na horizontal e vertical.

valor: <ratio>

Media onde se aplica: bitmap

Aceita min/max: sim

O exemplo abaixo aplicará determinado estilo quando a janela do navegador estiver com proporção 16:9

Exemplo: `@media screen and (aspect-ratio:16/9) { … }`

device-aspect-ratio: Define a proporção do display do dispositivo. Dois valores inteiros e positivos que representam a proporção em pixels na horizontal e vertical.

valor: <ratio>

Media onde se aplica: bitmap

Aceita min/max: sim

O exemplo abaixo aplicará determinado estilo quando o display estiver com proporção 16:9

Exemplo: `@media screen and (device-aspect-ratio:16/9) { … }`

color: Define o número de bits por componente de cor do dispositivo. Para dispositivos não coloridos o valor de default é 0.

valor: <inteiro>

Media onde se aplica: visual

Aceita min/max: sim

No exemplo abaixo aplicará determinado estilo para dispositivos coloridos.

Exemplo:

```
@media all and (color) { … }
```

No exemplo abaixo aplicará determinado estilo para dispositivos que possuem pelo menos 1 bit de cor.

Exemplo:

No exemplo abaixo aplicará determinado estilo para dispositivos que

```
@media all and (min-color: 1) { … }
```

color-index: Define o número de entradas na tabela de cores do dispositivo. Para dispositivos que não possuem tabela de cores o valor é 0.

valor: <inteiro>

Media onde se aplica: visual

Aceita min/max: sim

No exemplo abaixo aplicará determinado estilo para dispositivos com pelo menos 1 cor indexada.

Exemplo: `@media all and (min-color-index: 1) { … }`

monochrome: Define o número de bits por pixel em medias monocromáticas. Caso o dispositivo não seja monocrático o valor default é 0.

valor: <inteiro>

Media onde se aplica: visual

Aceita min/max: sim

O exemplo abaixo aplicará determinado estilo em dispositivos com mais de 1 bit por pixel.

Exemplo: `@media all and (min-monochrome: 1) { … }`

resolution: Define a resolução do dispositivo. É possível especificar em duas unidades (dpi ou dpcm). A unidade dpi é pontos por polegada e a unidade dpcm é pontos por centímetro.

valor: <inteiro>

Media onde se aplica: bitmap

Aceita min/max: sim

No exemplo abaixo aplicará determinado estilo em dispositivos com resolução maior que 300dpi.

Exemplo: `@media print and (min-resolution: 300dpi) { … }`

Em dispositivos com pixels não-quadrados, a resolução não atua sem os prefixos **min** e **max.**

scan: Define o processo de escaneamento para dispositivos TV.

valor: <progressive | interlace>

Media onde se aplica: TV

Aceita min/max: no

Exemplo:

O exemplo abaixo aplicará determinado estilo em dispositivos TV com escaneamento progressivo.

```
@media tv and (scan:progressive){...}
```

grid: Determina se o dispositivo é grade ou bitmap. Dispositivos em grade são: terminal TTY ou display de um telefone, com fonte única. Pare estes o valor será 1, caso contrário valor será 0.

valor: <inteiro>

Media onde se aplica: visual e tátil

Aceita min/max: no

No exemplo abaixo aplicará determinado estilo caso o dispositivo seja móvel e com display de 10 caracteres ou menos.

Exemplo:

```
@media handheld and (grid) and (max-width: 10em) {...}
```

Unidades

Media queries utilizam-se das mesmas unidades utilizadas em CSS. É importante se atentar que unidades relativas em Media queries são baseadas em valores iniciais e não em resultados de declarações CSS. Por exemplo: No HTML o **em** é relativo ao valor inicial de font-size que na maioria dos navegadores é de 16px, caso não seja definido.

Operadores Lógicos

Media Queries aceitam operadores lógicos, os operadores aceitos em Media Queries são: **and** e **or**, **not**, **only.**

and e or: Pode ser utilizada para combinar uma lista de Media Queries, por exemplo:

```
@media screen and (color), print and (color) {...}
```

No exemplo acima, será aplicado um determinado estilo para telas e impressões coloridas.

not: Pode ser utilizada para negar algum resultado. Este operador não pode ser utilizado para negar uma query individual.

No exemplo abaixo aplicará uma determinada folha de estilo para todas as medias não monocromáticas

Exemplo: `@media not all and (monochrome) { ... }`

only: Pode ser utilizada para seleção de folhas de estilo para navegadores mais antigos por exemplo:

```
<link rel="stylesheet" media="only screen and (color)" href="example.css">
```

Considerações sobre Media Queries

Bom, com os parâmetros de Media Queries já apresentados é possível avançar na busca de um layout responsivo. É perceptível a grande variedade de parâmetros que podem ser utilizados quando se trata de Media Queries, atualmente não é necessário o uso intenso de todos os parâmetros de Media Queries, à medida que se vai ganhando experiência em Media Queries a dificuldade enfrentada por desenvolvedores menos experientes e também mais experientes é: **Quais pontos de Interrupção deve-se utilizar?** Os famosos **breakpoints**.

Breakpoints são pontos de interrupção que irão delimitar as regras CSS permitindo assim atender diferentes tipos de dispositivos, ou seja, são pontos que são definidos ao usar parâmetros de Media Queries. Talvez uma das decisões mais difíceis em um projeto onde se deve suportar múltiplos meios de acesso é a definição desses breakpoints, uma vez que irá impactar diretamente no conteúdo a ser apresentado, por isso essa decisão deve ser muito bem pensada.

Abaixo seguem algumas alternativas a considerar para auxiliar na configuração de breakpoints:

Baseado em Mínimo ou Máximo Width

Pode-se assumir um min-width e max-width e em cima desta definição implementar as regras CSS específicas. Olhando pela primeira configuração (min-width) é possível iniciar a implementação das regras CSS a partir de tamanhos menores (Visão Mobile First) e com max-width é possível implementar regras específicas para telas maiores.

Baseado no Conteúdo

Antes de aplicar essa abordagem lembramos de uma frase famosa em web design responsivo: “O Conteúdo é o rei”. O ponto chave desta abordagem é definir quando o conteúdo deve se adaptar e não de acordo com tamanho de tela, resolução entre outros. Esta abordagem é a ideal, porém, é uma decisão muito particular do projeto, pois existem necessidades distintas e que mudam de projeto para projeto.

Baseado em em não em px

Principalmente quando estamos construindo Medias Queries com min or max witdth ou height é utilizada a unidade em Pixel (px), porém, não existem impedimentos em usar Media Queries baseadas em **em.**

Ao utilizar breakpoints baseados em **em** o layout pode se ajustar baseado em alterações de font-size, além de ajudar no zoom in e zoom out esse tipo de abordagem pode auxiliar a definir breakpoints ainda mais baseados em conteúdo.

Na web existem várias discussões sobre breakpoints e inúmeras formas de aplicar tal conceito, no entanto ainda não há uma formula mágica que defina qual ou quais os melhores breakpoints utilizar, porém, existem alguns breakpoints que são chaves para auxiliar pelo menos na fase inicial de um projeto, abaixo será listado alguns breakpoints mais conhecidos segundo o livro de Tárcio Zemel:

320 and Up

É um conjunto de valores comuns prática conhecida como device-driven-breakpoints, que são valores atribuídos de acordo com o tamanho dos dispositivos.

Less Framework

Breakpoints utilizados pelo Less Framework, que separa por tipo de dispositivo.

```
/* Tablet Layout */
@media only screen and (min-width: 768px)
and (max-width: 991px) { ... }
/* Mobile Layout */
@media only screen and (max-width: 767px) { ... }
/* Layout largo de mobile */
@media only screen and (min-width: 480px)
and (max-width: 767px) { ... }
/* Retina display */
@media
only screen and (-webkit-min-device-pixel-ratio: 2),
only screen and (min-device-pixel-ratio: 2) { ... }
```

Skeleton

Já possui um conjunto de breakpoints baseados mais na orientação do dispositivo (landscape ou portrait).

```
/* Menor que 960 */
@media only screen and (max-width: 959px) { ... }
/* Tablet Portrait ao padrão 960 */
@media only screen and (min-width: 768px)
and (max-width: 959px) { ... }
/* Todos tamanhos de mobile */
@media only screen and (max-width: 767px) { ... }
/* Mobile em landscape a tablet Portrait */
@media only screen and (min-width: 480px)
and (max-width: 767px) { ... }
/* Mobile em portrait a mobile em landscape */
@media only screen and (max-width: 479px) { ... }
```

Twitter Bootstrap

Os breakpoints default do Twitter Bootstrap já cobrem uma grande variedade de dispositivos a pesar dos poucos breakpoints em relação aos que já foram citados.

```
/* Telefones em landscape e abaixo */
```

```
@media (max-width: 480px) { ... }
/* Telefone em landscape a tablet em portrait */
@media (max-width: 767px) { ... }
/* tablet em portrait a landscape e desktop */
@media (min-width: 768px) and (max-width: 979px) { ... }
/* Desktop grande */
@media (min-width: 1200px) { ... }
```

Bom, com base nos breakpoints chave é possível se inspirar em algum e iniciar o projeto, vale ressaltar que essa escolha deve ser bem analisada e sempre visando uma melhor adequação ao projeto.

Características importantes sobre Media Queries

- Para parâmetros desconhecidos os navegadores consideram **not all** para esses parâmetros.
  Exemplo:
  `@media (max-orientation:portrait) {...}`
  A consulta acima será toda negada pois há erro de sintaxe, neste caso **max-orintation** está incorreto, pois o parâmetro **orientation** não aceita **min** e **max.**
- Para valores desconhecidos o conceito é o mesmo, a consulta é toda negada.
  Exemplo:

  ```
  @media (max-width: 2apx) {...}
  ```

  A consulta acima será interpretada como **not all**, pois **max-width** deve ser um valor inteiro.

- Para consultas mal formadas o conceito é o mesmo, a consulta é toda negada.
  Exemplo:
  `@media (example , all , ), screen {...}`

  A consulta acima será interpretada como **not all** pois há um token inexperado para consulta.

- Recomenda-se a utilização de **apenas uma folha de estilo**. É possível separar as medias queries em vários arquivos, por exemplo, um arquivo específico para cada breakpoint, porém, é recomendado utilizar apenas uma folha de estilo.
- Medias Queries podem ser **empilhadas** ou **sobrepostas.**
  Exemplo:

  ```
  @media all and (min-width:300px) {
        .leftMenu {
              background:blue;
        }
  }

  @media all and (min-width:500px) {
        .leftMenu {
              background:red;
        }
  }
  ```

  No exemplo acima, caso o viewport seja 600px as duas regras serão aplicadas correto? Sim. E isso caso não seja o esperado irá causar muitos problemas. Não é errado usar desta forma, porém, deve se pensar se realmente este comportamento é o esperado.

Exemplo:

```
@media all and (min-width:500px) {
	.leftMenu {
		background:red;
	}
}
@media all and (min-width:300px) {
	.leftMenu {
		background:blue;
	}
}
```

Neste exemplo, considerando que o viewport seja 600px, a regra que será aplicada é:

```
@media all and (min-width:300px) {
	.leftMenu {
		background:blue;
	}
}
```

Lembrando que ainda estamos em CSS então a **cascata** e a **especificidade** ainda tem relevância.

Empilhando Media Queries

Exemplo:

```
@media all and (min-width:300px) and (max-width:499px) {
	.leftMenu {
		background:blue;
	}
}
@media all and (min-width:500px) {
	.leftMenu {
		background:red;
	}
}
```

No exemplo acima empilhamos duas Media Queries, sendo que na primeira query será aplicado determinado estilo para todas as medias que possuem largura de no mínimo 300px a 499px, enquanto que para medias com largura de no mínimo 500px será aplicado outro estilo.

Sobrepondo Media Queries

Exemplo:

```
@media all and (min-width:500px) {
	.leftMenu {
		background:red;
	}
}
@media all and (min-width:300px) {
	.leftMenu {
		background:blue;
```

```
	}
}
```

No exemplo sobrepomos as media queries, ou seja, neste caso o estilo que será aplicado é:

```
@media all and (min-width:300px) {
	.leftMenu {
		background:blue;
	}
}
```

As duas técnicas têm suas vantagens e desvantagens, deve-se avaliar em termos de manutenção de código, se há muitas diferenças entre a versão mobile e versão desktop, complexidade de código, entre outras.

Suporte Media Queries para versões antigas de Internet Explorer

No caso do Internet explorer, versões anteriores a versão 9 não suportam Media Queries, diante deste fato, temos duas opções:

- Uso de comentários condicionais.
  No blog da Microsoft existem algumas alternativas e mais detalhes sobre as versões do Internet Explorer, inclusive para Windows Phone que não suportam CSS3 Medias Queries.
- Uso de Javascript.
  Existem várias soluções em javascript para cobrir essa lacuna, dentre as várias soluções em javascript existente segue abaixo a mais conhecidas:
  CSS3-mediaqueries.js e Respond.

Conclusão

Passamos pelas tecnologias que possibilitam o desenvolvimento de web sites, aplicações web mais acessíveis independente de dispositivo, porém, foi visível que tecnicamente é simples a utilização dos pilares tecnológicos para implementar design responsivo que são: **Layout Fluído**, **Recursos Flexíveis** e **Media Queries**. A maior dificuldade está no planejamento, ou seja, é em como, onde e quando utilizá-las.

Para ajudar no desenvolvimento de web design responsivo considere utilizar ferramentas para acelerar neste processo, existem vários frameworks que podem ajudar a desenvolver layouts responsivos com agilidade, simplicidade e qualidade.

Alguns dos principais frameworks para auxiliar no processo de desenvolvimento visando layout responsivo são:

Twitter Bootstrap 320 and UP Foundation 3 Skeleton

5. Persistência de dados lado cliente

LocalStorage e SessionStorage

Até o HTML4, quando precisávamos armazenar dados no agente de usuário (navegador) que persistissem entre as páginas, usávamos Cookies. Cookies nos permitiam armazenar o status de um menu javascript que precisava ser mantido entre as páginas, lembrar o nome do usuário, o histórico de operações realizadas por ele ou a última vez que ele visitou nosso site.

Com o aumento da complexidade das aplicações baseadas em web, duas grandes limitações dos Cookies nos incomodam:

Interface complexa: o código para armazenar Cookies envolve complexos cálculos com datas e controle do nome de domínio.

Limite de armazenamento: alguns agentes de usuário permitiam o armazenamento de no máximo 20 Cookies, com apenas 4KB cada.

O HTML5 traz uma nova maneira de armazenar dados no client, a API Storage.

Um objeto Storage possui os métodos:

- **getItem**(key): obtém um valor armazenado no Storage
- **setItem**(key,value) guarda um valor no Storage
- **removeItem**(key) exclui um valor do Storage
- **clear**() limpa o Storage

A especificação da API Storage rege que qualquer valor Javascript pode ser armazenado e recuperado, porém, em alguns navegadores os valores são convertidos para strings assim como nos Cookies. As APIs de storage podem ser acessadas via escopo global (window) e possuem duas implementações: **localStorage** e **sessionStorage**.

O objeto **localStorage** armazena os dados no client sem expiração definida. Ou seja, se o usuário fechar o navegador e voltar ao site algum tempo depois os dados estarão lá.

O **sessionStorage** armazena os dados durante a sessão atual de navegação.

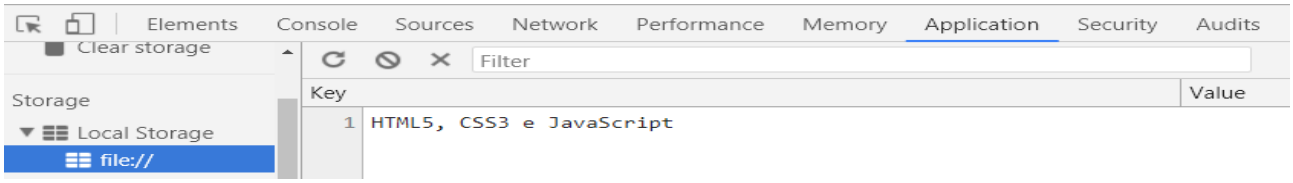
O código para armazenar um valor na Storage é o seguinte:

```
localStorage.setItem("DM121","HTML5, CSS3 e JavaScript");
```

E quando precisar ler esse valor:

```
console.log(localStorage.getItem("DM121"));
```

Nas ferramentas de desenvolvedor de cada navegador é possível verificar se o dado está realmente sendo persistido, no google chrome é possível ver o localStorage no menu Application:



Quando se usa **localStorage** ou **sessionStorage** o dono do armazenamento é o domínio, em nosso caso estamos rodando a partir do file system então o dono da informação armazenada é o **file://**

O espaço de armazenamento sugerido pela documentação é de **5MB** para cada domínio, outro ponto importante é que tanto o localStorage quanto sessionStorage são síncronos, ou seja, enquanto não termina a operação a próxima linha do javascript não será executada.

Web SQL Database

Utiliza conceitos de banco de dados relacional e é suportado pela maioria dos navegadores. Porém, está ***deprecated*** (https://www.w3.org/TR/webdatabase/) seu uso deve ser fortemente avalidado.

IndexedDB

É outra alternativa de armazenamento do lado cliente, no entanto nem todos os navegadores suportam 100% das features disponíveis. Para mais detalhes de compatibilidade acesse aqui.

Basicamente os dados são armazenados em forma de objetos indexados por uma chave e as operações são por meio de transações. Diferentemente do localStorage e sessionStorage o IndexedDB consegue armazenar maior quantidade de informações e de maneira estruturada e as consultas tem performance satisfatória.

Observe a persistência de dados com IndexedDB:

```
// verifica se suporta indexdb
var indexedDB = window.indexedDB || window.mozIndexedDB ||
window.webkitIndexedDB || window.msIndexedDB || window.shimIndexedDB;

// Abre ou cria uma conexão
var DB_NAME = "DISCIPLINAS";
var OBJECT_STORE = "ObjectStore";
var INDEX_NOME = "NameIndex";
var open = indexedDB.open(DB_NAME, 1);

// Cria o esquema
open.onupgradeneeded = function() {
    var db = open.result;
    var store = db.createObjectStore(OBJECT_STORE, {keyPath:
"nome"});//indexando pelo nome
```

```
    var index = store.createIndex(INDEX_NOME, ["nome", "titulo"]);
//criando o indice em dois campos
};

open.onsuccess = function() {
    // Inicia uma nova transação
    var db = open.result;
    var tx = db.transaction(OBJECT_STORE, "readwrite");
    var store = tx.objectStore(OBJECT_STORE);
    var index = store.index(INDEX_NOME);

    // Adicionando dados
    var items = [
        {
            nome: 'DM121',
            titulo: 'HTML5, CSS3 e Javascript',
            data: new Date().toISOString(),
            concluido:false
        },
        {
            nome: 'DM122',
            titulo: 'Desenvolvimento Híbrido',
            data: new Date().toISOString(),
            concluido:false
        }
    ];

    //Adicionando dados
    for (var i in items) {
        store.put(items[i]);
    }

    // Query the data
    var dm121 = store.get("DM121");
```

```
    var dm122 = index.get(["DM122", "Desenvolvimento Híbrido"]);

    dm121.onsuccess = function() {
        console.log(dm121.result);  // DM121
    };

    dm122.onsuccess = function() {
        console.log(dm122.result);   //DM122
    };

    // Fechando a trasação
    tx.oncomplete = function() {
        db.close();
    };
}
```

6. REFERÊNCIAS

ECMAScript. Disponível em < http://www.ecma-international.org/>. Acesso em 14 Jan. 2018.

Mozilla Developer. Disponível em < https://developer.mozilla.org />. Acesso em 05 Mar. 2018.

Caniuse. Disponível em <https://caniuse.com >. Acesso em 03 Abr. 2018.

Bootstrap. Disponíve em <https://getbootstrap.com/>. Acesso em 08 Abr 2018.