

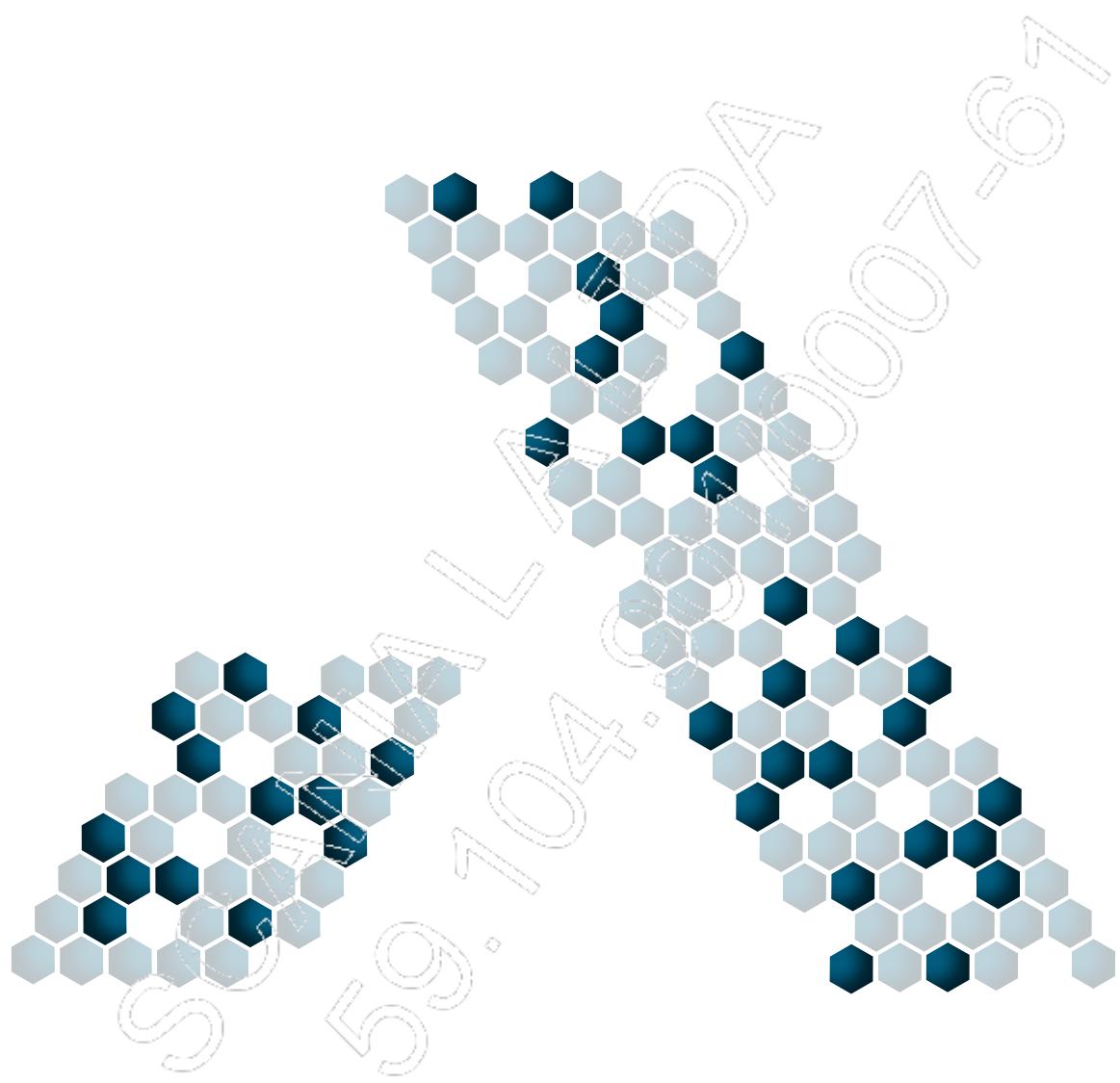


Desenvolvimento Front End com HTML5, CSS3 e JavaScript



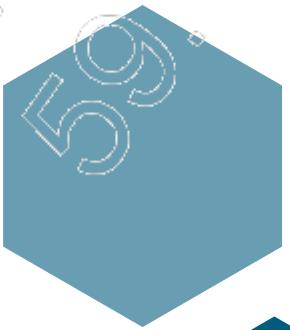
ALTA
TDA
10002
904
704
SCANIA



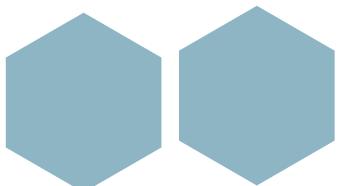




Desenvolvimento Front End com HTML5, CSS3 e JavaScript



Editora
IMPACTA



Créditos

Copyright © Monte Everest Participações e Empreendimentos Ltda.

Todos os direitos autorais reservados. Este manual não pode ser copiado, fotocopiado, reproduzido, traduzido ou convertido em qualquer forma eletrônica, ou legível por qualquer meio, em parte ou no todo, sem a aprovação prévia, por escrito, da Monte Everest Participações e Empreendimentos Ltda., estando o contrafator sujeito a responder por crime de Violação de Direito Autoral, conforme o art.184 do Código Penal Brasileiro, além de responder por Perdas e Danos. Todos os logotipos e marcas utilizados neste material pertencem às suas respectivas empresas.

"As marcas registradas e os nomes comerciais citados nesta obra, mesmo que não sejam assim identificados, pertencem aos seus respectivos proprietários nos termos das leis, convenções e diretrizes nacionais e internacionais."

Desenvolvimento Front End com HTML5, CSS3 e JavaScript

Coordenação Geral

Henrique Thomaz Bruscagin

Autoria

Emilio Celso de Souza

Revisão Ortográfica e Gramatical

Fernanda Monteiro Laneri

Diagramação

Bruno de Oliveira Santos

Edição nº 1 | 1892_0

Junho/ 2020

Este material constitui uma nova obra e é uma derivação da seguinte obra original, produzida por Monte Everest Participações e Empreendimentos Ltda., em Fev/2017: **Desenvolvendo Aplicações Web com HTML5, CSS3, JavaScript e PhoneGap**

Autoria: Emilio Celso de Souza

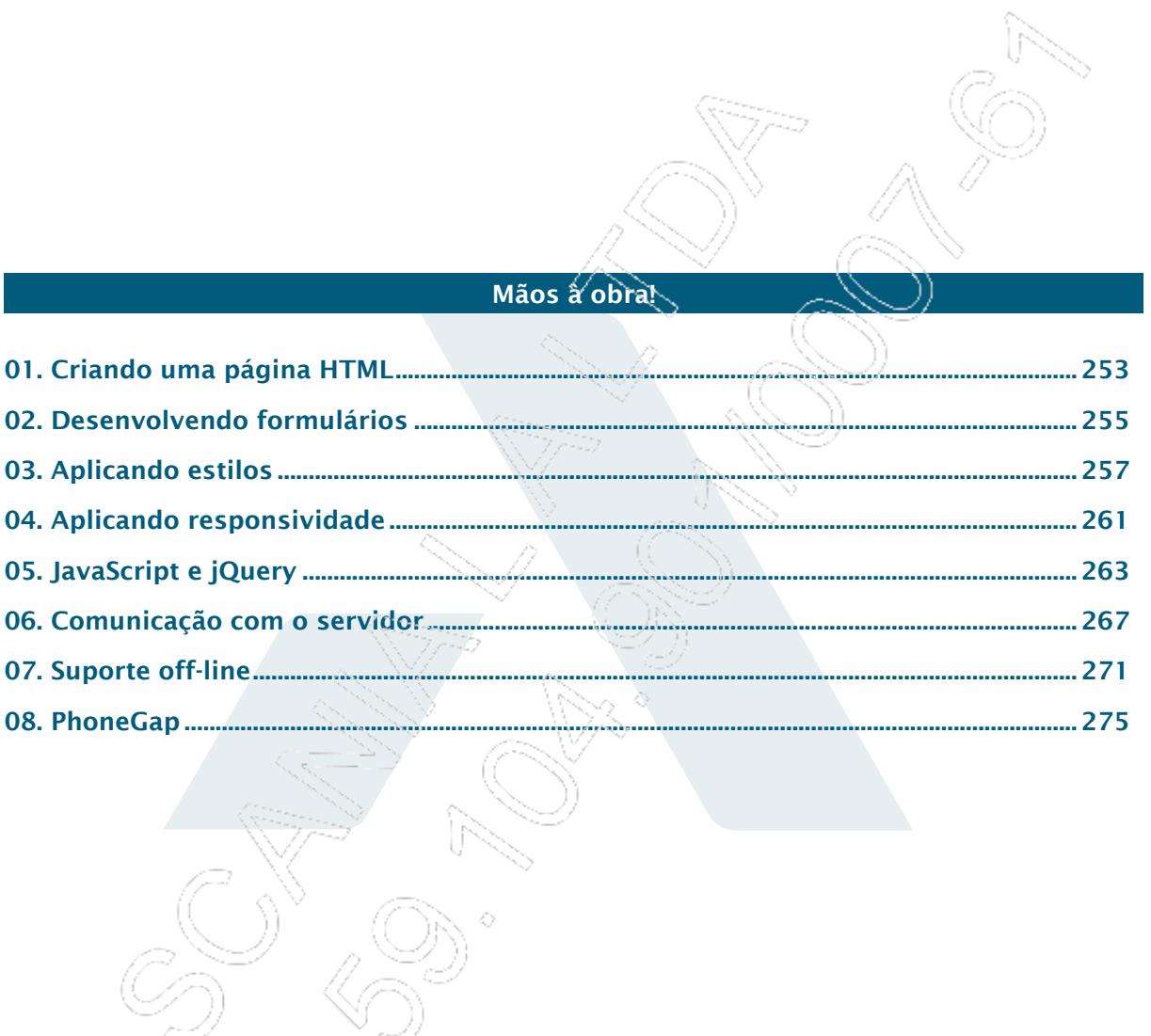
Sumário

Conteúdo de referência

Introdução	07
Conteúdo programático	09
01. Visão geral do HTML5	15
02. Estilos com CSS3.....	31
03. Responsividade com media queries.....	43
04. Conceitos do JavaScript e do Node.js	51
05. JavaScript - Chamadas assíncronas e promises.....	69
06. Conceitos do jQuery.....	83
07. Otimizações de layout com Bootstrap	97
08. Animações em telas de fundo.....	109
09. Elementos de áudio e vídeo.....	117
10. Comunicação com serviços REST	121
11. Criando objetos com JavaScript - Prototype	129
12. Criando páginas interativas e suporte off-line	137
13. Complemento: Aplicações híbridas com PhoneGap	147

Atividades

Apresentando o projeto	165
01. Preparando o ambiente.....	167
02. Criação de páginas com HTML5.....	169
03. Inclusão de formulários.....	173
04. Aplicação de estilos com CSS3	179
05. Desenvolvimento de responsividade com media queries.....	189
06. Inclusão de funcionalidades JavaScript e jQuery	195
07. Consumo de serviços REST	199
08. Otimizações de layout com Bootstrap	207
09. Inclusão e listagem de registros com Web services	223
10. Aplicação de Canvas API	233
11. Uso de áudio e vídeo.....	237
12. Armazenamento local	239
13. Complemento: PhoneGap	245



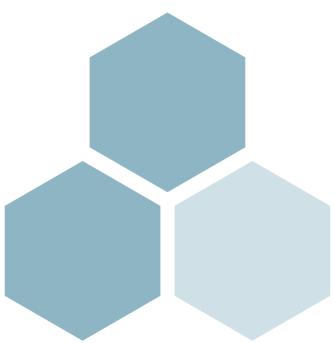
Mãos à obra!

01. Criando uma página HTML.....	253
02. Desenvolvendo formulários	255
03. Aplicando estilos	257
04. Aplicando responsividade	261
05. JavaScript e jQuery	263
06. Comunicação com o servidor.....	267
07. Suporte off-line.....	271
08. PhoneGap	275



Introdução

SCAMIA ALTA
59.704.967-0007-67



Introdução

Este curso tem como objetivo apresentar e aplicar, de forma bastante prática, os conceitos de HTML5, CSS3 e JavaScript, visando a capacitação no desenvolvimento de aplicações responsivas.

Para todas as aulas, os alunos poderão contar com estas referências on-line:

http://www.w3schools.com/html/html5_intro.asp
<https://www.w3schools.com/css/default.asp>
<http://www.w3schools.com/js/>
<http://www.w3schools.com/jquery/>
<http://docs.phonegap.com/>





Conteúdo programático

SCAMVIA
59.704.
ALTA
10007-67



Editora
IMPACTA





Conteúdo programático – 1/10

1 - Visão geral do HTML5

- Estrutura do documento
 - Modelos de conteúdo
 - Conteúdo interativo
 - Novos elementos e atributos
 - Campos de entrada
 - Validadores



Conteúdo programático – 2/10

2 - Estilos com CSS3

- Seletores
 - Gradientes
 - Bordas
 - Transições e animações



Conteúdo programático - 3/10

3 - Responsividade com media queries

- Elemento @media
- Viewport



L
T
D
A
9
0
7
1
0
0
0
7
6
7

Conteúdo programático - 4/10

4 - Conceitos do JavaScript e do Node.js

- Introdução ao JavaScript
- Instruções do JavaScript
- Conceitos do Node.js
- Execução de programas JavaScript no HTML, no browser e no Node.js
- Variáveis
- Uso do var e do let
- Funções
- Comandos de decisão
- Comando switch
- Estruturas de repetição





Conteúdo programático – 5/10

- Objetos do JavaScript
- O formato JSON
- Integração do HTML com JavaScript
- Eventos em campos de formulários



Conteúdo programático – 6/10

5 - JavaScript - Chamadas assíncronas e promises

- Chamadas assíncronas
- Promises
- Aplicação - A função fetch

6 - Conceitos do jQuery

- Conceitos do jQuery
- Funções e eventos
- Iterações com \$.each



Conteúdo programático - 7/10

7 - Otimizações de layout com Bootstrap

- Usando Bootstrap
- Formulários e menus
- Personalização

8 - Animações em telas de fundo

- Background animado
- SVG
- Canvas API



LTD A 67
10007-907
C A M P A G N A
L T D A

Conteúdo programático - 8/10

9 - Elementos de áudio e vídeo

- Áudio
- Vídeo

10 - Comunicação com serviços REST

- Enviando e recebendo dados com JavaScript
- Enviando e recebendo dados com jQuery



Conteúdo programático – 9/10

11 - Criando objetos com JavaScript - Prototype

- Objetos customizados
- Uso do prototype

12 - Criando páginas interativas e suporte off-line

- Session storage
- Local storage
- IndexedDB



Conteúdo programático – 10/10

13 - Complemento: Aplicações híbridas com PhoneGap

- O Apache Cordova
- Projetos baseados no PhoneGap com Node.js
- O PhoneGap Build

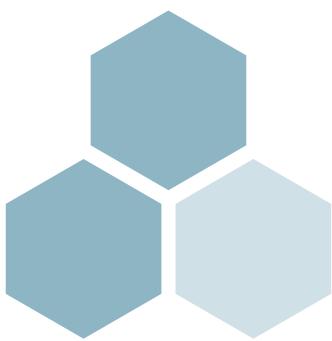




1

Visão geral do HTML5

SCAMILLA
59.704.
SANTO DOMINGO
LTDA
10007-67





Estrutura do documento

Todo documento HTML5 possui a mesma estrutura formal:

- Declaração DOCTYPE
- Seção HTML, contendo:
 - **header**
 - **body**

```
<!DOCTYPE html>
<head>
...
</head>
<body>
...
</body>
```



Estrutura do documento

- Os elementos HTML definem a estrutura e a semântica do conteúdo de uma página
- Estes elementos podem ser aninhados a outros elementos. Por exemplo:

```
<div>
  <h1>Elemento aninhado ao div</h1>
  <strong>Outro elemento</strong>
</div>
```

- É permitido a alguns elementos incluir atributos para fornecer informações adicionais:

```

```



DOCTYPE

- Cada versão do HTML possui sua própria versão para a declaração DOCTYPE
- Para o HTML5, basta incluir no início do arquivo:

```
<!DOCTYPE html>
```
- Bem diferente de versões anteriores:
 - **HTML 4.0.1 Strict:** Inclui todos os elementos e atributos, mas não inclui elementos depreciados:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

- **HTML 4.0.1 Frameset:** Igual ao anterior, incluindo o uso de frameset:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">
```

Modelos de conteúdo

- A estrutura do HTML5 permanece a mesma das versões anteriores, com alterações no DOCTYPE (como já apresentado anteriormente)
- Os elementos se classificam, de uma forma geral, como:
 - **Elementos de linha:** São elementos que possuem seu fechamento na mesma instrução de abertura
 - **Elementos de bloco:** São elementos que abrem e fecham em instruções diferentes, podendo comportar outros grupos de elementos como conteúdo, o que também podemos chamar de **subelementos**





Modelos de conteúdo

Os principais elementos são apresentados a seguir:

- Cabeçalhos e parágrafos:

```
<h1>Meu cabeçalho</h1>
<p>Este texto terá um destaque</p>
<h2>Outro parágrafo, um nível abaixo de h1</h2>
```

- Dando ênfase:

Para **strong**destacar este texto**/strong** usamos o elemento apresentado. **em**Este texto aparecerá em itálico**/em**



Modelos de conteúdo

- Apresentando listas não numeradas:

```
<ul>
  <li>Primeiro elemento</li>
  <li>Segundo elemento</li>
</ul>
```

- Listas numeradas:

```
<ol>
  <li>Primeiro elemento</li>
  <li>Segundo elemento</li>
</ol>
```



Modelos de conteúdo

- Apresentando links:

```
<a href="conteudo.html" alt="Conteúdo interativo">Ver conteúdo</a>
```

- Apresentando imagem:

```

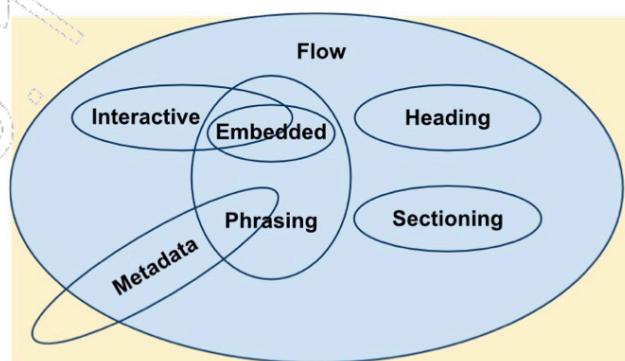
```



Modelos de conteúdo

Os elementos HTML podem ou não pertencer a algum grupo contendo características similares.

- Conteúdo de **metadados**
- Conteúdo de **fluxo**
- Conteúdo de **seccionamento**
- Conteúdo do **cabeçalho**
- Conteúdo **fraseado**
- Conteúdo **embutido**
- Conteúdo **interativo**



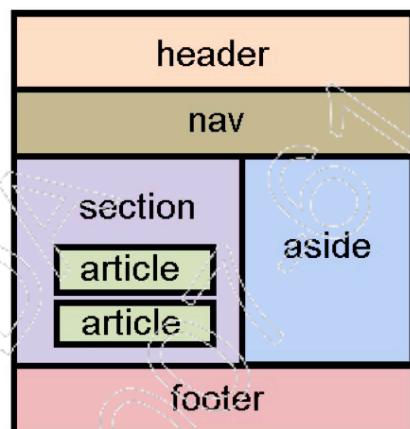
Fonte: https://developer.mozilla.org/pt-BR/docs/Web/Guide/HTML/Categorias_de_conteudo



Modelos de conteúdo

As tags `<section>`, `<header>`, `<nav>`, `<article>` e `<footer>` foram adicionadas ao HTML5 para permitir uma melhor semântica, especialmente quando desejamos aplicar estilos separadamente a cada um deles, ou a um grupo.

Podemos considerar a ilustração ao lado para decidir como posicionar esses elementos na página:



Conteúdo interativo

Conteúdo interativo inclui os elementos que possuem a finalidade de interagir com o usuário. Esses elementos incluem botões, caixas de textos, imagens, menus, links, entre outros.

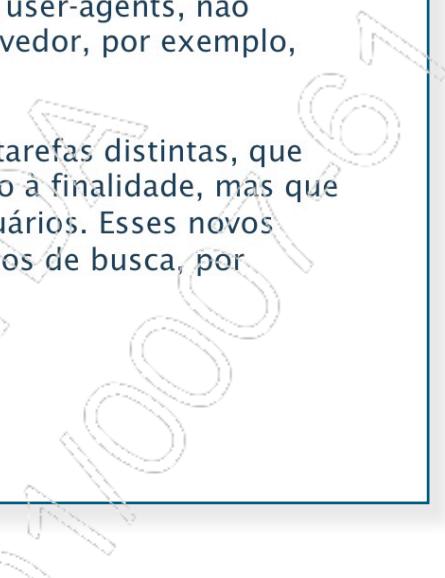
Veja a lista dos principais elementos interativos:

- `<audio>`, se o atributo `controls` estiver presente
- ``, se o atributo `usemap` estiver presente
- `<input>`, se o atributo `type` não estiver no modo escondido (`hidden`)
- `<menu>`, se o atributo `type` estiver no modo barra de ferramentas
- `<object>`, se o atributo `usemap` estiver presente
- `<video>`, se o atributo `controls` estiver presente

Fonte: https://developer.mozilla.org/pt-BR/docs/Web/Guide/HTML/Categorias_de_conteudo

Novos elementos e atributos

- No HTML tradicional, os robôs de busca, ou user-agents, não diferenciavam elementos; cabia ao desenvolvedor, por exemplo, diferenciar um cabeçalho de um rodapé
- O HTML5 agregou diversos elementos com tarefas distintas, que muitas vezes se parecem no que diz respeito à finalidade, mas que são úteis em tarefas não visíveis para os usuários. Esses novos elementos agilizam o trabalho de mecanismos de busca, por exemplo



Novos elementos e atributos

- Outra característica importante que o HTML5 trouxe foi a habilidade de interagir com o usuário de forma mais simplificada. Por exemplo, temos uma caixa de textos específica para datas, o que não era possível em versões anteriores
- Com isso, temos novos elementos disponíveis:



Novos elementos e atributos

Elemento	Definição
section	Permite definir novas seções no documento, em diversas partes
nav	Representa uma seção da página com elementos de naveabilidade, como links e menus
article	Representa uma parte do documento que pode ser reutilizada, como posts
aside	Permite criar uma parte do conteúdo separada do conteúdo principal
hgroup	Permite agrupar títulos, quando forem representados por vários níveis
header	Representa um grupo de introdução ou mesmo elementos de navegação
footer	Semelhante ao header, mas usado como último elemento da página
time	Serve para marcar uma parte do documento para apresentar uma data



Novos elementos e atributos

- Vários elementos ganharam novos atributos, cuja finalidade foi expandir suas funcionalidades. Vejamos os principais:



Novos elementos e atributos

Atributo	Aplicado a
<code>autofocus</code>	Elementos <code>input</code> , <code>textarea</code> , <code>select</code> e <code>button</code>
<code>media</code>	Juntamente com a tag <code>link</code> , no elemento <code>a</code>
<code>novalidate</code>	Elemento <code>form</code> (para cancelar a validação)
<code>reversed</code>	<code>ol</code> (mostra a lista na ordem inversa)
<code>disabled</code>	<code>fieldset</code>
<code>placeholder</code>	Elementos <code>input</code> e <code>textarea</code>
<code>hreflang</code> , <code>rel</code>	<code>area</code>
<code>target</code>	<code>base</code>



Novos elementos e atributos

No link abaixo é possível encontrar os novos elementos com mais detalhes, bem como os elementos e atributos descontinuados.

<<http://www.w3.org/TR/2014/NOTE-html5-diff-20141209/>>

É importante analisar esses dados para não incluirmos por engano elementos que não mais são úteis

- Elaborar a Atividade 1 – Preparando o ambiente
- Elaborar a Atividade 2 – Criação de páginas com HTML5





Campos de entrada

- Novos elementos: **tel**, **search**, **email**, **url**, **date**, **number**, **range**, **color**
- Validadores: **required**, **maxlength**, **pattern**, **autofocus**, **placeholder**
- Desenvolver formulários é uma das tarefas mais comuns no desenvolvimento Web. O HTML5 incluiu novos elementos e validadores para agilizar e facilitar esta tarefa



Campos de entrada

- Neste tópico, vamos explorar o uso de formulários, seus elementos e como os novos elementos são usados

Exemplo 1 – Coletando dados de um usuário



Campos de entrada

```
<form name="LoginUsuario" method="post" action="login.aspx">
    <fieldset>
        <legend>Detalhes do login</legend>
        <div id="username" class="field">
            <label for="uname">Nome do usuário:</label>
            <input id="uname" name="username" type="text" placeholder="Nome e sobrenome"/>
        </div>
        <div id="password" class="field">
            <label for="pwd">Senha do usuário:</label>
            <input id="pwd" name="password" type="password" placeholder="Sua senha" />
        </div>
    </fieldset>
    <input type="submit" value="Enviar"/>
</form>
```



Campos de entrada

- Formulário em execução

Detalhes do login
Nome do usuário: Nome e sobrenome
Senha do usuário: Sua senha

Enviar





Campos de entrada

- date

```
<label for="dnasc">Data Nasc.:</label><br />
<input id="dnasc" name="dnasc" type="date" />
```

Data Nasc.:

dd/mm/aaaa ▾

dezembro de 2016 ▾

dom	seg	ter	qua	qui	sex	sáb
27	28	29	30	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31



Campos de entrada

- range

```
<label for="tempo">Tempo de Residência:</label><br />
<input id="tempo" name="tempo" type="range" min="1" max="20" step="0.5" />
```

Tempo de Residência:



Campos de entrada

- number

```
<label for="filhos">Num. Filhos:</label><br />
<input id="filhos" name="filhos" type="number" min="0" max="10" />
```

Num. Filhos:
8

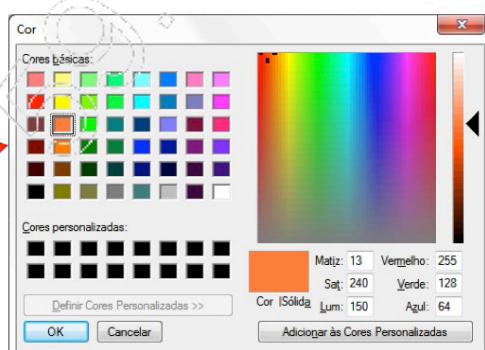
Treinamentos
IMPACTA

Campos de entrada

- color

```
<label for="cor">Cor preferida:</label><br />
<input id="cor" name="cor" type="color" />
```

Cor preferida:



Cor preferida:

Treinamentos
IMPACTA

Campos de entrada

- Os demais elementos possuem o aspecto de uma caixa de textos e são importantes quando se trata de validações



Validadores

- Os validadores são usados extensivamente por componentes de formulários que necessitam de tratamento especial antes de serem submetidos
- Mesmo que o formulário esteja sendo enviado para um recurso externo, o envio propriamente dito é bloqueado por conta dos validadores
- A seguir, alguns exemplos para ilustrar seu uso:



Validadores

- **required** e **maxlength**

```
<input id="nome" name="nome" type="text"  
      required="required" maxlength="20"/>
```

Nome:

 Preencha este campo.

Nome:



Treinamentos
IMPACTA

Validadores

- O campo do tipo **email** não possui nenhum validador em especial, mas, pela sua própria natureza, um campo inválido para o tipo é indicado. O mesmo vale para outros campos específicos

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

Email:

 A parte depois de "@" não deve conter o simbolo "@".



Treinamentos
IMPACTA

Validadores

- O atributo **pattern** permite que o valor do campo seja fornecido em conformidade com uma expressão regular. No exemplo a seguir, uma placa de carro deve ser fornecida com 3 dígitos alfabéticos e 4 numéricos. Se o valor fornecido não corresponder a este padrão, a validação falha:

```
<label for="placa">Placa do veículo:</label><br />
<input id="placa" name="placa" type="text" pattern="[a-zA-Z]{3}[0-9]{4}" />
```

Placa do veículo:

AB123

! É preciso que o formato
corresponda ao exigido.

Treinamentos
IMPACTA

Validadores

- O atributo **autofocus** permite que o componente receba o foco assim que a página for carregada
- O atributo **placeholder** já foi apresentado em exemplos anteriores. Ele permite que o controle de entrada contenha mais informações a respeito do conteúdo do campo. É um ótimo lugar para incluir o formato da placa do veículo, do exemplo anterior
- Elaborar a Atividade 3 – Inclusão de formulários

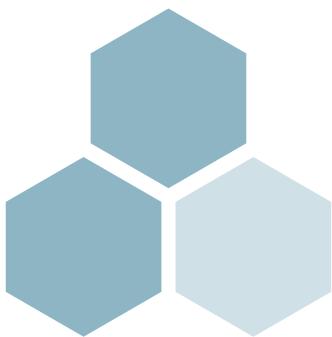
Treinamentos
IMPACTA



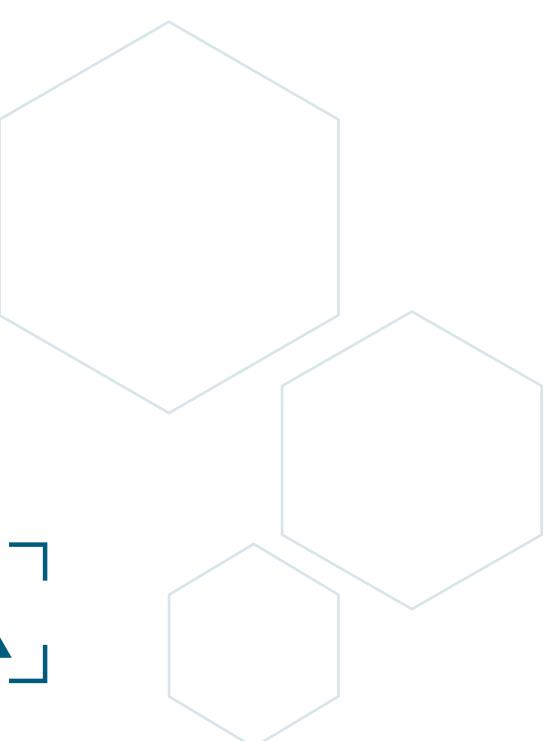
2

Estilos com CSS3

SCAMILLA
59.704.901-0007-67
LTD A



Editora
IMPACTA



Introdução

- **CSS** (Cascading Style Sheets), também chamado de **folhas de estilo**, é uma linguagem baseada em atributos e propriedades que define o layout de uma página
- O conteúdo CSS passa a funcionar a partir do momento em que o incluímos em nossa página. Essa inclusão pode ocorrer de três formas:
 - **inline**: Dentro das tags `<style>`, no próprio HTML

```
<style>
    div {background:yellow;border:1px;}
</style>
```

- **local**: No próprio elemento

```
<div style="background:yellow;border:1px;> ... </div>
```

Introdução

- **global**: Em um arquivo separado
- O uso de um arquivo separado é preferido, pois permite vasta reutilização
- Um arquivo CSS deve possuir a extensão `*.css` e deve ser incluído na página por meio do elemento `link`
- O exemplo ilustra a inclusão de um arquivo chamado **estilos.css**, localizado na mesma pasta do arquivo HTML em questão:

```
<head>
    <title></title>
    <meta charset="utf-8" />
    <link href="estilos.css" rel="stylesheet" />
</head>
```

Seletores

- Todas as regras baseadas em CSS possuem a mesma sintaxe geral:

```
seletor {  
    propriedade1:valor1;  
    propriedade2:valor2;  
    ...  
    propriedade_n:valor_n  
}
```

Existem três tipos de seletores:

- Seletor de elementos: `elemento {}`
- Seletor de classe: `.classe {}`
- Seletor de ID: `#id {}`



Seletores

- Exemplo 1:

```
body {  
    background-color: lightblue;  
}  
  
h1 {  
    color: white;  
    text-align: center;  
}  
  
p {  
    font-family: verdana;  
    font-size: 20px;  
}
```

```
<!DOCTYPE html>  
<html>  
<head>  
    <title></title>  
    <meta charset="utf-8" />  
    <link href="estilos.css" rel="stylesheet" />  
</head>  
<body>  
    <h1>Exemplo de CSS</h1>  
    <p>Primeiro parágrafo.</p>  
</body>  
</html>
```



Seletores

- Exemplo 1:



Seletores

- Exemplo 2:

```
* {  
    margin: 10px;  
}  
  
.borda {  
    background-color: rgb(255, 216,  
0);  
    padding: 20px;  
    border: solid 1px red;  
}  
  
#cabecalho {  
    background-color: rgb(0, 255,  
144);  
    font-family: Verdana;  
    font-size: 20px;  

```

```
<!DOCTYPE html>  
<html>  
<head>  
    <title></title>  
    <meta charset="utf-8" />  
    <link href="estilos.css" rel="stylesheet" />  
</head>  
<body>  
  
    <div id="cabecalho">  
        CURSO DE DESIGN RESPONSIVO  
    </div>  
  
    <p class="borda">  
        Neste curso, todos aprenderão a criar páginas  
        muito interessantes  
    </p>  
  
</body>  
</html>
```

Seletores

- Exemplo 2:



Seletores

- Os seletores podem ser combinados para criarmos regras mais específicas
- O símbolo * (asterisco) retorna a lista de todos os elementos
- Os colchetes [...] permitem o refinamento de seletores com base nos atributos

A seguir, listaremos algumas possíveis combinações de seletores:

Seletores

Combinação	Definição	Exemplo
<code>elem1 > elem2</code>	Seleciona o elem2 que for filho de elem1	<code>div > p {}</code>
<code>elem1 elem2</code>	Seleciona todos os elem2 que estiverem abaixo de elem1	<code>div p {}</code>
<code>elem1, elem2</code>	Seleciona os elementos elem1 e elem2 (e outros que estiverem separados por vírgula)	<code>h1, h2, #borda {}</code>
<code>elem:link</code>	Pseudoclasse para link. elem é selecionado se for um link e não tiver sido visitado ainda	<code>a:link {}</code>
<code>elem:visited</code>	Pseudoclasse para link. elem é selecionado se for um link já visitado	<code>a:visited {}</code>

Seletores

Combinação	Definição	Exemplo
<code>elem:active</code>	Pseudoclasse para link. elem é selecionado no momento em que está sendo clicado	<code>a:active {}</code>
<code>elem:hover</code>	Pseudoclasse para link. elem é selecionado no momento em que o mouse se move sobre ele	<code>a:hover {}</code>
<code>elem:focus</code>	Seleciona os elementos elem quando recebem o foco. Útil para elementos de formulário	<code>input:focus {}</code>
<code>elem[atributo]</code>	Seleciona qualquer elemento que possua o atributo especificado	<code>input[required]</code>
<code>elem[atributo="valor"]</code>	Seleciona qualquer elemento que possua o atributo com o valor especificado	<code>input[type="password"]</code>

Seletores

- Fontes e medidas

```
@font-face {
    font-family: minhaFonte;
    src: url(CandaraPlus.ttf);
}
```

```
.fonte {
    font-size: 16pt;
    line-height: 0.5in;
    letter-spacing: 12mm;
}
```

```
.fonte {
    font-size: 1em;
    border-width: 300px;
    padding: 16rem;
}
```

- Definição e acesso a fontes externas
- Medidas com tamanho absoluto
- Medidas com tamanho relativo



Seletores

- Fontes e medidas

```
.texto {
    word-spacing: 2rem;
}
```

```
.texto {
    text-shadow: 2px 2px 0 red;
}
```

- Espaço entre palavras do texto
- Efeitos de sombreamento



Gradientes

- Gradientes são especialmente úteis quando desejamos dar ênfase a uma página, ou a um plano de fundo
- Infelizmente, nem todos os navegadores suportam gradiente de maneira uniforme. Por isso, é necessário incluir, no CSS, opções de gradiente de acordo com o fabricante

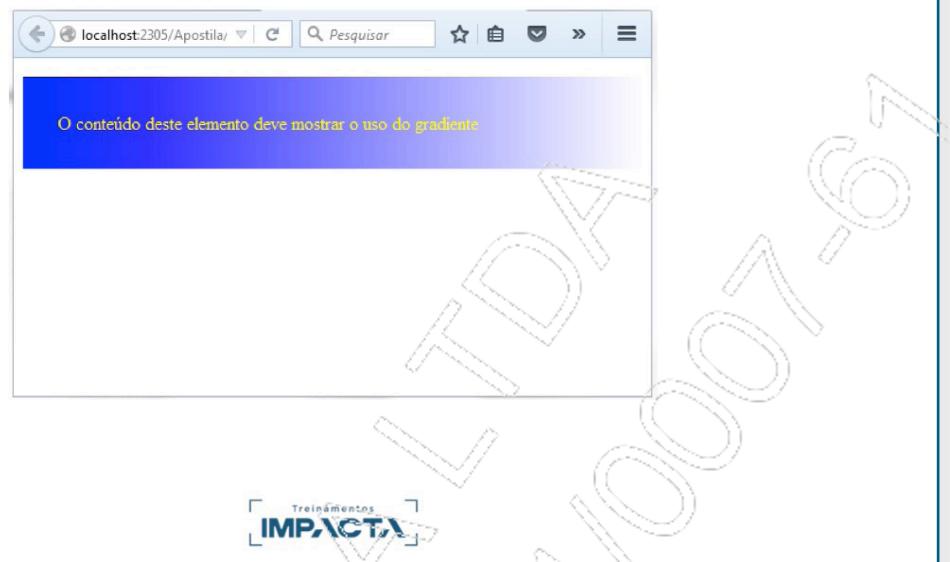
Vejamos o exemplo a seguir:

Gradientes

```
#grad {  
    /* Para navegadores que não suportam gradiente */  
    background: red;  
    /* Safari, versão 5.1 a 6.0 */  
    background: -webkit-linear-gradient(left, red , yellow);  
    /* Opera, versão 11.1 a 12.0 */  
    background: -o-linear-gradient(right, red, yellow);  
    /* Firefox, versão 3.6 a 15 */  
    background: -moz-linear-gradient(right, red, yellow);  
    /* Todas as demais condições */  
    background: linear-gradient(to right, blue , white);  
}
```

```
<body>  
    <div id="grad">  
        <p class="pgrad">  
            O conteúdo deste  
            elemento deve mostrar  
            o uso do gradiente  
        </p>  
    </div>  
</body>
```

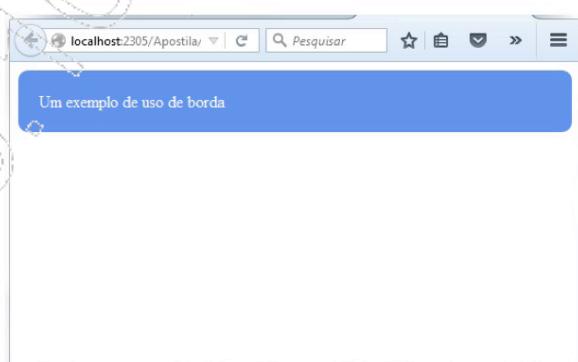
Gradientes



Bordas

- Definir bordas com CSS3 é bastante simples. Veja o exemplo:

```
.borda {  
    border-radius: 10px;  
}  
  
<div class="borda fundo">  
    Um exemplo de uso de borda  
</div>
```



Transições e animações

- Além de definirmos propriedades visuais para elementos, podemos, também, apresentar informações sob um destaque especial

Exemplo de rotação:

```
.rotacao {  
    transform: rotate(5deg);  
}  
  
.conteudo {  
    padding: 20px;  
    color: blue;  
    background-color: lightyellow;  
    margin: 20px;  
}
```

- Rotaciona o conteúdo em 5 graus

```
<body>  
    <article class="conteudo rotacao">  
        <strong>Sobre o ICS</strong>  
        <p>  
            É importante que ao término do curso o  
            aluno realize seu ICS  
        </p>  
    </article>  
</body>
```

Transições e animações

Sobre o ICS
É importante que ao término do curso o aluno realize seu ICS

Transições e animações

- Neste exemplo, a primeira letra de um texto aparece em destaque:

```
p::first-letter {  
    font-size: 50px;  
    color: blue;  
}
```

```
<body>  
    <p>  
        "O poeta é um fingidor.<br />  
        Finge tão completamente<br />  
        Que chega a fingir que é dor<br />  
        A dor que deveras sente."<br />  
    </p>  
</body>
```

"O poeta é um fingidor.
Finge tão completamente
Que chega a fingir que é dor
A dor que deveras sente."

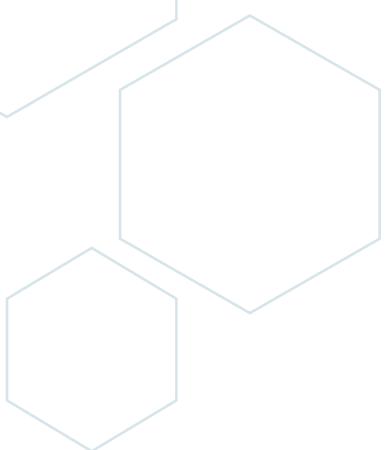
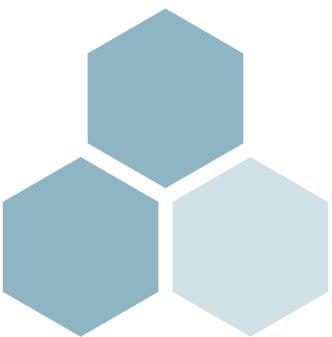
- Elaborar a Atividade 4 - Aplicação de estilos com CSS3



3

Responsividade com media queries

SCAM 59.704.
LTDA 007-67



Elemento @media

- Com o uso cada vez mais frequente de dispositivos móveis, tornou-se natural o usuário acessar seus sites preferidos quando estiver em movimento, ou seja, por meio principalmente do celular
- Com isso, nada mais natural que as aplicações Web se adaptarem a esse perfil, de forma mais econômica e personalizada
- O CSS3 incluiu um novo recurso conhecido como **Design Adaptativo**, ou **Design Responsivo**, que permite uma adaptação da aplicação de acordo com o dispositivo. Por exemplo, se uma aplicação Web possui um vídeo de apresentação, este poderia ser dispensável em um dispositivo móvel, uma vez que pode consumir recursos de conexão desnecessariamente

Elemento @media

- A ilustração a seguir apresenta a configuração de uma aplicação em diferentes dispositivos.



Fonte: <http://brolik.com/blog/responsive-web-design-menu-examples-css-jquery-tips/>

Elemento @media

- Para que essa tarefa seja viabilizada, algumas regras devem ser seguidas na elaboração da aplicação, principalmente no que diz respeito ao CSS



Viewport

- As dimensões de resolução da CSS são medidas de uma forma diferente da resolução física do aparelho. A isso chamamos de **css pixels**
- Para adequar o conteúdo a vários dispositivos, utilizamos como referência a tag **meta** com o **name="viewport"**, que determina a área visível do conteúdo em várias dimensões diferentes
- Para evitar que o usuário tenha que aplicar um zoom no dispositivo, aplicamos esta tag:

```
<head>
    <title>Design Responsivo</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
</head>
```



Viewport

- Podemos, também, transferir este controle para o CSS:

```
@viewport {  
    width: device-width;  
    zoom: 1;  
}
```

- Nestes exemplos, aplicando o valor de 1.5% no zoom, o dispositivo apresenta o conteúdo com 50% de zoom

@media

- Uma **media query** consiste em um tipo de mídia contendo uma ou mais expressões avaliadas como verdadeiras ou falsas
- Os **media types** são:

Valor	Descrição
all	Usada para todos os tipos de dispositivos
print	Aplicada a impressoras
screen	Aplicada a telas de computadores e outros dispositivos
speech	Usada para telas que obtêm dados da página por meio de sons (normalmente provenientes de fala)

Responsividade com media queries

@media

- Como exemplo, considere o CSS a seguir ([estilos.css](#)):

```
body{
    background-color: lightyellow;
}

.titulo {
    padding: 15px;
    background-color: #0094ff;
    color: white;
    font-family: Verdana;
    font-size: 16px;
}
```

```
@media screen and (max-width: 480px) {
    body {
        background-color: lightgreen;
    }

    .titulo {
        display: none;
    }
}
```



@media

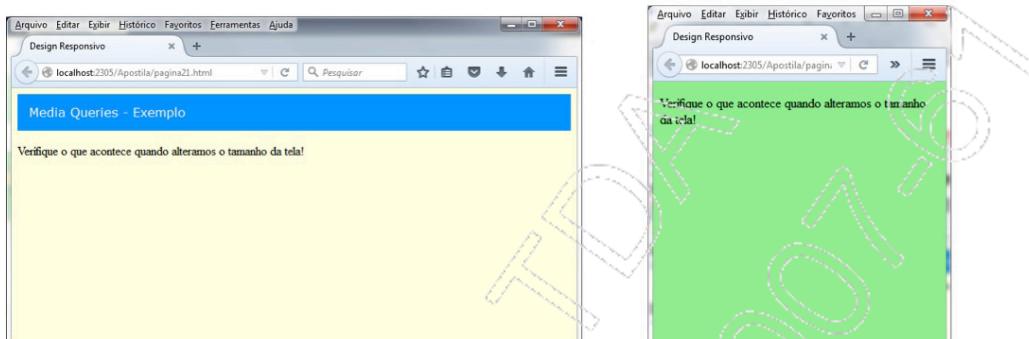
- Agora, a página HTML:

```
<!DOCTYPE html>
<html>
<head>
    <title>Design Responsivo</title>
    <meta charset="utf-8" />
    <link href="estilos.css" rel="stylesheet" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
</head>
<body>
    <div class="titulo">
        Media Queries - Exemplo
    </div>
    <p>
        Verifique o que acontece quando alteramos o tamanho da tela!
    </p>
</body>
</html>
```



@media

- O resultado para desktop e para uma mídia com menos de 480 pixels:



@media

Considere esta expressão:

```
@media screen and (max-width: 480px) { }
```

Nela, incluímos as configurações para os seletores, quando o tamanho da tela (screen) for, no máximo, 480px (max-width: 480px)

Responsividade com media queries

@media

- Para resoluções superiores a esse tamanho, valem as regras CSS estabelecidas no conteúdo do estilo
- Podemos definir vários blocos com **@media**, de acordo com a quantidade de páginas que tivermos em nossa aplicação, ou de elementos que desejarmos configurar para diferentes dispositivos

Mais exemplos e modelos podem ser encontrados em <http://www.w3schools.com/css/css_rwd_intro.asp>.

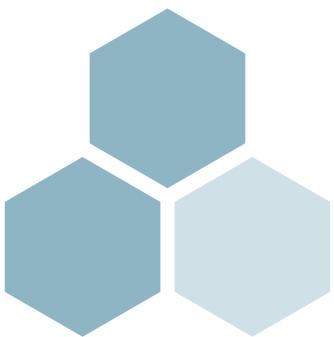
@media

- Elaborar a Atividade 5 – Desenvolvimento de responsividade com media queries



4

Conceitos do JavaScript e do Node.js



Editora
IMPACTA

SCAMAN
59.704.
SANTO DOMINGO
LTD A
0007-67



Introdução ao JavaScript

- JavaScript é uma linguagem interpretada cuja execução ocorre no browser, ou seja, no cliente
- O JavaScript deve ser usado com a finalidade de tornar as páginas dinâmicas



Introdução ao JavaScript

DOM (Document Object Model)

- O DOM é usado pelo navegador Web para representar seus elementos. É uma estrutura construída pelo navegador, em conformidade com o conteúdo da página e com a natureza dos componentes
- Quando alteramos a estrutura dos elementos via JavaScript, alteramos, também, o DOM, ou a forma como os elementos são mostrados



Fonte: <http://www.w3schools.com/jsref/>



Introdução ao JavaScript

Estrutura do JavaScript

- Instruções são finalizadas com ponto e vírgula

```
var variavel = 3;  
variavel = variavel + 1;  
document.write("Instruções longas \  
podem ser escritas em outra linha");
```

- Comentários

```
document.write("Introdução ao JavaScript"); //apresenta mensagem  
/*  
Este bloco pode ser usado para múltiplas  
anotações  
*/
```

Execução de instruções JavaScript

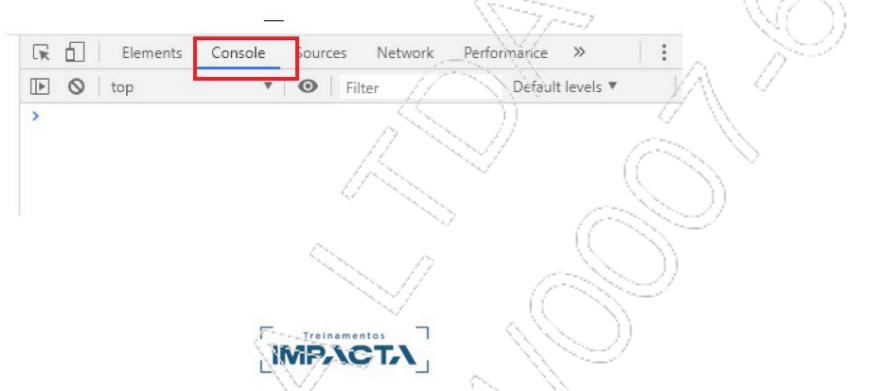
Para executarmos instruções JavaScript, temos diversos caminhos. Neste curso, apresentaremos três formas:

- Por meio do console do browser
- Por meio de páginas HTML
- Por meio do Node.js

Execução de instruções JavaScript

1 - Execução por meio do browser

- Tomando como base o Chrome, pressione F12 e selecione o item **Console**:



Execução de instruções JavaScript

1 - Execução por meio do browser

- Quando digitada, cada instrução é executada após o usuário pressionar ENTER:

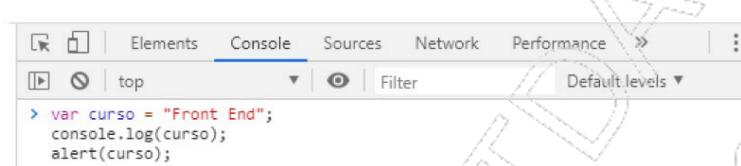
```
console.log("Executando no browser");
Executando no browser
VM120:1
```

A screenshot of the Google Chrome DevTools Console tab. A user has typed the command 'console.log("Executando no browser");' into the input field. The output shows the string 'Executando no browser' followed by the file path 'VM120:1'. The background features a watermark with the word 'IMPACTA' and some geometric shapes.

Execução de instruções JavaScript

1 - Execução por meio do browser

- Para digitar mais de uma instrução (mais de uma linha), o usuário deve manter a tecla SHIFT pressionada enquanto pressiona ENTER.



```
var curso = "Front End";
console.log(curso);
alert(curso);
```

Execução de instruções JavaScript

2 - Execução por meio de páginas HTML

- Nesta abordagem, o código é acionado por meio de uma página HTML. Na verdade, esta é a abordagem mais comum, uma vez que as instruções JavaScript são, na sua maioria, executadas para atender páginas HTML.
- Geralmente, um código em JavaScript é executado por meio de eventos (o clique de um botão, por exemplo), escrito diretamente no arquivo HTML ou em um arquivo separado. Seu acesso é realizado no elemento <script>



Execução de instruções JavaScript

No exemplo a seguir, uma mensagem é apresentada quando o botão é clicado:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Javascript</title>
</head>
<body>
    <button type="button" onclick="mostrar()">Clique aqui</button>

    <script>
        function mostrar(){
            alert("Javascript");
        }
    </script>
</body>
</html>
```

Execução de instruções JavaScript

3 - Execução por meio do Node.js

- Node.js é uma ferramenta baseada no JavaScript cujo propósito é o desenvolvimento de módulos. É possível desenvolver módulos representando componentes de servidor, componentes visuais, entre outros

Não é propósito deste curso desenvolver aplicações baseadas no Node.js. Nós o usaremos como ferramenta de execução de instruções JavaScript e para criar recursos necessários ao desenvolvimento do conteúdo.

Execução de instruções JavaScript

Para usar o Node.js, é necessário instalá-lo. Ele deve ser baixado por meio deste link:

<https://nodejs.org/en/>

Basta seguir os passos no processo de instalação.

Na instalação, a ferramenta **npm** (node package modules) também é incluída. Futuramente usaremos essa ferramenta para gerar um serviço REST de teste para nossas aplicações.



Execução de instruções JavaScript

Uma vez instalado e devidamente configurado, podemos executar instruções JavaScript escritas em arquivos ***.js** por meio desta sintaxe:

`node arquivo.js`

Em que **arquivo.js** é o arquivo contendo as instruções JavaScript desejadas.

Veja o exemplo:



Execução de instruções JavaScript

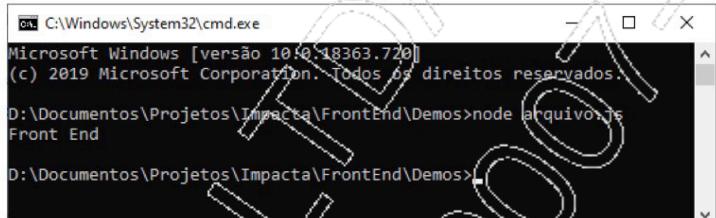
- **arquivo.js:**

```
var curso = "Front End";
console.log(curso);
```

- execução no prompt de comandos:

```
node arquivo.js
```

- resultado:



The screenshot shows a Windows command prompt window titled 'C:\Windows\System32\cmd.exe'. The window displays the following text:
Microsoft Windows [versão 10.0.18363.720]
(c) 2019 Microsoft Corporation. Todos os direitos reservados.
D:\Documentos\Projetos\Impacta\FrontEnd\Demos>node arquivo.js
Front End
D:\Documentos\Projetos\Impacta\FrontEnd\Demos>

Variáveis

- Usamos **var** ou **let** para declarar variáveis

```
var nome = "Sebastian";
var idade = 35;
let altura = 1.75;
```

Variáveis – Diferenças entre var e let

- Declaração com **var**: A variável pode ser declarada e/ou redeclarada, mesmo com valores de tipos diferentes. Existe a possibilidade de usarmos a variável antes mesmo que ela seja declarada

```
var texto = "Impacta";
console.log(texto);
var texto = 123;
console.log(texto);

console.log(numero);
var numero = 100;
```



Variáveis – Diferenças entre var e let

The screenshot shows a browser's developer tools console tab. The code is identical to the one in the previous block:

```
> var texto = "Impacta";
> console.log(texto);
> var texto = 123;
> console.log(texto);

> console.log(numero);
> var numero = 100;
```

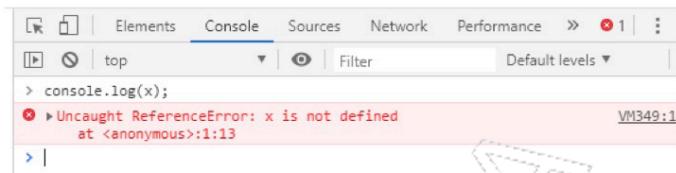
The output in the console is:

Output	Line Number
Impacta	VM275:2
123	VM275:4
undefined	VM275:6

Observe que, ao exibir a variável **numero**, tivemos como resposta **undefined**. O mesmo não ocorreria se tentássemos apresentar uma variável inexistente:



Variáveis – Diferenças entre var e let



```
console.log(x);
Uncaught ReferenceError: x is not defined
    at <anonymous>:1:13
```

Treinamentos
IMPACTA

Variáveis – Diferenças entre var e let

- Declaração com **let**: A variável pode ser declarada, mas não pode ser redeclarada. Não existe a possibilidade de usarmos a variável antes que ela seja declarada. Uma vez definida, devemos mantê-la até o final do seu ciclo de vida

```
let texto = "Impacta";
console.log(texto);
let texto = 123;
console.log(texto);

console.log(numero);
let numero = 100;
```

Treinamentos
IMPACTA

Variáveis – Diferenças entre var e let

```
Elements Console Sources Network Performance > 1 | ...  
top Filter Default levels  
> let texto = "Impacta";  
      console.log(texto);  
      let texto = 123;  
      console.log(texto);  
  
      console.log(numero);  
      let numero = 100;  
✖ Uncaught SyntaxError: Identifier 'texto' has already been declared VM357:3  
> |
```

JavaScript - Funções

- Funções são usadas para que um conjunto de instruções possa ser reaproveitado
- Parâmetros são acessíveis somente dentro das funções
- Funções podem retornar valores
- Variáveis locais podem ser declaradas dentro das funções
- Variáveis declaradas fora das funções possuem acesso global

```
function nomeFuncao(parametro1, parametro2) {  
    //instruções  
}
```



JavaScript – Comandos de decisão

- Comando **if**

```
if (nuloValor == null) {  
    document.write("Valor nulo");  
}
```

```
if (idade < 18) {  
    document.write("Menor de idade");  
}  
else {  
    document.write("Maior de idade");  
}
```



JavaScript – Comando **switch**

- Comando **switch**

```
var taxa;  
switch (tipoQuarto) {  
    case "Suite":  
        taxa = 500;  
        break;  
    case "King":  
        taxa = 400;  
        break;  
    default:  
        taxa = 300;  
}
```



JavaScript – Estruturas de repetição

- Estruturas de repetição: **while**, **do..while**, **for**

```
while (valor > 0) {
    saldo = saldo + valor;
    valor--;
}
```

```
do {
    if (valor <= 0) {
        break;
    }
} while (true);
```

```
for (var i = 0; i < 10; i++) {
    executar(i);
}
```



JavaScript - Objetos

Objetos implícitos do JavaScript:

- String**
- Date**
- Array**
- RegExp**

```
//array
var diasUteis = ["Segunda", "Terça", "Quarta", "Quinta", "Sexta"];

//Date
var hoje = new Date();

//RegExp
var reg = new RegExp("a.c");
if (reg.test("abc")) {
    //instrução
}
```



JavaScript - JSON

Objetos:

- O formato JSON (JavaScript Object Notation) é o mais adequado para representação de objetos:

```
//JSON  
var aluno = { "nome": "Carlos", "curso": "HTML5" };  
  
var alunos = [  
    { "nome": "Gerson", "curso": ".NET" },  
    { "nome": "Talita", "curso": "Java" },  
    { "nome": "Zé", "curso": "Manutenção" }  
];
```

- JavaScript possui APIs para serializar e realizar o parsing de objetos JSON:
 - **JSON.parse()**
 - **JSON.stringify()**

Integrando HTML com JavaScript

Vimos que a integração do JavaScript com HTML é a que faz mais sentido quando desejamos executar instruções, já que as páginas Web são as que de fato solicitam a execução dessas instruções. O JavaScript possui uma API de programação que permite manipular elementos da página Web, de diversas formas:

- Alterando elementos
- Executando eventos
- Modificando estilos, quando aplicável
- Validando e atualizando páginas

A seguir, apresentaremos exemplos de uso dos recursos mencionados



Integrando HTML com JavaScript

Considere este formulário:

```
<form name="LoginUsuario">
    <label for="uname">Nome do usuário:</label>
    <input id="uname" name="username" type="text" />
    <input type="submit" value="Enviar" />
</form>
```

Podemos referenciar o form:

```
document.forms[0];
document.forms["LoginUsuario"];
document.forms.LoginUsuario;
document.LoginUsuario;
```

Integrando HTML com JavaScript

Podemos, também, referenciar um elemento do form (elemento **username**):

```
document.forms.LoginUsuario.elements[0];
document.forms.LoginUsuario.elements["username"];
document.forms.LoginUsuario.username;
document.LoginUsuario.username;
document.getElementById("uname");
```

Integrando HTML com JavaScript

A manipulação de objetos no DOM consiste nos seguintes passos:

- Criar um novo objeto com o dado
- Buscar o elemento pai que deve conter o novo dado
- Manipular o elemento com o novo dado

Para remover um elemento ou atributo:

- Busque o elemento
- Use **removeChild** ou **removeAttribute**, conforme o caso



Integrando HTML com JavaScript

Exemplo de aplicação:

```
<body>
    <div id="conteudo">
        ...
    </div>
    <input type="button" value="Alterar conteúdo da div" onclick="inserir();"/>
    <script>
        function inserir() {
            var elemento = document.getElementById("conteudo");
            elemento.innerHTML = "TEXTO INSERIDO NO ELEMENTO";
        }
    </script>
</body>
```



Eventos em campos de formulário

- Nós tivemos a oportunidade de ver como uma função **JavaScript** pode ser chamada para ser executada pelo evento **click** de um botão
- Existem outras alternativas para executarmos eventos, sempre visando uma melhor estruturação do código
- Muitos elementos HTML possuem eventos embutidos que podem ser chamados conforme surge a necessidade. É o caso do evento **onclick** do elemento **button**, que vimos no exemplo anterior



Eventos em campos de formulário

- Para separar a chamada ao evento do formulário, podemos adicionar o evento via **JavaScript** e incluí-lo no **DOM**.
Veja o exemplo:

```
<body>
    <input type="button" id="btn" value="Apresentar mensagem" />

    <script>
        var botao = document.getElementById("btn");

        botao.addEventListener("click", function () {
            alert("O evento click foi acionado");
        });
    </script>
</body>
```





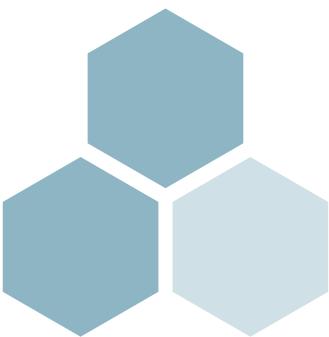
Eventos em campos de formulário

- No exemplo, usamos uma função anônima, pois sua funcionalidade é exclusiva para este evento
- É possível definir funções e associá-las a eventos, bastando indicar seu nome no lugar da definição de função
- Este procedimento é útil, especialmente se o evento associado a um controle depender de uma condição especial, além de poder ser usado para múltiplos eventos



5

JavaScript - Chamadas assíncronas e promises



Editora
IMPACTA

Chamadas assíncronas

- É comum que uma função, ao ser executada, receba como parâmetro outra função. A função passada como parâmetro é tecnicamente chamada de **callback**
- Tivemos a oportunidade de ver a função `addEventListener`, que recebe um callback como segundo parâmetro:

```
botao.addEventListener("click", function () {
    alert("O evento click foi acionado");
});
```



Chamadas assíncronas

- No exemplo, foi passada uma função anônima (sem nome) como segundo parâmetro. Esse tipo de tarefa é bastante comum em JavaScript por duas razões:
 1. A função callback é executada de forma **assíncrona**, ou seja, em segundo plano em relação à função principal. Quando seu resultado é obtido, a função principal é finalizada;
 2. Ela permite uma flexibilização na execução da função principal.



Chamadas assíncronas

- Com base no item 2, podemos destacar a própria função **addEventListener** mostrada como exemplo. Ao clicar no botão, não temos um comportamento único em todos os elementos. Cada elemento possui sua finalidade, e esta finalidade é descrita por meio da função callback



Chamadas assíncronas

- Uma função **callback** pode ser passada de três formas:
 1. Pelo nome da função;
 2. Como função anônima;
 3. Como arrow function.





Chamadas assíncronas

Para ilustrar, vamos considerar um botão e seu evento click, nas três formas:

1. Pelo nome da função:

```
function mostrar(){
    alert("Usando referência à função");
}

botao.addEventListener("click", mostrar);
```



Chamadas assíncronas

2. Como função anônima:

```
botao.addEventListener("click", function(){
    alert("Usando função anônima");
});
```

3. Como arrow function:

```
botao.addEventListener("click", () => alert("Usando função anônima"));
```



Promises

Os **promises** são definidos no JavaScript como forma de representar cadeias de operações assíncronas.

Vimos que uma função callback é executada assincronamente, mas, em muitos cenários, podemos ter mais de uma operação, sendo que a conclusão da segunda depende da finalização da primeira.

Para conseguir esse propósito, o JavaScript disponibiliza um objeto chamado **promise**. Na sua definição, ele recebe como parâmetro uma função que, por sua vez, recebe outras duas funções callback.



Promises

A forma geral de um promise é a seguinte:

```
const p = new Promise((resolve, reject) => {
  try {
    resolve(funcao(){})
  } catch (e) {
    reject(e)
  }
})
```

Nessa representação, temos:



Promises

- **resolve**: A função desejada
- **reject**: A função chamada em caso de erro

Vamos apresentar este conceito por meio de um exemplo:



Promises

```
let promise = new Promise((resolve, reject) => {
  let x = Math.random();
  if(x > 0.5){
    resolve('Valor válido: ' + x);
  } else {
    reject('Valor inválido - insuficiente: ' + x);
});
```

A variável **promise** no exemplo anterior representa o objeto a partir do qual a execução será executada. Sua chamada pode ser realizada da seguinte forma:



Promises

```
function mostrarResposta(s){  
    console.log('RESPOSTA: ');  
    console.log(s);  
}  
  
promise  
    .then(mostrarResposta)  
    .catch(console.error);
```

Observe que a execução da operação referenciada por **resolve** é chamada por meio de **then()**. O valor passado em **catch** é referenciado por **reject**.

Para executar, escreva o conteúdo em um arquivo chamado **promises.js**:



Promises

promises.js:

```
let promise = new Promise((resolve, reject) => {  
    let x = Math.random();  
    if(x > 0.5){  
        resolve('Valor válido: ' + x);  
    } else {  
        reject('Valor inválido - insuficiente: ' + x);  
    }  
});  
  
function mostrarResposta(s){  
    console.log('RESPOSTA: ');  
    console.log(s);  
}  
  
promise  
    .then(mostrarResposta)  
    .catch(console.error);
```



Promises

Executando com o Node.js, temos:

```
node promises.js
```

Execute mais de uma vez e observe o resultado.



Aplicação de promises – A função `fetch`

Uma das grandes forças do JavaScript é a capacidade de acessar serviços padrão **REST** (Representational State Transfer).

Trata-se de um padrão de Web service no qual o ponto de acesso é uma URL, semelhante a uma aplicação Web, mas que disponibiliza informações no formato JSON ou XML (sendo JSON o mais comum).

Além do consumo por meio de uma consulta, é possível, também, enviar informações para o Web service no sentido de adicionar ou atualizar informações no servidor remoto. Uma das funções disponíveis no JavaScript que possibilita essa tarefa é a função **fetch**.

Vamos apresentar sua utilidade por meio de um exemplo prático.

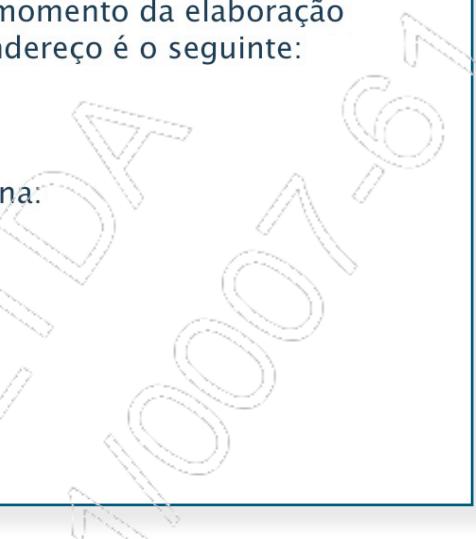


Aplicação de promises – A função `fetch`

No exemplo, apresentaremos um meio de obtermos os dados do endereço a partir do CEP informado. Até o momento da elaboração deste material, o link que disponibiliza o endereço é o seguinte:

<http://viacep.com.br/>

Acessando esse link, temos a seguinte página:



Aplicação de promises – A função `fetch`

ViaCEP - Webservice CEP e IBGE

Procurando um webservice gratuito e de alto desempenho para consultar Códigos de Endereçamento Postal (CEP) do Brasil? Utilize o serviço, melhore a qualidade de suas aplicações web e colabore para manter esta base de dados atualizada.

Acessando o webservice de CEP

Para acessar o webservice, um CEP no formato de {8} dígitos deve ser fornecido, por exemplo: "01001000". Após o CEP, deve ser fornecido o tipo de retorno desejado, que deve ser "json", "xml", "piped" ou "query".

Exemplo de pesquisa por CEP:
viacep.com.br/ws/01001000/json/

Validação do CEP

Quando consultado um CEP de formato inválido, por exemplo: "950100100" (9 dígitos), "95010A10" (alfanumérico),



Aplicação de promises – A função fetch

No tópico intitulado como **Acessando o webservice do CEP**, temos uma URL contendo o campo onde informamos o CEP com 8 dígitos. Vamos informar o CEP onde está localizada a Impacta:

<http://viacep.com.br/ws/01311100/json/>

Isso nos fornece o seguinte resultado:



Aplicação de promises – A função fetch

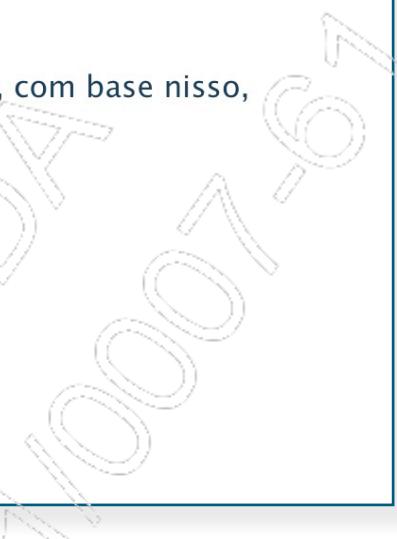
```
{  
  "cep": "01311-100",  
  "logradouro": "Avenida Paulista",  
  "complemento": "de 611 a 1045 - lado ímpar",  
  "bairro": "Bela Vista",  
  "localidade": "São Paulo",  
  "uf": "SP",  
  "unidade": "",  
  "ibge": "3550308",  
  "gia": "1004"  
}
```



Aplicação de promises – A função **fetch**

O nosso desafio é aplicar a função **fetch** para consumir este Web service em uma página HTML.

Em um campo de entrada, forneceremos o CEP e, com base nisso, apresentaremos as propriedades **rua** e **cidade**:



Aplicação de promises – A função **fetch**

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>
    <div>
        <input type="text" name="" id="cep" placeholder="Forneça o cep" /><br />
        <input type="text" name="" id="rua" placeholder="rua" /><br />
        <input type="text" name="" id="cidade" placeholder="cidade" /><br />
        <button id="mostrarButton">Mostrar</button>
    </div>
</body>
```



Aplicação de promises – A função fetch

```
<script>

    document.getElementById('mostrarButton')
        .addEventListener('click', function () {

            let cep = document.getElementById('cep').value;

            let url = `http://viacep.com.br/ws/${cep}/json/`;

            let resposta = {};

            fetch(url)
                .then(res => {
                    let x = res.json();
                    return x;
                })
                .then(valor => {
                    resposta = valor;
                    document.getElementById('rua').value = resposta.logradouro;
                    document.getElementById('cidade').value = resposta.localidade;
                });
        });
    </script>
</body>
</html>
```



Aplicação de promises – A função fetch

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



Aplicação de promises – A função **fetch**

No exemplo, a função **fetch** foi aplicada usando-se duas chamadas a **then()**. O processo ocorre da seguinte forma:

1. **fetch** recebe a URL a ser consumida como parâmetro;
2. Quando o resultado da execução do Web service estiver completado, seu conteúdo é repassado para a primeira função **then**. A resposta é repassada por meio do parâmetro **res**;
3. A função **then** recebe o valor referente à resposta e gera um objeto no formato JSON, representado pela variável **x**;



Aplicação de promises – A função **fetch**

4. O valor retornado pela primeira chamada a **then** (no nosso exemplo, **x**) é passado como parâmetro para a segunda chamada a **then**. Esse valor é recebido por meio do parâmetro **valor**;
5. O parâmetro **valor** contém o objeto obtido por meio do consumo do Web service.





Aplicação de promises – A função fetch

Dessa forma, a execução dessa página produz o seguinte resultado:

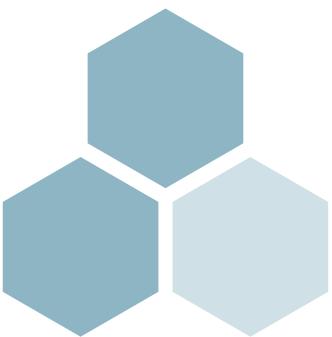
The image shows two side-by-side browser windows. Both have a title bar labeled 'Document'. The left window has three input fields: 'cep' containing '01311100', 'rua' (empty), and 'cidade' (empty). Below these is a button labeled 'Mostrar'. The right window shows the results of a search: 'cep' is still '01311100', 'rua' contains 'Avenida Paulista', 'cidade' contains 'São Paulo', and there is another 'Mostrar' button.



6

Conceitos do jQuery

SCAMIA 59. 704.907-0007-67



Conceitos do jQuery

- O jQuery é uma biblioteca JavaScript que fornece portabilidade de código JavaScript, além de permitir a construção de aplicações cross-browser
- Para trabalharmos com jQuery, primeiro temos que ter disponível a API (obtida de diversas formas, como pelo site jquery.com ou por meio do CDN)
- As versões mudam bastante de implementação para implementação
- Um exemplo é dado a seguir:

```
<head>
    <title></title>
        <meta charset="utf-8" />
        <script src="../Scripts/jquery-3.4.1.js"></script>
</head>
```

Conceitos do jQuery

Os principais recursos do jQuery incluem:

- Seleções de elementos HTML
- Manipulação de elementos HTML
- Manipulação CSS
- Eventos HTML
- Efeitos e animações JavaScript
- HTML DOM
- AJAX

Conceitos do jQuery

A sintaxe jQuery é orientada para selecionar elementos HTML e efetuar ações neles.

A sintaxe básica é **\$(seletorHTML).ação()**

Em que:

- O símbolo **\$** define o elemento;
- **(seletorHTML)** define onde fica o elemento; e
- **ação()** representa a tarefa a ser executada.

Exemplos:

```
$(this).hide()           // Esconde o elemento atual  
$("p").hide()           // Esconde todos os parágrafos  
$("p.teste").hide()      // Esconde todos os parágrafos com a classe="teste"  
$("#test").hide()         // Esconde o elemento com o id="teste"
```



Conceitos do jQuery

Os seletores permitem selecionar e manipular elementos HTML.

É possível selecionar por nome, atributo ou conteúdo.

Seletores de elementos

```
 $("p")                //seleciona os elementos <p>  
 $("p.intro")          //seleciona todos os elementos <p> com class="intro".  
 $("p#demo")           //seleciona o primeiro elemento <p> com id="demo".
```



Conceitos do jQuery

Seletores de atributos ou conteúdos

```
$("[href]")           // Seleciona todos os elementos com um atributo href  
$("[href='#']")      // Seleciona todos os elementos com um valor href igual a "#"  
$("[href!='#']")      // Seleciona todos os elementos com um valor href não igual a "#"  
$("[href$='.jpg']")   // Seleciona todos os elementos com um atributo href que termine com ".jpg"
```



Conceitos do jQuery

Os eventos jQuery são as funções que lidam com as ações no HTML.
Veja o exemplo a seguir:

```
<html>  
<head>  
    <title></title>  
    <meta charset="utf-8" />  
    <script src="../Scripts/jquery-3.4.1.js"></script>  
    <script type="text/javascript">  
        $(document).ready(function () {  
            $("button").click(function () {  
                $("p").hide();  
            });  
        });  
    </script>  
</head>
```



Conceitos do jQuery

```
<body>
  <h2>Isto é um título</h2>
  <p>Isto é um parágrafo.</p>
  <p>Isto é mais um parágrafo.</p>
  <button>Clica-me</button>
</body>
</html>
```

Quando o usuário clica no botão, todos os elementos **<p>** são escondidos.

Conceitos do jQuery

Mostrar e esconder elementos - **hide()**, **show()**:

```
$("#hide").click(function(){
  $("p").hide();
});

$("#show").click(function(){
  $("p").show();
});
```

Conceitos do jQuery

Ambos podem funcionar em conjunto com parâmetros opcionais - **speed** e **callback**:

```
$(selector).hide(speed,callback)  
$(selector).show(speed,callback)
```

O parâmetro **speed** especifica a velocidade de mostrar/esconder e pode ter os valores **slow**, **normal**, **fast** ou em milissegundos:

```
$("#botao").click(function(){  
    $("p").hide(800);  
});
```

Conceitos do jQuery

O parâmetro **callback** é o nome de uma função a ser executada depois que a função **hide/show** estiver completa.

Alternar: **toggle()**

O método **toggle()** permite alterar a visibilidade de elementos HTML que usam a função **show/hide**. Os elementos escondidos são mostrados, e os elementos visíveis são escondidos:

```
$(selector).toggle(speed,callback)
```

```
$("#botao").click(function(){  
    $("p").toggle(850);  
});
```

Conceitos do jQuery

Deslizar: **slideDown()**, **slideUp()**, **slideToggle()**

Os métodos de deslizamento do jQuery alteram gradualmente a altura dos elementos selecionados, por meio dos seguintes métodos:

```
$(selector).slideDown(speed,callback)
$(selector).slideUp(speed,callback)
$(selector).slideToggle(speed,callback)
```



Conceitos do jQuery

Exemplos:

```
//slideDown()
$("flip").click(function(){
    $(".panel").slideDown();
});

//slideUp()
$("flip").click(function(){
    $(".panel").slideUp();
});

//slideToggle()
$("flip").click(function(){
    $(".panel").slideToggle();
});
```



Conceitos do jQuery

Desvanecer: **fadeIn()**, **fadeOut()**, **fadeTo()**

Os métodos de desvanecer alteram gradualmente a opacidade dos elementos selecionados. jQuery tem os seguintes métodos de desvanecimento:

```
$(selector).fadeIn(speed,callback)
$(selector).fadeOut(speed,callback)
$(selector).fadeTo(speed,opacity,callback)
```

Conceitos do jQuery

Exemplos:

```
//fadeIn()
$("botão").click(function(){
    $("div").fadeIn(2000);
});

//fadeOut()
$("botão").click(function(){
    $("div").fadeIn(2000);
});

//fadeTo()
$("botão").click(function(){
    $("div").fadeTo("slow",0.30);
});
```

Conceitos do jQuery

- As animações são introduzidas por meio do seguinte código:

```
$(selector).animate({parametros},[duracao],[easing],[callback])
```

- O parâmetro chave é **parametros**, no qual serão introduzidas propriedades CSS que vão ser animadas. Podem ser animadas várias propriedades ao mesmo tempo:

```
animate({width:"60%",opacity:0.3,marginTop:"0.3in",fontSize:"2px"})
```

Conceitos do jQuery

- O segundo parâmetro é a duração, que define o tempo da animação. Aceita valores **slow**, **fast**, **normal** e em milissegundos.

Vejamos um exemplo:

```
<script type="text/javascript">
$(document).ready(function(){
    $("#botao").click(function(){
        $("div").animate({left:'29px'},"slow");
        $("div").animate({fontSize:'4px'},"slow");
    });
});
</script>
```

Conceitos do jQuery

- Utilizando a sintaxe **\$(selector).append(conteudo)**, pode-se anexar informação aos elementos selecionados
- Utilizando a sintaxe **\$(selector).prepend(conteudo)**, pode-se retirar uma informação dos elementos selecionados

```
$(“p”).append(“Curso HTML”);  
$(“p”).prepend(“Curso HTML”);
```



Conceitos do jQuery

Retorno de propriedade CSS

- Utilize o método **css(nome)** para retornar uma propriedade CSS escolhida, que será extraída do primeiro elemento encontrado que tenha a propriedade:

```
$(this).css(“background-color”);
```



Conceitos do jQuery

Definição de propriedade e valor CSS

- Utilize **css(nome,valor)** para definir os valores de uma propriedade CSS para todos os elementos que combinam com o introduzido:

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



Conceitos do jQuery

Definição de múltiplas propriedades e valores CSS

Utilize **css({propriedades})** para definir uma ou mais propriedades/valores para os elementos selecionados:

```
$("p").css({"background-color":"yellow","font-size":"200%"});
```

Outros exemplos:

```
$("#div1").height("200px"); // altura  
$("#div1").width("200px"); // largura
```



Iterações com jQuery - `$.each`

O jQuery disponibiliza uma função bastante interessante para realizarmos iterações sobre coleções: `$.each`.

A forma geral desta função é:

```
$.each(colecao, function(indice, elemento){  
    //instruções  
});
```



Iterações com jQuery - `$.each`

O primeiro parâmetro (**coleção**) é a lista que desejamos percorrer.

O segundo parâmetro é uma função **callback** contendo o índice (posição) do elemento da coleção e o elemento propriamente dito.

Esta função é chamada uma vez para cada elemento.

Vamos ilustrar sua utilização por meio de um exemplo:



Iterações com jQuery - `$.each`

```
let cursos = ['Java', 'HTML5', 'PHP', 'Angular'];
$.each(cursos, function(indice, item){
    console.log(indice + ' - ' + item);
});
```

- Elaborar a Atividade 6 – Inclusão de funcionalidades JavaScript e jQuery
- Elaborar a Atividade 7 – Consumo de serviços REST

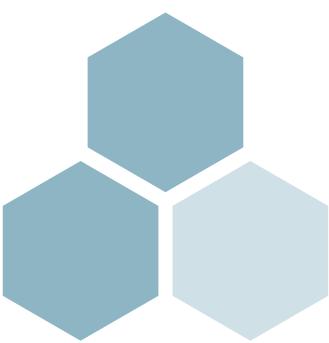




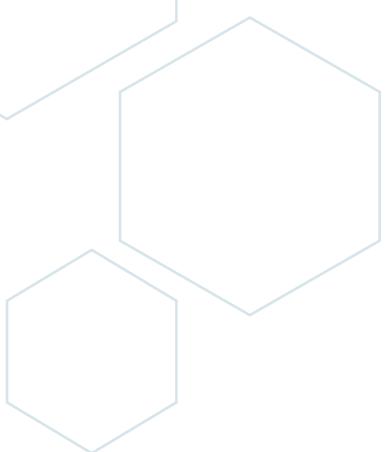
7

Otimizações de layout com Bootstrap

SCAMPIA
59.704.
LTD A
007-67



Editora
IMPACTA



Usando Bootstrap

- Bootstrap é um framework baseado em CSS e JavaScript, elaborado para criar páginas responsivas
- A ideia do Bootstrap é poupar tempo do desenvolvedor com grandes volumes de CSS. Por causa disso, é uma excelente alternativa para quem deseja criar páginas rapidamente, sem o esforço adicional de elaborar códigos em CSS ou JavaScript, que frequentemente podem ser usados apenas uma vez
- O princípio básico do Bootstrap é padronizar o nome das classes usadas para criar o layout de uma aplicação Web



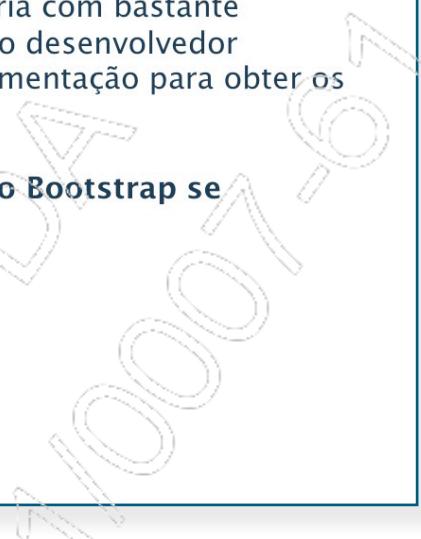
Usando Bootstrap

- A API do Bootstrap pode ser obtida em <<http://getbootstrap.com/>>
- Alternativamente, é possível usar a versão disponível no CDN (Content Delivery Network). Trata-se de uma versão on-line dos arquivos. Como vantagem, podemos destacar a não necessidade de organizar os arquivos no projeto, mas, como desvantagem, destacamos a necessidade de permanecermos on-line
- No portal <http://getbootstrap.com/>, estão apresentados os links dos recursos on-line
- A estrutura dos arquivos do Bootstrap é apresentada a seguir:



Usando Bootstrap

- Vale lembrar que a versão do Bootstrap, assim como de qualquer outra tecnologia que venhamos a utilizar, varia com bastante frequência, e é altamente recomendado que o desenvolvedor mantenha um constante contato com a documentação para obter os recursos mais recentes, se for necessário
- **Por ocasião da elaboração deste material, o Bootstrap se encontra na versão 4.4.1**



Usando Bootstrap – Exemplo

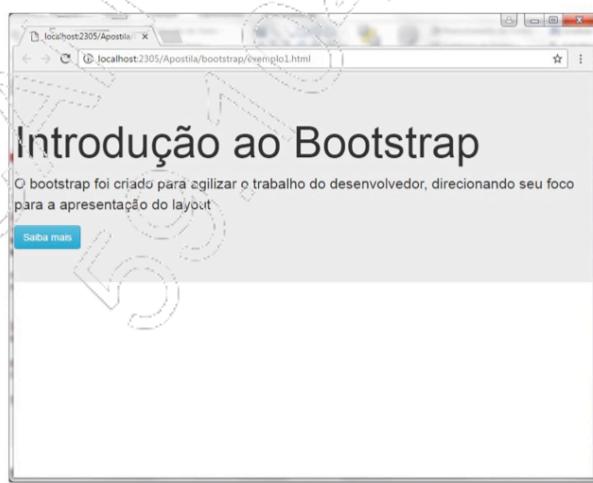
- A implementação do Bootstrap em nossas aplicações segue o mesmo princípio da aplicação do CSS e do JavaScript tradicionais. O desafio é entender a aplicabilidade das classes e dos elementos corretamente
- No exemplo a seguir, apresentaremos o princípio de aplicação do Bootstrap:



Usando Bootstrap – Exemplo

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <link href="bootstrap.min.css" rel="stylesheet" />
</head>
<body>
    <div class="jumbotron">
        <h1>Introdução ao Bootstrap</h1>
        <p>
            O bootstrap foi criado para agilizar o trabalho do desenvolvedor,
            direcionando seu foco para a apresentação do layout
        </p>
        <button type="button" class="btn btn-info">Saiba mais</button>
        <script src="Scripts/jquery-3.1.1.min.js"></script>
        <script src="Scripts/bootstrap.min.js"></script>
    </div>
</body>
</html>
```

Usando Bootstrap – Exemplo



Usando Bootstrap – Exemplo

- Observe a utilização das APIs:

```
<link href="bootstrap.min.css" rel="stylesheet" />  
<script src="Scripts/jquery-3.4.1.min.js"></script>  
<script src="Scripts/bootstrap.min.js"></script>
```

- Nem todas serão necessárias, especialmente para um exemplo básico como esse. O arquivo **bootstrap.min.css** seria suficiente
- Porém existem situações em que os elementos são produzidos com base em algum script, como é o caso de menus de navegação ou slides
- Como o jQuery é uma dependência, sua declaração deve vir antes de **bootstrap.min.js**



Usando Bootstrap – Exemplo

- Existem diversos componentes disponíveis para usarmos com o Bootstrap. No geral, todos os elementos HTML possuem classes especiais quando usadas com Bootstrap. A documentação oficial apresenta diversos exemplos e está disponível em <http://getbootstrap.com/getting-started/>
- Dentre os conceitos envolvidos, existe um de especial importância: a divisão em colunas
- No Bootstrap, uma linha é dividida em 12 colunas, ajustáveis de acordo com o dispositivo que o executar. Esse número foi escolhido por ser divisível por 2, 3, 4 e 6



Usando Bootstrap – Exemplo

- O importante é que o total de colunas seja 12, para que ocupe toda a largura da página
- Por exemplo, se desejarmos uma página com 2 colunas, a largura de cada coluna será 6
- Um exemplo será conveniente para apresentar esse processo.

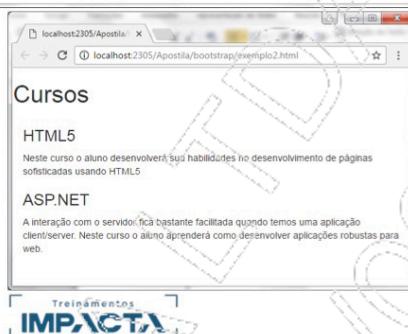
Usando Bootstrap – Exemplo

```
<h1>Cursos</h1>
<div class="container">
  <div class="row">
    <div class="col-md-6">
      <h3>HTML5</h3>
      <p> Neste curso o aluno desenvolverá sua habilidades no desenvolvimento de páginas sofisticadas usando HTML5
      </p>
    </div>
    <div class="col-md-6">
      <h3>ASP.NET</h3>
      <p> A interação com o servidor fica bastante facilitada quando temos uma aplicação client/server. Neste curso o aluno aprenderá como desenvolver aplicações robustas para web.
      </p>
    </div>
  </div>
</div>
```

Usando Bootstrap – Exemplo



- Em um dispositivo menor:



Usando Bootstrap – Exemplo

- Observe que usamos esta classe:

```
<div class="col-md-6">
  ...
</div>
```

- Existem classes para diferentes tamanhos de dispositivos. O prefixo de colunas serve para indicar em quais tipos de telas a coluna vai se manter posicionada, como no design principal
- Os prefixos têm o seguinte padrão:

Celulares	Tablets	Desktops	Telas grandes
.col-xs-*	.col-sm-*	.col-md-*	.col-lg-*



Bootstrap – Formulários e menus

- Neste tópico, analisaremos a aplicação do Bootstrap para formulários e menus de navegação, iniciando pelos formulários
- O conceito de colunas estudado no tópico anterior será importante na definição dos formulários
- O exemplo a seguir ilustra sua utilização:

Bootstrap – Formulários e menus

```
<div class="container">
    <h1>Cadastro de alunos</h1>
    <div class="form-horizontal">
        <hr />
        <div class="form-group">
            <div class="col-md-1">Nome:</div>
            <div class="col-md-11">
                <input type="text" name="nome" />
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-1">Curso:</div>
            <div class="col-md-11">
                <input type="text" name="curso" />
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-offset-1 col-md-11">
                <input type="submit" value="Incluir imagem" class="btn btn-default" />
            </div>
        </div>
    </div>
</div>
```

Bootstrap – Formulários e menus

- Execução



Bootstrap – Formulários e menus

- É interessante analisar o código-fonte da página quanto estiver da dimensão normal e reduzida, analisando as implicações da responsividade
- Apresentaremos, agora, um exemplo contendo um menu de navegação

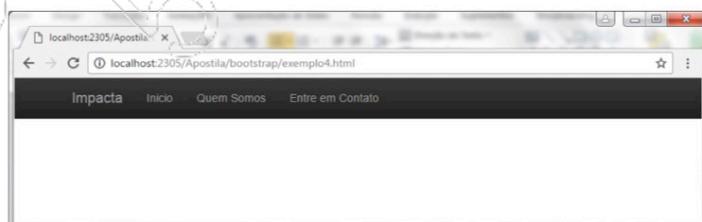


Bootstrap – Formulários e menus

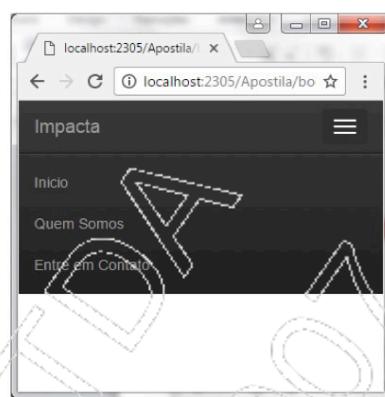
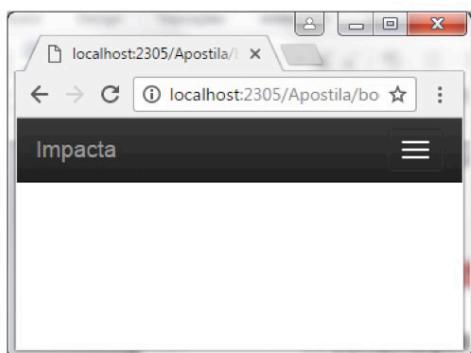
```
<div class="navbar navbar-inverse navbar-fixed-top">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle"
              data-toggle="collapse" data-target=".navbar-collapse">
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="#">Impacta</a>
    </div>
    <div class="navbar-collapse collapse">
      <ul class="nav navbar-nav">
        <li><a href="#">Inicio</a></li>
        <li><a href="#">Quem Somos</a></li>
        <li><a href="#">Entre em Contato</a></li>
      </ul>
    </div>
  </div>
</div>
```

Bootstrap – Formulários e menus

- A combinação de classes CSS é grande. Vale a pena avaliar o código do Bootstrap para conhecer os elementos
- Os elementos **data-toggle** e **data-target** não são nativos do elemento **button**; eles são gerados via JavaScript
- O resultado desta execução é apresentado a seguir:



Bootstrap – Formulários e menus



Bootstrap – Personalizando

- Pode parecer um pouco estranho sentir a necessidade de personalizar uma API que já foi desenvolvida para ser usada sem personalizações
- Acontece que, muitas vezes, configurações como cor de fundo de um controle, da fonte, largura, ou de outros itens podem não nos atender da forma como desejamos, ou fogem aos padrões de layout estabelecidos pela empresa ou por um cliente
- O que jamais deve ser feito é alterar o código do Bootstrap, pois isso o descaracteriza

Bootstrap – Personalizando

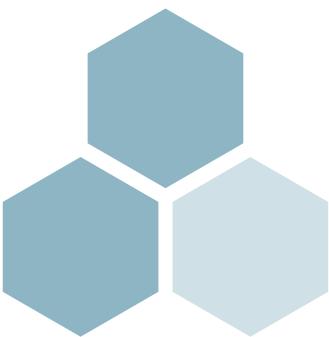
- Quando for necessário alterar os valores padrão para classes Bootstrap, faça o seguinte:
 - Localize as classes do elemento (é possível que haja classes combinadas entre si em diversos pontos do arquivo **bootstrap.css**)
 - Defina um novo arquivo, incluindo apenas esses elementos, com os novos valores
 - Inclua-o na página, após o arquivo **bootstrap.css** (ou **bootstrap.min.css**)
 - Elaborar a Atividade 8 – Otimizações de layout com Bootstrap



8

Animações em telas de fundo

SCAMIA 59.704.901-0007-67





Background animado

Com CSS3 é possível criar animações com a maioria dos elementos HTML. Uma observação aqui é que ele depende de um complemento presente no browser, normalmente, o Shockwave Player.

Exemplo:

```
@keyframes animacao {  
    from {background-color: red;}  
    to {background-color: yellow;}  
}  
  
div.anima {  
    width: 100px;  
    height: 100px;  
    background-color: red;  
    color: white;  
    font-size: 16px;  
    animation-name: animacao;  
    animation-duration: 4s;  
}
```

```
<body>  
    <div class="anima">  
        HTML5 + CSS3  
    </div>  
</body>
```



SVG

- SVG (Scalable Vector Graphics) é uma ferramenta presente no HTML5 que permite criar gráficos de forma bastante simples
- O componente SVG, representado pelo elemento **<svg>**, possui diversos métodos para elaboração de desenhos
- A fonte abaixo contém diversas aplicações do SVG:

Fonte: http://www.w3schools.com/html/html5_svg.asp



SVG

- Exemplo 1 – Desenhando um círculo:

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemplo SVG</title>
    <meta charset="utf-8" />
</head>
<body>
    <h2>Círculo com SVG</h2>
    <svg height="200" xmlns="http://www.w3.org/2000/svg">
        <circle cx="50" cy="50" r="50" fill="red" />
    </svg>
</body>
</html>
```

Círculo com SVG



SVG

- Exemplo 2 – Desenhando um polígono:

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemplo SVG</title>
    <meta charset="utf-8" />
</head>
<body>
    <h2>Polígono com SVG</h2>
    <svg height="200" xmlns="http://www.w3.org/2000/svg">
        <polygon points="20,10 300,20, 170,50" fill="red" />
    </svg>
</body>
</html>
```

Polígono com SVG



SVG

- Exemplo 3 – Desenhando um polyline:

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemplo SVG</title>
    <meta charset="utf-8" />
</head>
<body>
    <h2>Polyline com SVG</h2>
    <svg height="200" xmlns="http://www.w3.org/2000/svg">
        <polyline points="0,0 0,20 20,20 20,40 40,40 40,60" fill="red" />
    </svg>
</body>
</html>
```

Polyline com SVG

SVG e MathML

- Assim como o SVG, o HTML5 inclui uma ferramenta muito poderosa para escrevermos fórmulas matemáticas, chamada **MathML**
- Você pode encontrar mais detalhes no site <https://www.w3.org/Math/>

Vejamos um exemplo de utilização:

SVG e MathML

```
<html>
<head>
    <meta charset="UTF-8">
    <title>Teorema de Pitágoras</title>
</head>
<body>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
        <mrow>
            <msup><mi>a</mi><mn>2</mn></msup>
            <mo>+</mo>
            <msup><mi>b</mi><mn>2</mn></msup>
            <mo>=</mo>
            <msup><mi>c</mi><mn>2</mn></msup>
        </mrow>
    </math>
</body>
</html>
```

localhost:2305/Apo
 $a^2 + b^2 = c^2$



Canvas API

- Assim como o SVG, o Canvas API fornece um meio poderoso de desenhar formas (como gráficos) usando JavaScript
- Sendo assim, um gráfico pode ser gerado em função de dados externos, obtidos assincronamente via jQuery, por exemplo
- Para trabalhar com esta API, usamos o elemento <canvas>



Canvas API

- Como primeiro exemplo, apresentaremos um meio de desenhar um retângulo:

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <style>
        #ex_canvas {
            border: 1px solid blue;
        }
    </style>
</head>
<body>
    <canvas id="ex_canvas" width="100" height="100"></canvas>
</body>
</html>
```



Canvas API

- O elemento `<canvas>` inicialmente é apresentado em branco
- Para que algum conteúdo seja produzido sobre ele, é necessário criar seu contexto e, em seguida, usá-lo para elaborar o desenho
- O elemento `<canvas>` possui um método baseado em DOM chamado `getContext()`. Esse método recebe um parâmetro de contexto chamado **2D**
- Uma observação: o Internet Explorer 8 não suporta o Canvas nativamente. Nesse caso, é importante incluir esta instrução na página:

```
<!--[if IE]><script src="excanvas.js"></script><![endif]-->
```

Em que `excanvas.js` é um arquivo a ser incluído no seu projeto.



Canvas API

Vamos, agora, apresentar mais um exemplo de utilização do elemento <canvas>:

```
<html>
<head>
    <style>
        #figura {
            width: 100px;
            height: 100px;
            margin: 0px auto;
        }
    </style>

    <script type="text/javascript">
        function drawShape(){
            // obtém o elemento canvas
            var canvas = document.getElementById('elemento');

            // verifica se o canvas é suportado
            if (canvas.getContext){
                // chama getContext()
                var ctx = canvas.getContext('2d');
            }
        }
    </script>

```



Canvas API

```
// Desenha imagens
ctx.beginPath();
ctx.arc(75,75,50,0,Math.PI*2,true); // Circulo externo

ctx.moveTo(110,75);
ctx.arc(75,75,35,0,Math.PI,false); // Boca

ctx.moveTo(65,65);
ctx.arc(60,65,5,0,Math.PI*2,true); // Olho esquerdo

ctx.moveTo(95,65);
ctx.arc(90,65,5,0,Math.PI*2,true); // Olho direito

ctx.stroke();
```



Canvas API

```
        }
        else {
            alert('Seu browser não suporta canvas');
        }
    }
</script>
</head>

<body id="figura" onload="drawShape();">
    <canvas id="elemento"></canvas>
</body>

</html>
```



Canvas API

- Aqui está o resultado:



- Elaborar a Atividade 9 – Inclusão e listagem de registros com Web services
- Elaborar a Atividade 10 – Aplicação de Canvas API





9

Elementos de áudio e vídeo

SCAMILLA
59.704.
S.C.A.M.I.L.L.A.



Editora
IMPACTA

Áudio

- Para inserir um áudio na sua página, utilize o elemento **audio**:

```
<audio controls="controls">  
    <source src="mymusic.mp3" type="audio/mpeg" />  
</audio>
```

- Na página, você visualiza e pode reproduzir ou baixar a música:



Áudio

Existem outros atributos importantes para o áudio:

- autoplay**: Inicia a reprodução assim que a página é carregada;
- controls**: Apresenta os controles padrão;
- loop**: Ao terminar a música, inicia novamente.

```
<audio controls autoplay loop src="mymusic.mp3" />
```

Fonte: https://developer.mozilla.org/pt-BR/docs/Web/HTML/Using_HTML5_audio_and_video

Áudio

Se houver necessidade de um controle mais refinado sobre o **audio**, será necessário usar JavaScript em conjunto com o elemento:

```
<audio id="audio" controls>
  <source src="mymusic.ogg" type="audio/ogg">
  <source src="mymusic.mp3" type="audio/mpeg">
  Seu navegador não possui suporte ao elemento audio
</audio>

<div>
  <a href="#" onclick="play()">Play</a>
  <a href="#" onclick="pause()">Pause</a>
  <a href="#" onclick="stop()">Stop</a>
</div>

<div>
  <a href="#" onclick="aumentar_volume()> + volume</a>
  <a href="#" onclick="diminuir_volume()>- volume</a>
  <a href="#" onclick="mute()"> Mute</a>
</div>
```



Áudio

```
<script>
  audio = document.getElementById('audio');

  function play(){ audio.play(); }

  function pause(){ audio.pause(); }

  function stop(){ audio.pause(); audio.currentTime = 0; }

  function aumentar_volume() { if (audio.volume < 1) audio.volume += 0.1; }

  function diminuir_volume(){ if( audio.volume > 0) audio.volume -= 0.1; }

  function mute(){
    if( audio.muted ){
      audio.muted = false;
    }else{
      audio.muted = true;
    }
  }
</script>
```



Vídeo

Para inserir um vídeo, o processo é semelhante ao áudio.
Veja o exemplo:

```
<video controls="controls" width="400" height="300">
  <source src="Clip_480_5sec_6mbps_h264.mp4" type="video/mp4" />
  <p>
    Seu navegador não possui suporte ao elemento video
  </p>
</video>
```

Na sua página:



Vídeo



Você pode obter mais informações neste link:

Fonte: https://developer.mozilla.org/pt-BR/docs/Web/HTML/Using_HTML5_audio_and_video

- Elaborar a Atividade 11 – Uso de áudio e vídeo

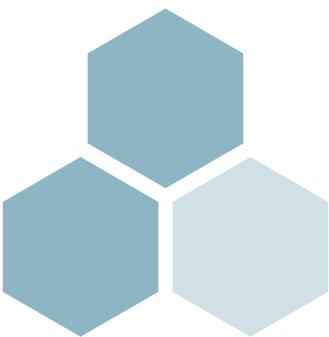




10

Comunicação com serviços REST

SCAMPIA
59.704.900-0007-67
LTD A



Editora
IMPACTA



Introdução

- Uma página Web, quando solicitada, realiza um processo de requisição por meio do método **HTTP GET**. Esse processo ocorre sempre que requisitamos a página, seja pela URL ou por meio de um link
- Para que a resposta seja gerada, o browser realiza a renderização para uma estrutura DOM
- Códigos JavaScript podem disparar requisições **GET**, de forma síncrona ou assíncrona. Por padrão, requisições ao servidor via JavaScript são assíncronas



Enviando e recebendo dados com JavaScript

- Para enviar uma requisição HTTP com JavaScript:
 - Crie um objeto **XMLHttpRequest**
 - Especifique o método e a URL a ser requisitada
 - Envie a requisição
- No exemplo a seguir, mostraremos esses passos:



Enviando e recebendo dados com JavaScript

- Na página, temos dois elementos: um div e um botão.

Observe o **id** para o **div** e a função JavaScript executada pelo evento **click** do botão:

```
<body>
    <div id="conteudo"></div>
    <input type="button" value="Executar Ajax" onclick="executarAjax();"/>
</body>
```



Enviando e recebendo dados com JavaScript

- Na área de script, definimos a função **executarAjax()**:

```
function executarAjax() {
    var xmlhttp = getXmlHttpRequest();

    xmlhttp.open("GET", "pagina.html", true);
    xmlhttp.onreadystatechange = function () {
        if (xmlhttp.readyState == 4) { //requisição completa
            if (xmlhttp.status == 200) { //resposta OK
                //processa resposta
                document.getElementById("conteudo").innerHTML = xmlhttp.responseText;
            } else {
                alert("Problema ao recuperar resposta");
            }
        }
    }
    xmlhttp.send();
}
```



Enviando e recebendo dados com JavaScript

- Definição da função `getXmlHttpRequest()`:

```
function executarAjax() {  
    var xmlhttp = getXmlHttpRequest();  
  
    xmlhttp.open("GET", "pagina.html", true);  
    xmlhttp.onreadystatechange = function () {  
        if (xmlhttp.readyState == 4) { //requisição completa  
            if (xmlhttp.status == 200) { //resposta OK  
                //processa resposta  
                document.getElementById("conteudo").innerHTML = xmlhttp.responseText;  
            }  
            else {  
                alert("Problema ao recuperar resposta");  
            }  
        }  
        xmlhttp.send();  
    }  
}
```

Enviando e recebendo dados com JavaScript

O processo todo funciona assim:

- Definimos um objeto `XMLHttpRequest`, verificando se o browser possui essa capacidade:

```
var xmlhttp = getXmlHttpRequest();
```

- Criamos a chamada à `pagina.html`:

```
xmlhttp.open("GET", "pagina.html", true);
```

Enviando e recebendo dados com JavaScript

- Definimos uma função **callback**, a ser executada assincronamente quando o envio da requisição for finalizado e a resposta pronta para ser exibida:

```
xmlhttp.onreadystatechange = function () {
    if (xmlhttp.readyState == 4) {
        if (xmlhttp.status == 200) {
            document.getElementById("conteudo").
                innerHTML = xmlhttp.responseText;
        }
        else { alert("Problema ao recuperar resposta"); }
    }
}
```

- Executamos o envio:

```
xmlhttp.send();
```

Enviando e recebendo dados com jQuery

- A biblioteca jQuery fornece métodos assíncronos para envio de requisições e recebimento de respostas. Seu uso é semelhante ao de **XMLHttpRequest**, um pouco mais simples. Exemplo:

```
var resposta;
$.get("pagina1.html", function (dados) {
    resposta = dados;
    document.getElementById("conteudo").innerHTML = resposta;
}).error(function () {
    alert("Problema ao recuperar resposta");
});
```



Enviando e recebendo dados com jQuery

- O exemplo a seguir recebe os dados de um arquivo cujo conteúdo está no formato JSON e o exibe no elemento `<div>` cujo `id` é indicado:

```
$('#botao').click(function () {
    $.getJSON('exemplo.json', function (resposta) {
        document.getElementById("conteudo").innerHTML = resposta;
    });
});
```



Enviando e recebendo dados com jQuery

- A função `ajax()` é mais completa e fornece propriedades adicionais que permitem um controle mais detalhado sobre as requisições HTTP:

```
$.ajax({
    url: 'pagina.html',
    type: 'GET',
    timeout: 12000,
    dataType: 'text'
}).done(function (resposta) {
    $('#conteudo').text(resposta);
}).fail(function () {
    alert('Ocorreu um erro durante a requisição');
});
```



Enviando e recebendo dados com jQuery

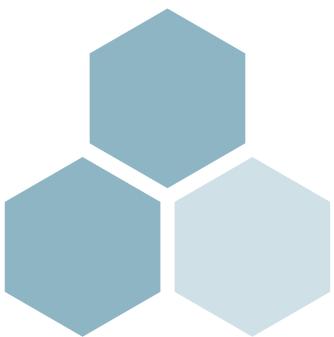
- Para incluir dados usando a função **ajax()**, usamos a propriedade **data**. No exemplo, o nome de uma pessoa fornecido em um formulário é passado para a função:

```
$.ajax({  
    url: 'pagina.aspx',  
    type: 'GET',  
    timeout: 12000,  
    dataType: 'text'  
    data: {  
        nome: formulario.txtNome.value;  
    }  
}).done(function (resposta) {  
    $('#conteudo').text(resposta);  
}).fail(function () {  
    alert('Ocorreu um erro durante a requisição');  
});
```




11

Criando objetos com JavaScript - Prototype



Editora
IMPACTA

Criando objetos customizados

No JavaScript, o que conhecemos como objetos (instâncias de classes) são apenas elementos formados por pares chave/valor, e suas propriedades são anexadas dinamicamente.

- Objetos podem ser criados de uma das duas formas:

```
var obj1 = new Object();
var obj2 = {};
```

- Podemos, também, definir as propriedades:

```
obj1.data = new Date();
obj2["data"] = new Date();
```



Criando objetos customizados

As funções, diferentemente de métodos em linguagens orientadas a objetos, podem ser definidas de forma explícita ou implícita. Funções implícitas são, também, chamadas de funções anônimas.

```
function func(x) {
    alert(x);
}
func("JavaScript");

var funcao = function (x) {
    alert(x);
}
funcao("JavaScript");
```



Criando objetos customizados

Temos, ainda:

```
var funcao = new Function("x","alert(x)");
funcao("JavaScript");
```

Uma função, na verdade, é um objeto, cujo objetivo é executar uma tarefa.



Criando objetos customizados

Podemos ter uma variedade para o uso de funções:

1. Passando como parâmetro;
2. Definindo funções anônimas como **callback**;
3. Definindo como atributo de outros objetos.

No caso do item 3, podemos ter:

```
var aluno = {
    nome: "Beto",
    curso: "Engenharia",
    exibir: function () {
        alert("Nome = " + nome + "\nCurso = " + curso);
    }
};
```

Esse modelo é o que mais se aproxima de uma classe em Java ou C#.





Criando objetos customizados

Quanto à referência **this**, podemos ter diferentes valores, dependendo da chamada. O exemplo seguinte ilustra esse processo:

```
function exibirTexto() {  
    //o valor de this é alterado,  
    //dependendo de onde ele é chamado  
    alert(this.mensagemTexto);  
}  
  
var williamShakespeare = {  
    mensagemTexto: "Grande poeta William Shakespeare",  
    mostrar: exibirTexto  
};
```



Criando objetos customizados

```
var markTwain = {  
    mensagemTexto: "Excelente escritor",  
    mostrar: exibirTexto  
};  
  
var oscarWilde = {  
    mensagemTexto: "Excelente dramaturgo",  
    //mostrar: exibirTexto  
};  
  
williamShakespeare.mostrar(); //exibe Grande poeta William Shakespeare  
markTwain.mostrar(); //exibe Excelente escritor  
exibirTexto.call(oscarWilde); //exibe Excelente dramaturgo
```



Estendendo objetos - Prototype

- O objeto **prototype** é bastante importante em JavaScript, pois é a partir dele que podemos definir herança entre objetos (e não entre classes, como em Java ou C#)
- Todas as funções no JavaScript possuem uma propriedade chamada **prototype** que, por sua vez, possui uma propriedade chamada **constructor**, que referencia a própria função
- Em outras palavras, todos os objetos em JavaScript são gerados com base em um protótipo, como se fosse um modelo pronto a partir do qual novos objetos são gerados
- Por se tratar de um conceito pouco trivial, vamos desenvolver um exemplo e explicá-lo a cada etapa de sua elaboração



Estendendo objetos - Prototype

- Considere a seguinte função e observe seu aspecto e a semelhança com uma classe:

```
function Pessoa(nome, idade) {  
    this.nome = nome;  
    this.idade = idade;  
  
    this.exibirInfo = function () {  
        var msg;  
        msg = "Nome: " + this.nome + ", Idade: " + this.idade;  
    }  
}
```

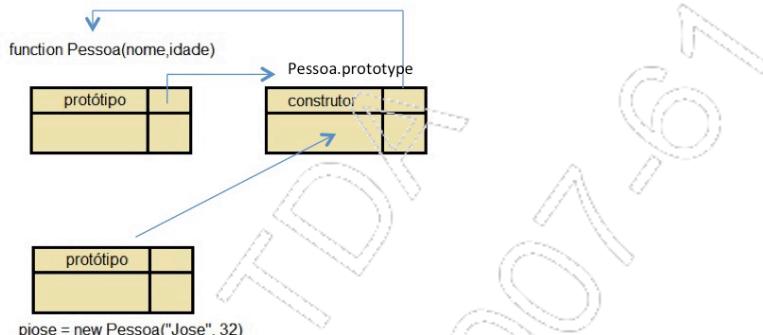
- Podemos definir um objeto da seguinte forma:

```
var pjose = new Pessoa("Jose", 32);
```



Estendendo objetos - Prototype

- Veja sua representação:



Estendendo objetos - Prototype

- O objeto referenciado por `pjose` herda as propriedades e métodos de seu protótipo, que é `Pessoa.prototype`.
- O termo `Pessoa("Jose", 32)` é equivalente a `Pessoa.prototype.constructor("Jose", 32)`
- Neste sentido, é possível executarmos a instrução que nos apresentará `true`:

```
alert(Pessoa.prototype.isPrototypeOf(pjose));
```

Observe o uso do método `isPrototypeOf`. Mas de onde vem esse método?



Estendendo objetos - Prototype

- No JavaScript, existe um mecanismo de herança baseado em objetos, como já descrito anteriormente
- Todos os protótipos, como em **Pessoa.prototype**, herdam todas as definições de **Object.prototype**. Não há nenhum objeto acima de **Object**, de forma que sua propriedade **prototype** é **null**
- Apesar de não estar representado, o que temos é:

```
Pessoa.prototype = new Object();
```



Estendendo objetos - Prototype

- Definir métodos e/ou propriedades no protótipo de um objeto fica automaticamente visível a todos os herdeiros
- Se em um objeto herdeiro for definido o mesmo método que existia no objeto pai, seu conteúdo fica sobreescrito no objeto herdeiro





Estendendo objetos - Prototype

Suponhamos, agora, que desejamos definir uma herança de **Pessoa**, por exemplo, uma função **Aluno**. Veja o código:

```
function Aluno(nome, idade, curso) {  
    Pessoa.call(this, nome, idade);  
    this.curso = curso;  
  
    //Esta instrução viabiliza a herança  
    Aluno.prototype = new Pessoa(nome, idade);  
  
    //Toda instância de Aluno deve apontar para Aluno  
    Aluno.prototype.constructor = Aluno;  
  
    //Agora vamos sobreescriver o método exibirInfo  
    Aluno.prototype.exibirInfo = function () {  
        var msg;  
        msg = "Nome: " + nome + ", Idade: " + idade + ", Curso: " + this.curso;  
        return msg;  
    }  
}
```



Estendendo objetos - Prototype

- Mais sobre **prototype** pode ser pesquisado na fonte a seguir:

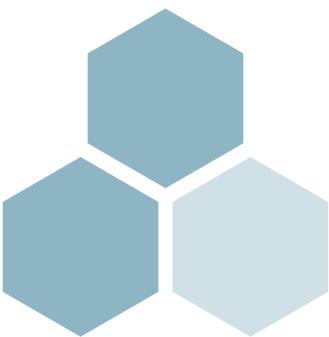
Fonte: http://www.w3schools.com/js/js_object_prototypes.asp





12

Criando páginas interativas e suporte off-line



Editora
IMPACTA

Introdução

- Os termos **Web** e **on-line** estão intimamente ligados, tanto que quando pensamos em Web off-line parece estranho
- Uma aplicação off-line é aquela que mantém os dados armazenados localmente e, de alguma forma, esses dados são acessados quando necessário
- Este recurso pode ser usado para melhorar o desempenho de aplicativos, tornar os dados persistentes entre sessões de usuário ou ao atualizar e restaurar páginas
- Vamos conhecer os mecanismos usados para esta tarefa bastante importante



Persistindo dados com Session Storage

- Por definição, uma aplicação Web é stateless (cada requisição é nova, não mantendo o estado referente aos dados de requisições anteriores)
- Para evitar a falta de vínculo entre duas requisições, especialmente quando a segunda depender de informações da primeira, é necessário algum mecanismo de armazenamento das informações
- Um mecanismo para armazenar o estado de uma página é o uso de **sessionStorage** ou **localStorage**



Persistindo dados com Session Storage

- Usamos o objeto **sessionStorage** para armazenar e recuperar dados por meio de uma sessão. Temos três maneiras:

```
sessionStorage.setItem("chave", "texto a ser armazenado");
var sessao1 = sessionStorage.getItem("chave");

sessionStorage["chave"] = "texto a ser armazenado";
var sessao2 = sessionStorage["chave"];

sessionStorage.chave = "texto a ser armazenado";
var sessao3 = sessionStorage.chave;
```

- Os dados da sessão são disponíveis apenas na sessão que os criar
- Esses dados serão excluídos quando o usuário finalizar o navegador



Persistindo dados com Session Storage

- Recuperando dados de uma sessão e apresentando-os para o usuário (supondo que os dados armazenados na sessão sejam exibidos em forma de lista):

```
var lista = document.getElementById("lista");
for (var i = 0; i < sessionStorage.length; i++) {
    lista.innerHTML += "<br />" + sessionStorage.key(i);
}
```



Persistindo dados com Session Storage

- Para remover um item, usamos a função **removeItem**:

```
sessionStorage.removeItem("myKey");
```

- Pra excluir uma sessão:

```
sessionStorage.clear();
```

Persistindo dados com Local Storage

- Usamos o objeto **localStorage** para armazenar e recuperar dados por meio de sessões e de páginas de uma aplicação:

```
localStorage.setItem("chave", "texto a ser armazenado");
var sessao1 = localStorage.getItem("chave");

localStorage["chave"] = "texto a ser armazenado";
var sessao2 = localStorage["chave"];

localStorage.chave = "texto a ser armazenado";
var sessao3 = localStorage.chave;
```

- Os dados permanecem persistentes até que sejam explicitamente removidos

Persistindo dados com Local Storage

- Podemos usar o evento **storage** para notificar uma aplicação Web sobre alterações realizadas em **session** e **local storage**:

```
function storageCallback( e ) {
    alert( "Chave:" + e.key + " alterada para " + e.newValue );
}

window.addEventListener("storage", storageCallback, true );
```

- Propriedades do evento **storage**:

- key**: nome do valor a ser alterado
- oldValue**: valor original
- newValue**: novo valor
- url**: origem do evento
- storageArea**: referência ao armazenamento que foi alterado



Persistindo dados com IndexedDB

- IndexedDB** fornece um mecanismo eficiente para armazenar dados estruturados no computador do usuário
- A API é assíncrona e inclui as seguintes características:
 - Armazenamento de múltiplos objetos
 - Operações: **add()**, **put()**, **get()** e **delete()** (CRUD)
 - Transações
 - Uso de cursores
 - Índices para agilizar buscas comuns



Persistindo dados com IndexedDB

- Usamos a propriedade **IndexedDB** no objeto **window**
- Armazenamos dados em um banco de dados nomeado

```
var db; // Referência ao banco de dados
var requisicao = indexedDB.open("bancoDB");

requisicao.onsuccess = function (evento) {
    db = evento.target.result;
};

requisicao.onerror = function (evento) {
    alert("Error " + evento.target.errorCode + " na abertura do banco");
};
```



Persistindo dados com IndexedDB

- Um banco de dados pode armazenar um ou mais objetos, de forma semelhante a um banco de dados relacional
- Podemos definir um objeto de armazenamento usando a função **createObjectStore**
- Especificamos o objeto a ser armazenado juntamente com uma chave que permitirá sua busca



Persistindo dados com IndexedDB

- No exemplo seguinte, um objeto é criado para armazenar os dados de um usuário. A propriedade **id** é a chave deste objeto:

```
var usuario = {  
    id: "1",  
    name: "Benedito Silva",  
    password: "Imp@ct@"  
};  
var usuarioStore = db.createObjectStore("usuarios", { keyPath: "id" });
```



Persistindo dados com IndexedDB

- Usamos a função **add()** para incluir objetos adicionais:

```
var usuarioStore = db.createObjectStore("usuarios", { keyPath: "id" });  
  
var novoUsuario = {  
    id: "2",  
    name: "Euclides da Cunha",  
    password: "Secret@"  
};  
  
var adiciona = usuarioStore.add(novoUsuario);
```



Persistindo dados com IndexedDB

- Podemos manipular os eventos de sucesso ou erro:

```
var usuarioStore = db.createObjectStore("usuarios", { keyPath: "id" });

var novoUsuario = {
    id: "2",
    name: "Euclides da Cunha",
    password: "Secret@"
};

var adiciona = usuarioStore.add(novoUsuario);

adiciona.onsuccess = function(evento) {
    //evento para sucesso
};
adiciona.onerror = function(evento) {
    //evento para erro
};
```

Persistindo dados com IndexedDB

- Para alterar um registro existente, usamos a função **put()**:

```
var atualizaUsuario = {
    id: "2",
    name: "Euclides da Cunha",
    password: "novasenha*@"
};

var atualiza = usuarioStore.put(novoUsuario);

atualiza.onsuccess = function(evento) {
    //evento para sucesso
};
atualiza.onerror = function(evento) {
    //evento para erro
};
```

Persistindo dados com IndexedDB

- Para remover um registro, usamos a função **delete()**:

```
var remove = usuarioStore.delete("1");

remove.onsuccess = function(evento) {
    //evento para sucesso
};

remove.onerror = function(evento) {
    //evento para erro
};
```



Persistindo dados com IndexedDB

- Para buscar um registro, usamos a função **get()**:

```
var usuario;

var requisicao = usuarioStore.get("1")
remove.onsuccess = function(evento) {
    //evento para sucesso
    usuario = event.target.result
};

remove.onerror = function(evento) {
    //evento para erro
};
```

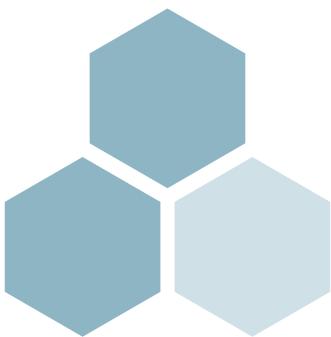
- Elaborar a Atividade 12 – Armazenamento local





13

Complemento: Aplicações híbridas com PhoneGap



Editora
IMPACTA

Conhecendo o Apache Cordova

- Cordova é uma plataforma para desenvolvimento de aplicações mobile com HTML5, CSS3 e JavaScript
- Seu princípio de funcionamento é estabelecer uma conexão entre as aplicações Web e os recursos de um dispositivo móvel, o que não é disponível apenas para aplicações Web

Conhecendo o Apache Cordova

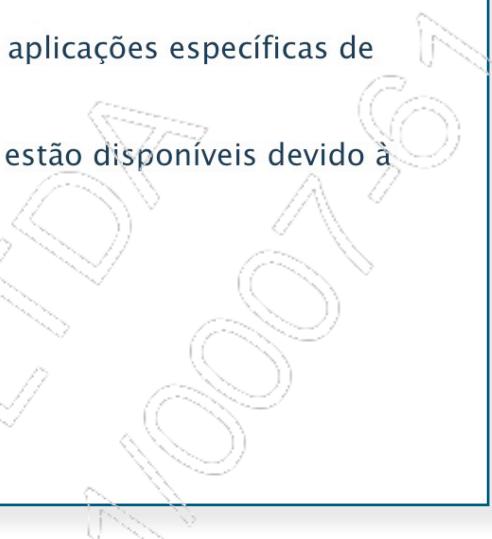
Vantagens:

- Podemos desenvolver aplicações híbridas, ou seja, que podem ser usadas em diferentes plataformas (essencialmente iOS e Android)
- O desenvolvimento é rápido, poupando tempo de desenvolvimento para várias plataformas
- Como usamos JavaScript, não precisamos aprender dados específicos da plataforma

Conhecendo o Apache Cordova

Desvantagens:

- Aplicações híbridas são mais lentas que aplicações específicas de uma plataforma
- Recursos específicos de plataforma não estão disponíveis devido à limitação das aplicações híbridas



Conhecendo o Apache Cordova

Diferenças entre PhoneGap e Cordova:

- No início, o Cordova se chamava PhoneGap, quando foi desenvolvido por uma empresa chamada Nitobi. Tempos depois, a empresa Adobe adquiriu a Nitobi, doando o código do Cordova para a fundação Apache. A Apache, por sua vez, nomeou o projeto de Cordova
- Isso significa que os dois (Cordova e PhoneGap) são, em essência, a mesma coisa



Conhecendo o Apache Cordova

- As diferenças são sutis e dizem respeito ao processo de criação da aplicação
- As linhas de comando, por exemplo, para criação de projetos tanto para Cordova como para PhoneGap também são diferentes, porém o resultado final é o mesmo

Criando projetos PhoneGap

O PhoneGap permite gerar aplicativos para diferentes plataformas. Sendo assim, temos à disposição diferentes ferramentas para criar os aplicativos.

Uma delas é usar o Node.js. Por meio do Node.js podemos executar instruções em linhas de comando para gerar os projetos.

```
npm install -g phonegap
```

Criando projetos PhoneGap

Não basta ter o componente Cordova instalado. Se desejarmos compilar nossos projetos para uma plataforma nativa (Android ou iOS), temos que ter disponível o ambiente de desenvolvimento específico para cada uma delas:

- **Android SDK** (<https://developer.android.com/index.html>)
ou
- **Xcode** (<https://developer.apple.com/downloads/>)



Instalação e configuração

- Instale o **git**, usado para alguns processos em segundo plano e disponível em <<https://git-scm.com/downloads>>
- Instale o componente PhoneGap. Usamos o NodeJS para instalar o módulo Cordova globalmente:

```
npm install -g phonegap
```
- A partir deste ponto, estamos prontos para criar nossos projetos.



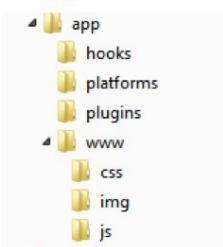
Criando projetos PhoneGap

Uma vez instalado o componente **PhoneGap**, podemos criar nosso projeto. Os passos são dados a seguir:

- Defina uma pasta onde seu projeto será criado;
- No prompt de comandos, acesse essa pasta;
- Suponha que queiramos criar um projeto chamado **AppCordova**, na pasta **app**. A linha de comando é esta:

```
>phonegap create app app.exemplo AppCordova
```

Criando projetos PhoneGap

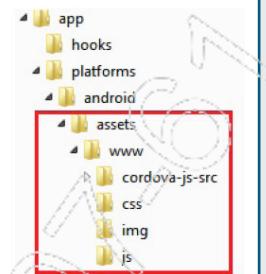
- Este comando cria a seguinte estrutura:
- A pasta **www** contém o núcleo da nossa aplicação. É aqui que colocaremos o código que será comum para todas as aplicações
- A pasta **platforms** é usada pelo Cordova para compilar sua aplicação para diferentes plataformas e não deve ser manipulada pelo usuário
- Na pasta **plugins**, estão os plugins que inserimos no projeto

Criando projetos PhoneGap

- Os parâmetros da aplicação são armazenados no arquivo **config.xml**, em **www**
- Agora vamos definir para qual plataforma desenvolveremos. Optamos por desenvolver para Android, neste exemplo. É necessário entrar no diretório do projeto:

```
C:\Documentos\Projetos\Cordova\app\cordova platform add android
```

- É importante que o **android sdk** esteja instalado
- Para cada plataforma adicionada, será criada uma nova pasta e, dentro dela, na subpasta **assets**, haverá uma cópia da pasta **www**



Criando aplicativos – PhoneGap Build

Para criarmos a versão final de um aplicativo, devemos ter disponível a plataforma completa instalada, como é o caso do SDK do Android. Isso demanda uma instalação complexa, e ainda exige o conhecimento adicional de configuração da plataforma.

Para evitar esse problema, a Adobe (proprietária do PhoneGap) nos oferece um recurso bastante simplificado de compilação de aplicativos.



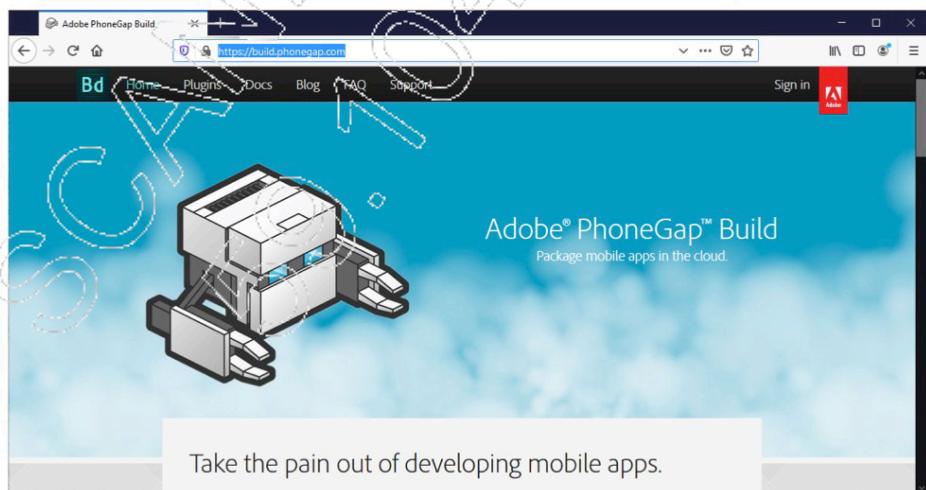
Criando aplicativos – PhoneGap Build

Para acessar o PhoneGap Build, devemos ter uma conta na Adobe. Se não tiver, basta criá-la, pois o processo é simples.

O link do PhoneGap build é <https://build.phonegap.com/>

Ao acessar esse link, obtemos uma interface semelhante à mostrada a seguir:

Criando aplicativos – PhoneGap Build



Criando aplicativos – PhoneGap Build

Basta seguir o processo de criação de um novo usuário ou fornecer seus dados, caso já esteja cadastrado.

Para exemplificar, vamos usar o projeto **app** criado anteriormente. Abra-o no Visual Studio Code (abra a pasta **app**):

Treinamentos
IMPACTA

Criando aplicativos – PhoneGap Build

Show All Commands **Ctrl + Shift + P**

Go to File **Ctrl + P**

Find in Files **Ctrl + Shift + F**

Start Debugging **F5**

Toggle Terminal **Ctrl + T**

Treinamentos
IMPACTA

Criando aplicativos – PhoneGap Build

A partir deste ponto, é possível acessar a pasta **www** e, nela, o arquivo **index.html**. Este é o arquivo representando a página inicial do projeto.

Na raiz, o arquivo **config.xml** contém as configurações do aplicativo. Consulte a documentação em <http://docs.phonegap.com/phonegap-build/configuring/>.

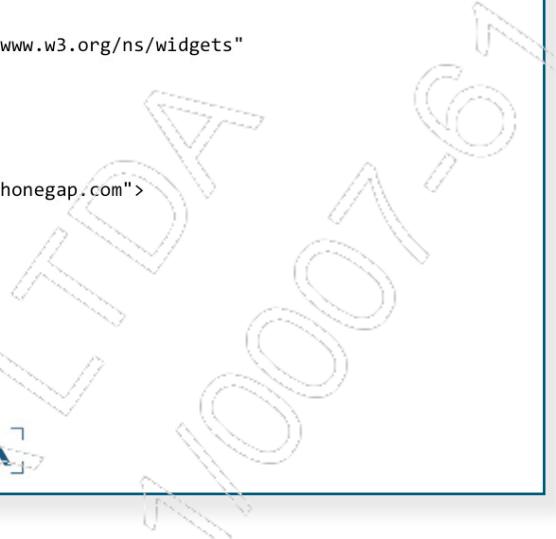
Criando aplicativos – PhoneGap Build

Vamos incluir um conteúdo básico no arquivo **index.html** com o objetivo de ilustrar a geração de um aplicativo Android. Além disso, incluiremos algumas configurações no arquivo **config.xml**.

Criando aplicativos – PhoneGap Build

- config.xml

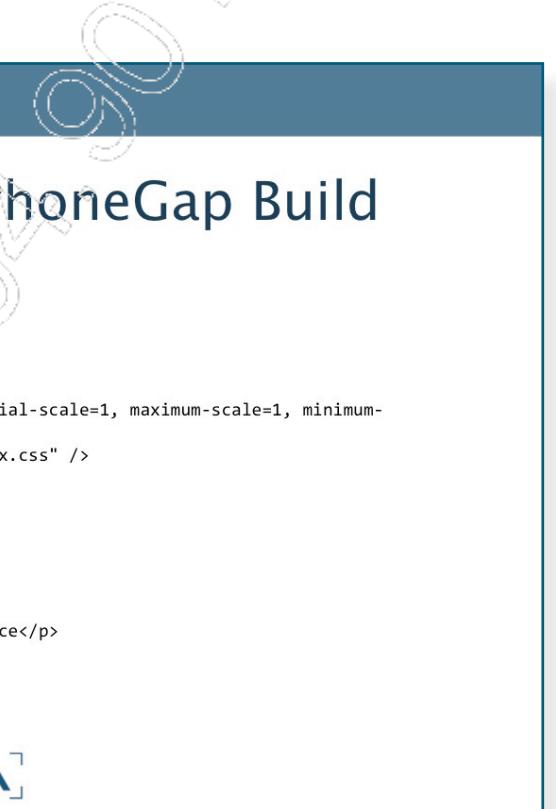
```
<?xml version='1.0' encoding='utf-8'?>
<widget id="app.exemplo" version="1.0.0" xmlns="http://www.w3.org/ns/widgets"
      xmlns:gap="http://phonegap.com/ns/1.0">
    <name>AppPhonegap</name>
    <description>
        Aplicativo de exemplo.
    </description>
    <author email="support@phonegap.com" href="http://phonegap.com">
        Impacta - Front End
    </author>
    <content src="index.html" />
<!-- restante do arquivo, omitido por brevidade -->
```



Criando aplicativos – PhoneGap Build

- index.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-scale=1, minimum-
scale=1, width=device-width" />
    <link rel="stylesheet" type="text/css" href="css/index.css" />
    <title>Hello World</title>
</head>
<body>
    <div class="app">
        <h1>PhoneGap</h1>
        <div id="deviceready" class="blink">
            <p>event listening</p>
        </div>
    </div>
</body>
</html>
```



Criando projetos – PhoneGap Build

Uma vez definido o projeto, podemos proceder de duas formas:

1. Compactar a pasta contendo o arquivo **config.xml**. No nosso exemplo, a pasta **app**;
2. Publicar o projeto no GitHub.



Criando projetos – PhoneGap Build

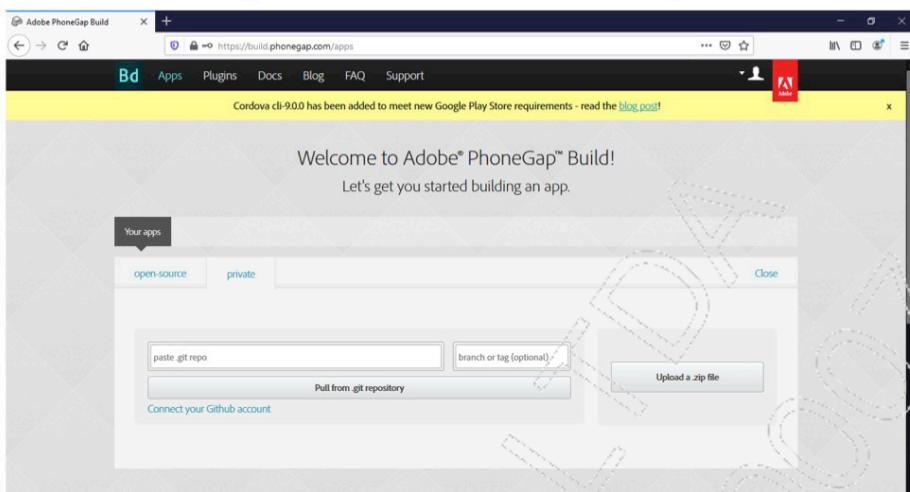
O projeto compactado será usado para publicação no PhoneGap Build.

Para essa tarefa, insira as credenciais no site <https://build.phonegap.com/>

Obteremos uma interface semelhante à mostrada a seguir:



Criando projetos – PhoneGap Build



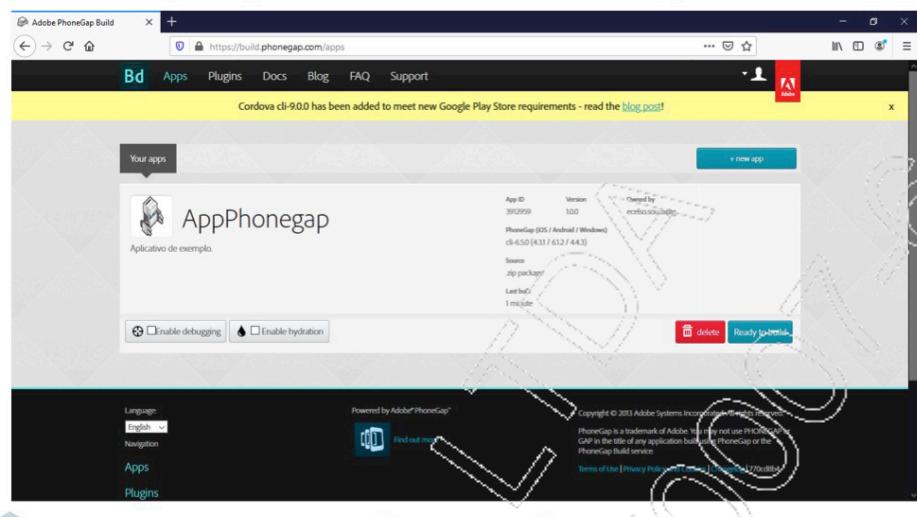
Criando projetos – PhoneGap Build

A conta gratuita permite fazer o upload de um arquivo compactado no modo **private**, e diversos projetos públicos, desde que o projeto esteja no GitHub. Basta incluir o endereço do repositório no GitHub.

No nosso caso, vamos fazer o upload ao arquivo compactado. Após o upload, teremos a seguinte interface:



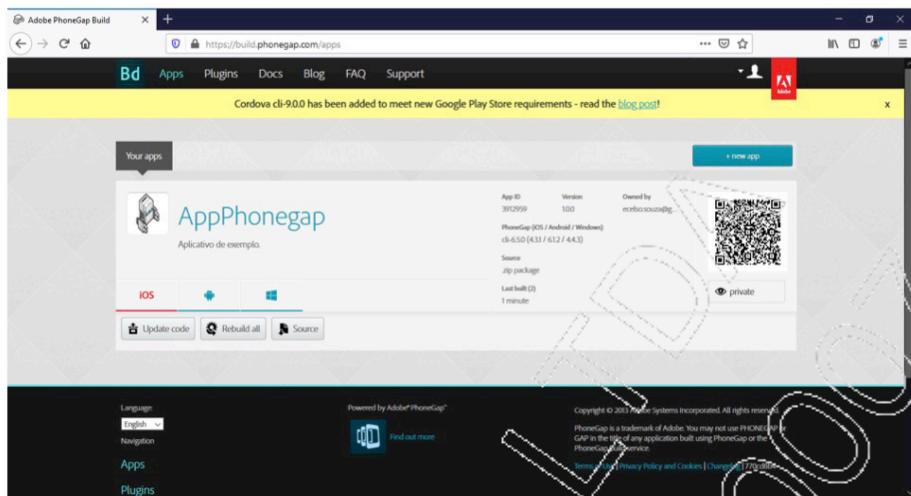
Criando projetos – PhoneGap Build



Criando projetos – PhoneGap Build

Clique no botão Ready to build. O aplicativo será compilado.

Criando projetos – PhoneGap Build



Criando projetos – PhoneGap Build

Desejando obter o aplicativo, basta clicar no ícone correspondente. Vamos obter a versão para Android.

Observe que o arquivo será baixado. Use os recursos que tiver disponível para executar a aplicação no seu celular. Alternativamente você pode apontar a câmera do celular para o código de barras.

Você já tem um aplicativo desenvolvido a partir de uma plataforma completa!

Criando projetos – PhoneGap Build

- Elaborar a Atividade 13 – PhoneGap

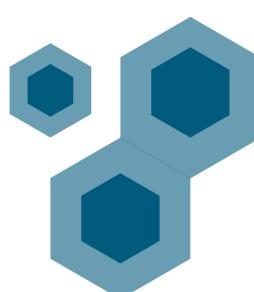


Desenvolvimento Front End

Elaborar as atividades referentes a Mão à obra:

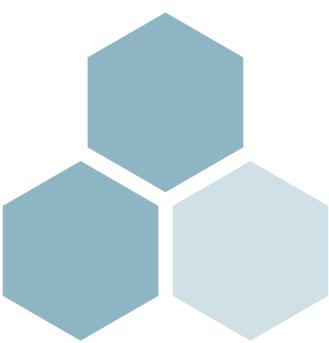
- Lab 01 – Criando uma página HTML
- Lab 02 – Desenvolvendo formulários
- Lab 03 – Aplicando estilos
- Lab 04 – Aplicando responsividade
- Lab 05 – JavaScript e jQuery
- Lab 06 – Comunicação com o servidor
- Lab 07 – Suporte off-line
- Lab 08 - PhoneGap

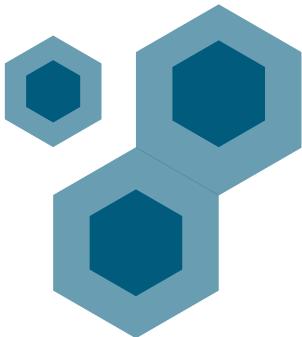




Atividades (projeto)

SCANNING
59.704.
ALTA
10007-67





Apresentando o projeto

SCANNING
59.704.
ALTA
10007-67

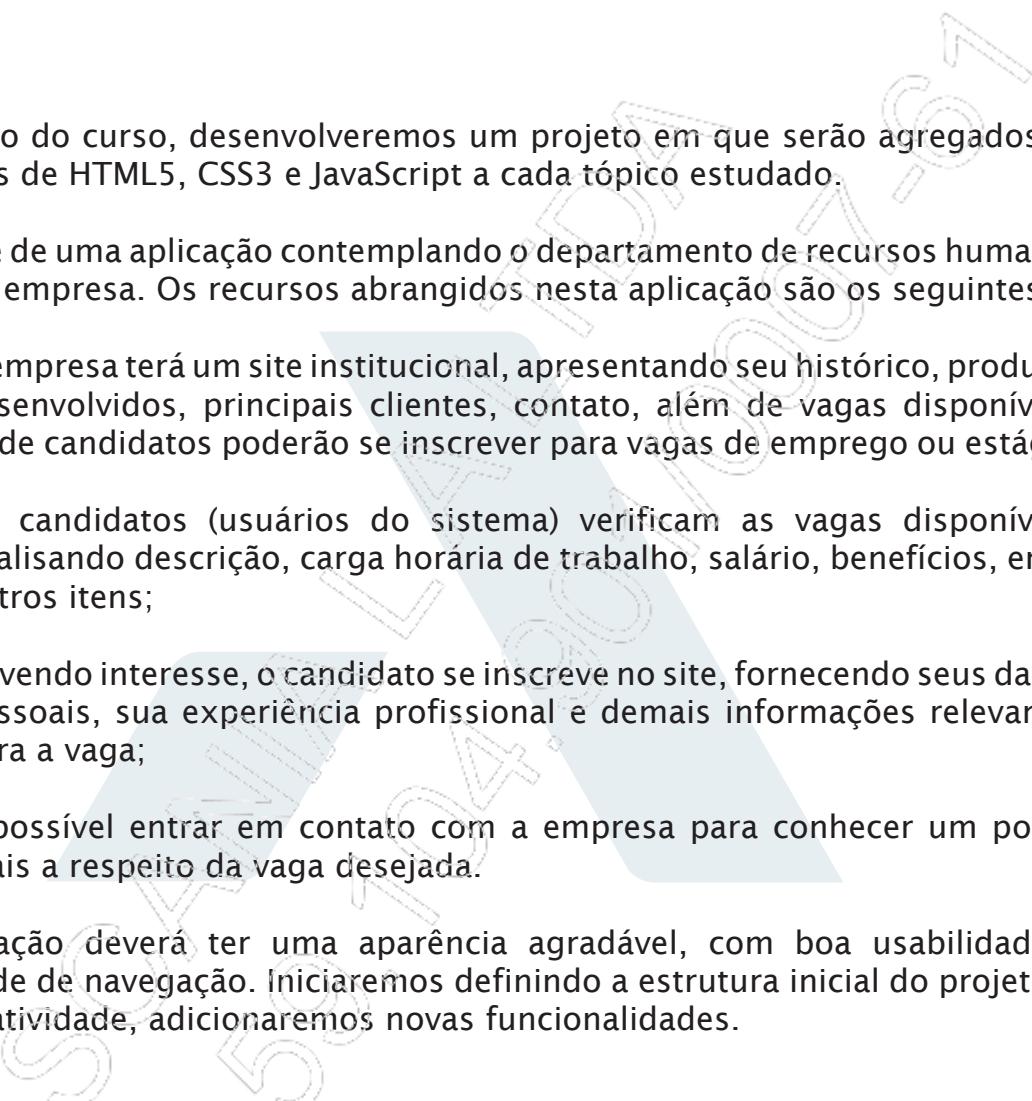


Ao longo do curso, desenvolveremos um projeto em que serão agregados os recursos de HTML5, CSS3 e JavaScript a cada tópico estudado.

Trata-se de uma aplicação contemplando o departamento de recursos humanos de uma empresa. Os recursos abrangidos nesta aplicação são os seguintes:

- A empresa terá um site institucional, apresentando seu histórico, produtos desenvolvidos, principais clientes, contato, além de vagas disponíveis, onde candidatos poderão se inscrever para vagas de emprego ou estágio;
- Os candidatos (usuários do sistema) verificam as vagas disponíveis, analisando descrição, carga horária de trabalho, salário, benefícios, entre outros itens;
- Havendo interesse, o candidato se inscreve no site, fornecendo seus dados pessoais, sua experiência profissional e demais informações relevantes para a vaga;
- É possível entrar em contato com a empresa para conhecer um pouco mais a respeito da vaga desejada.

A aplicação deverá ter uma aparência agradável, com boa usabilidade e facilidade de navegação. Iniciaremos definindo a estrutura inicial do projeto e, a cada atividade, adicionaremos novas funcionalidades.





1

Preparando o ambiente

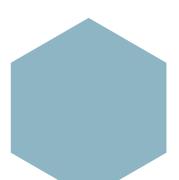


Atividade



Editora

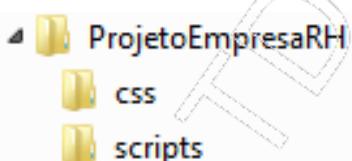
IMPACTA



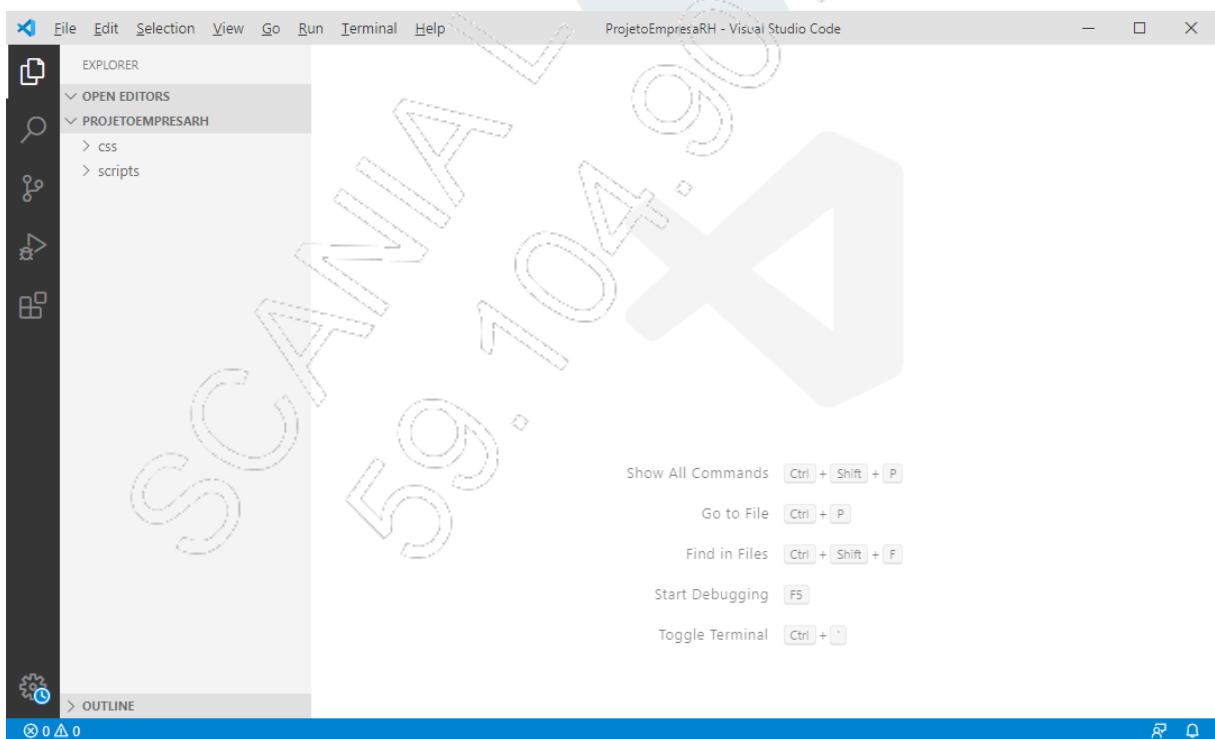
Laboratório 1

A - Definindo a estrutura inicial para o projeto

1. Em um local de sua preferência (unidade C, pasta **Documentos** etc.), defina a estrutura de pastas a seguir (outras pastas serão adicionadas ao longo do curso):



2. Abra o Visual Studio Code. Selecione a pasta **ProjetoEmpresaRH**. Esta será a base para a elaboração do projeto, também usada como raiz da aplicação Web:



A partir deste ponto, podemos adicionar, alterar ou remover pastas e arquivos para nossa aplicação.



2

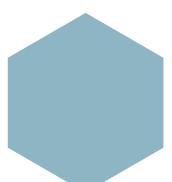
Criação de páginas com HTML5



Atividade

Editora

IMPACTA



Laboratório 1

A – Definindo o modelo referente à página inicial da aplicação

Vamos iniciar nosso projeto, começando pela página inicial.

1. Abra o Visual Studio Code (se não estiver aberto) apontando para a pasta **ProjetoEmpresaRH**. O resultado, no VSCode, deverá ser semelhante a este:



2. Inclua, na raiz do site, o arquivo **index.html**. Este arquivo representará a página inicial da aplicação. Inclua o elemento DOCTYPE, o cabeçalho, o corpo da página e os elementos principais. Inicie pela seguinte estrutura:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Site Institucional - ABC Tecnologia</title>
</head>
<body>
</body>
</html>
```

3. No elemento <body>, insira estes elementos:

```
<body>
  <header>
    <nav>

      </nav>
    </header>
    <section>
      <article>
        <header>

          </header>
        </article>
      </section>
      <aside>

        </aside>
      <footer>
        <p>Copyright 2020 - Todos os direitos reservados</p>
      </footer>
    </body>
```

4. Insira os elementos a seguir no elemento <nav>:

```
<nav>
  <ul>
    <li><a href="#">A Empresa</a></li>
    <li><a href="#">Produtos</a></li>
    <li><a href="#">Localização</a></li>
    <li><a href="#">Trabalhe Conosco</a></li>
  </ul>
</nav>
```

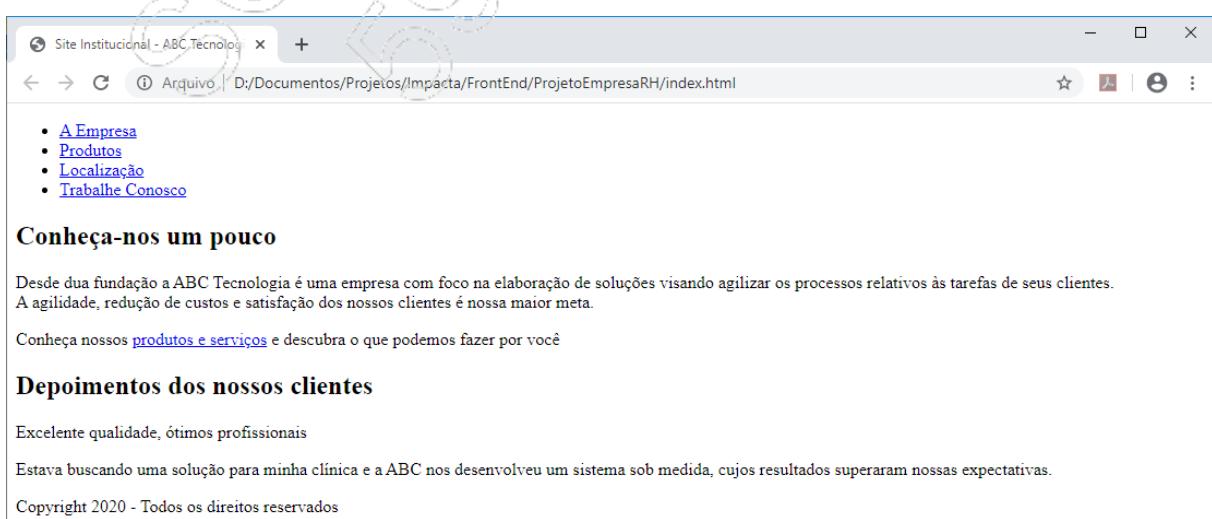
5. Agora, no elemento **<header>**, abaixo de **<article>** e **<section>**, e logo abaixo do fechamento do elemento **</header>**, deixe como indicado a seguir:

```
<section>
  <article>
    <header>
      <h2>Conheça-nos um pouco</h2>
      <p>
        Desde sua fundação a ABC Tecnologia é uma
        empresa com foco na elaboração de soluções
        visando agilizar os processos relativos às
        tarefas de seus clientes. <br/>
        A agilidade, redução de custos e satisfação
        dos nossos clientes é nossa maior meta.
      </p>
    </header>
    <p>
      Conheça nossos <a href="#">produtos e serviços</a>
      e descubra o que podemos fazer por você
    </p>
  </article>
</section>
```

6. Complete o elemento **<aside>**:

```
<aside>
  <h2>Depoimentos dos nossos clientes</h2>
  <p>Excelente qualidade, ótimos profissionais</p>
  <p>
    Estava buscando uma solução para minha clínica
    e a ABC nos desenvolveu um sistema sob medida,
    cujos resultados superaram nossas expectativas.
  </p>
</aside>
```

Ao executar a página no navegador, o resultado deve ser similar ao da imagem adiante:



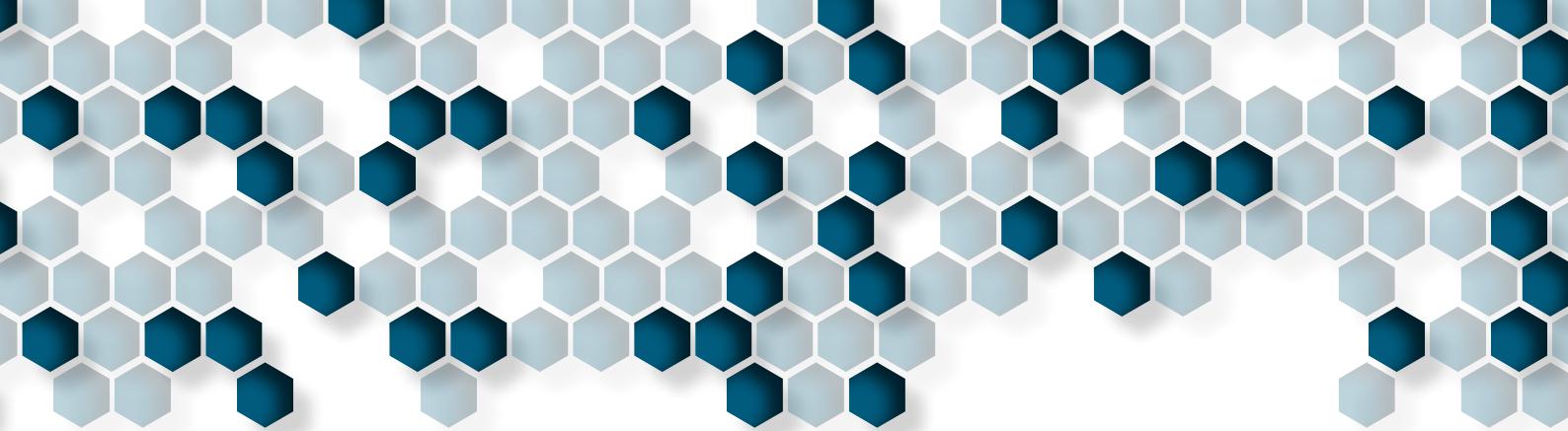


3

Inclusão de formulários

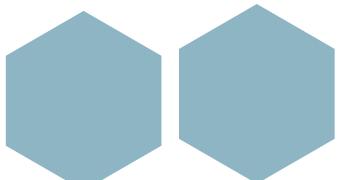


Atividade



Editora

IMPACTA

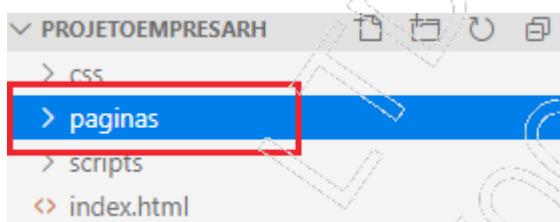


Laboratório 1

A – Definindo o formulário para o usuário se inscrever para uma vaga

Nesta etapa, incluiremos um formulário para o usuário se inscrever para uma vaga de emprego ou estágio. Utilizaremos campos adequados para cada informação e realizaremos as devidas validações para esses campos.

1. Na estrutura do projeto, inclua uma nova pasta chamada **paginas**. A estrutura deverá ficar como o modelo a seguir:



Esta pasta deverá ser criada no próprio Visual Studio Code.

2. Na pasta **paginas**, inclua um arquivo chamado **registro.html**. A estrutura básica do HTML desse novo arquivo deverá ter o seguinte aspecto:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title></title>
</head>
<body>
</body>
</html>
```

3. No elemento **<body>** do arquivo **registro.html**, insira um elemento **<div>**, um título e um elemento **<form>**:

```
<body>
  <div>
    <h1>Registro de Candidatos</h1>
    <form>

      </form>
    </div>
  </body>
```

No formulário serão inseridos os elementos para o registro de um novo candidato. Os itens do registro são:

- Nome;
- Data de nascimento;
- Sexo (masculino ou feminino);
- Telefone;
- E-mail;
- Vaga a se candidatar.

Os campos terão validadores de acordo com a natureza e o tipo da informação.

Usaremos uma combinação de elementos **<div>**, **<label>** e **<input>** para viabilizar a futura formatação via CSS.

! É extremamente importante que o aluno não use o recurso “copiar e colar” e escreva os elementos. Isso é importante para seu aprendizado e experiência.

4. Insira os campos nesta ordem, dentro do elemento <form>:

```
<form>
    
    <div>
        <label for="nome">Nome:</label>
        <div>
            <input type="text" id="nome" name="nome"
                   required="required" autofocus="autofocus" />
        </div>
    </div>

    
    <div>
        <label for="data">Data Nascimento:</label>
        <div>
            <input type="date" id="data" name="data"
                   required="required" />
        </div>
    </div>

    
    <div>
        <label for="sexo">Sexo:</label>
        <div>
            <fieldset>
                <legend>Sexo:</legend>
                <input type="radio" id="masculino" name="sexo"
                       value="masculino" >Masculino<br/>
                <input type="radio" id="feminino" name="sexo"
                       value="feminino" >Feminino<br/>
            </fieldset>
        </div>
    </div>

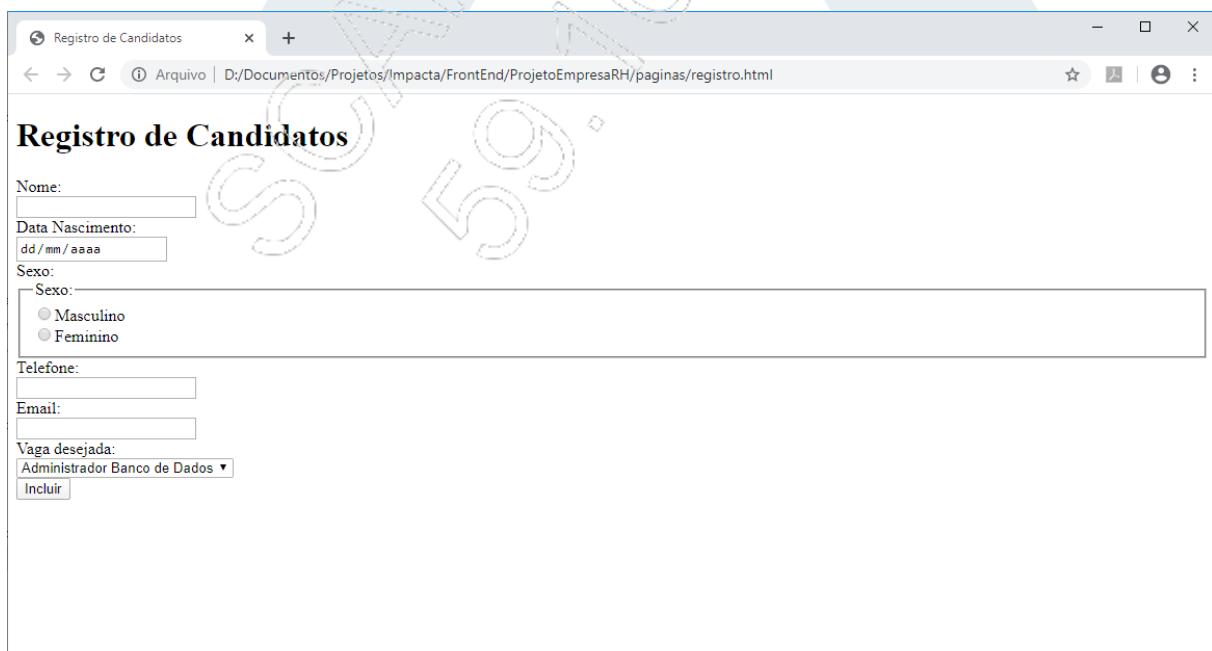
    
    <div>
        <label for="telefone">Telefone:</label>
        <div>
            <input type="tel" id="telefone" name="telefone"
                   required="required" />
        </div>
    </div>
```

```
<!-- Email -->
<div>
  <label for="email">Email:</label>
  <div>
    <input type="email" id="email" name="email"
      required="required" />
  </div>
</div>

<!-- Lista de Vagas -->
<div>
  <label for="vaga">Vaga desejada:</label>
  <div>
    <select name="vaga" id="vaga">
      <option value="1">Administrador Banco de Dados</option>
      <option value="2">Analista de Sistemas</option>
      <option value="3">web Designer</option>
      <option value="4">Programador Java</option>
    </select>
  </div>
</div>

<!-- Botão de comandos -->
<div>
  <input type="button" value="Incluir">
</div>
</form>
```

5. Teste o formulário. Não se preocupe com a falta de formatação. Ela será realizada oportunamente com CSS. O resultado é semelhante ao mostrado a seguir:



The screenshot shows a web browser window titled "Registro de Candidatos". The page contains the following form elements:

- Nome: [Text input field]
- Data Nascimento: [Text input field with placeholder "dd/mm/aaaa"]
- Sexo:
 - Sexo: [Text input field]
 - Masculino
 - Feminino
- Telefone: [Text input field]
- Email: [Text input field]
- Vaga desejada:
 - Administrador Banco de Dados
- Incluir [Large button]

6. No arquivo **index.html** (página inicial), redefina os links para registro e contato. Deixe-os conforme o modelo adiante:

```
<nav>
  <ul>
    <li><a href="#">A Empresa</a></li>
    <li><a href="#">Produtos</a></li>
    <li><a href="#">Localização</a></li>
    <li><a href="paginas/registro.html">Trabalhe Conosco</a></li>
  </ul>
</nav>
```

7. Teste a navegabilidade.





4

Aplicação de estilos com CSS3

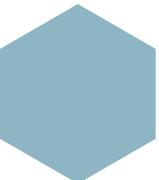
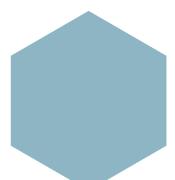


Atividade



Editora

IMPACTA



Laboratório 1

A – Definindo estilos comuns para as páginas do projeto

A partir de agora, vamos definir os estilos para as páginas do nosso projeto, iniciando pelos estilos comuns, como cores, fontes e margens. Em seguida, definiremos outros estilos para o menu na página inicial e seu conteúdo. Criaremos estilos diferentes para viabilizar futuras manutenções na aplicação.

1. Na pasta **css** do projeto, crie um arquivo chamado **estilos.css**. Inclua estes seletores:

```
* {  
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;  
}  
  
body{  
    background-color: rgb(235, 235, 216);  
}  
  
.container {  
    margin: 80px 50px 50px 50px;  
}
```

2. Configuramos todos os elementos (*) com a mesma fonte. Em todas as páginas, faça uma referência a este arquivo. Observe que as referências são diferentes para as páginas que estão em outra pasta:

- **index.html**

```
<!DOCTYPE html>  
<html>  
<head>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>Site Institucional - ABC Tecnologia</title>  
    <link rel="stylesheet" href="css/estilos.css">  
</head>
```

- **registro.html**, na pasta **paginas**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Registro de Candidatos</title>
    <link rel="stylesheet" href="../css/estilos.css">
</head>
```

3. Para contemplar a classe **container** definida no CSS, envolva o conteúdo (exceto o menu e o rodapé) em uma **div**, conforme apresentado a seguir:

```
<div class="container">
    <section>
        <article>
            <header>
                <h2>Conheça-nos um pouco</h2>
                <p>
                    Desde sua fundação a ABC Tecnologia é uma
                    empresa com foco na elaboração de soluções
                    visando agilizar os processos relativos às
                    tarefas de seus clientes. <br />
                    A agilidade, redução de custos e satisfação
                    dos nossos clientes é nossa maior meta.
                </p>
            </header>
            <p>
                Conheça nossos <a href="#">produtos e serviços</a>
                e descubra o que podemos fazer por você
            </p>
        </article>
    </section>
    <aside>
        <h2>Depoimentos dos nossos clientes</h2>
        <p>Excelente qualidade, ótimos profissionais</p>
        <p>
            Estava buscando uma solução para minha clínica
            e a ABC nos desenvolveu um sistema sob medida,
            cujos resultados superaram nossas expectativas.
        </p>
    </aside>
</div>
```

4. Quando for necessário implementar alguma alteração, esta será indicada adequadamente;

5. Execute a aplicação a partir de **index.html** e verifique o resultado.

B – Definindo estilos para o menu

1. Agora, vamos incluir um CSS para o menu. Inclua uma folha de estilos na pasta **css** com o nome **menu.css**. Defina o seguinte conteúdo:

```
nav {  
    position: fixed;  
    top: 0;  
    left: 0;  
    right: 0;  
}  
  
.menu {  
    list-style-type: none;  
    margin: 0;  
    padding: 0;  
    overflow: hidden;  
    background-color: #003264;  
}  
  
.menu li {  
    float: left;  
}  
  
.menu li a{  
    display: block;  
    color: white;  
    text-align: center;  
    padding: 14px 16px;  
    text-decoration: none;  
}  
  
.menu li a:hover {  
    background-color: #DFDFDF;  
    color: #003264;  
}
```

2. Inclua este arquivo no início de **index.html**:

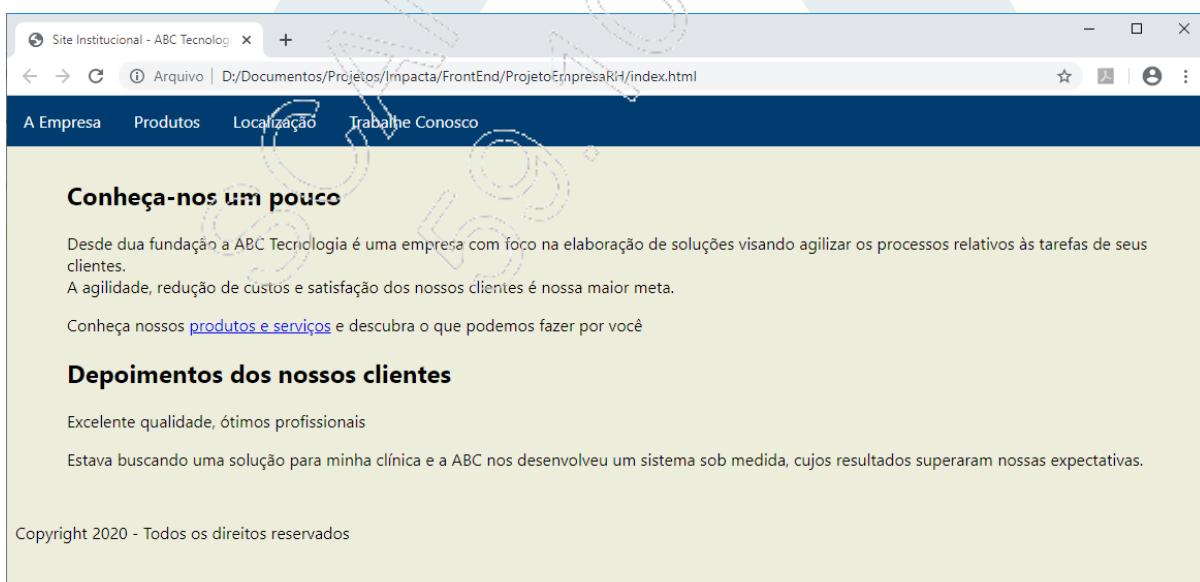
```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Site Institucional - ABC Tecnologia</title>
    <link rel="stylesheet" href="css/estilos.css">
    <link rel="stylesheet" href="css/menu.css">
</head>
```

3. Na página **index.html**, faça as alterações no trecho do elemento **<nav>**:

```
<header>
    <nav>
        <ul class="menu">
            <li><a href="#">A Empresa</a></li>
            <li><a href="#">Produtos</a></li>
            <li><a href="#">Localização</a></li>
            <li><a href="paginas/registro.html">Trabalhe Conosco</a></li>
        </ul>
    </nav>
</header>
```

4. Analise o código. Execute a página a partir de **index.html** e verifique o resultado. Até o momento, o resultado deverá ser parecido com este:

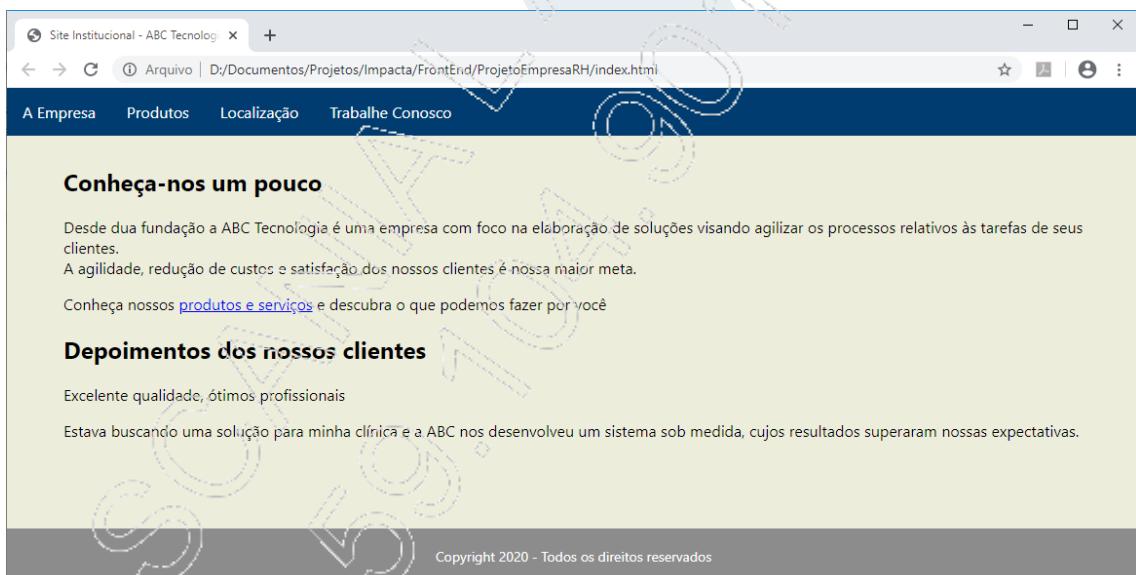


C – Definindo o rodapé e demais itens da página

1. Vamos, agora, criar o estilo para o rodapé. Este pode ser definido no próprio arquivo **estilos.css**, pois a codificação é relativamente simples. Inclua o seguinte elemento no arquivo:

```
/* estilos para o rodapé */  
footer {  
    left: 0;  
    right: 0;  
    bottom: 0;  
    position: fixed;  
    padding: 5px;  
    text-align: center;  
    font-size: 14px;  
    color: white;  
    background-color: #828282;  
}
```

2. Execute o arquivo **index.html** e visualize o rodapé inserido:



3. Agora, vamos melhorar o aspecto da nossa página inicial. Realizaremos algumas alterações tanto no CSS quanto na própria página. Comente o seletor **footer** que você definiu anteriormente;

4. No projeto, crie uma pasta chamada **imagens**;

5. Escolha duas imagens do seu gosto. Essas imagens devem ser relativamente grandes, ocupando o comprimento do seu vídeo. Elas servirão como imagem de apresentação e imagem de rodapé;

6. Inclua essas imagens na pasta **imagens** que você criou e nomeie-as como **inicio.JPG** e **rodapé.JPG**;

7. No arquivo **estilos.css**, inclua as seguintes instruções CSS:

```
.intro {  
    background: url(..../imagens/inicio.JPG) center center fixed;  
    background-size: cover;  
    padding: 220px 0;  
    margin-top: 40px;  
    left:0;  
    right:0;  
}  
  
/* estilos para o rodapé */  
  
.footer {  
    background: url(..../imagens/rodape.jpg) center center fixed;  
    background-size: cover ;  
    color: white;  
    padding: 30px;  
    text-align:center;  
}
```

8. Na página **index.html**, insira o elemento **<section>**, como mostrado a seguir:

```
<header>  
    <nav>  
        <ul class="menu">  
            <li><a href="#">A Empresa</a></li>  
            <li><a href="#">Produtos</a></li>  
            <li><a href="#">Localização</a></li>  
            <li><a href="paginas/registro.html">Trabalhe Conosco</a></li>  
        </ul>  
    </nav>  
</header>  
  
<section class="intro"></section>  
  
<div class="container">
```

9. Altere o elemento <footer>:

```
<footer class="footer">
  <p>
    Avenida Paulista, 1009 - Bela Vista <br />
    São Paulo - CEP 01311-100
  </p>
  <p>Copyright 2020 - Todos os direitos reservados</p>
</footer>
```

10. Execute a página e veja o resultado.

D – Definindo estilos para os formulários

Vamos, agora, incluir alguns estilos para os formulários. Os estilos mais importantes utilizarão as classes do **Bootstrap**, em tópicos futuros.

1. No arquivo **estilos.css**, inclua estes estilos:

```
.borda {
  box-shadow: 5px 5px 10px #000;
  padding: 30px;
  margin-right: 30px;
  background-color: white;
}
```

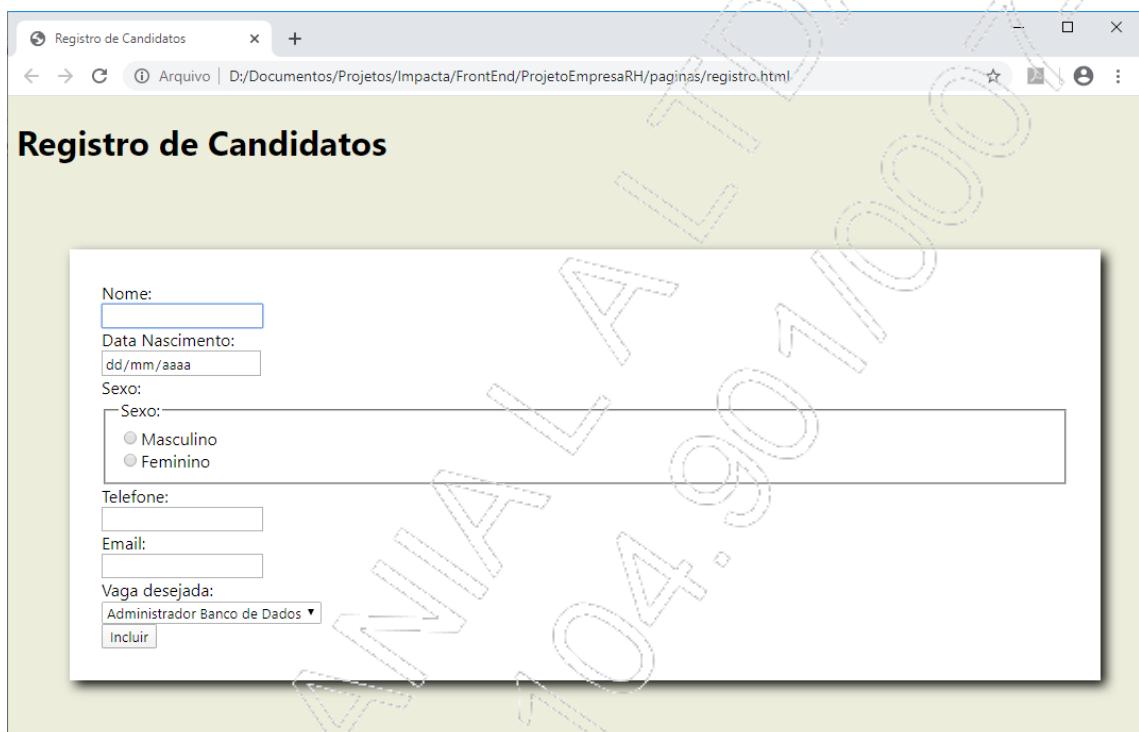
2. No arquivo **registro.html**, envolva o conteúdo do elemento **form** com uma **div**:

```
<form>
  <div>
    <!--Nome do candidato -->
    ...
    <!--Botão de comandos -->
  <div>
    <input type="button" value="Incluir">
  </div>
</div>
</form>
```

3. Configure esse novo estilo no elemento **div** adicionado ao formulário do arquivo **registro.html**. Verifique os pontos onde os estilos estão sendo aplicados:

```
<form>
  <div class="container borda">
    ...
  </div>
</form>
```

4. A página **registro.html** deverá ficar semelhante à imagem adiante:



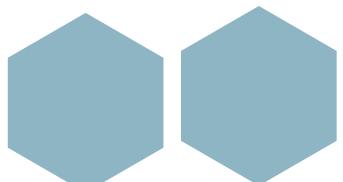


5

Desenvolvimento de responsividade com media queries



Atividade



Laboratório 1

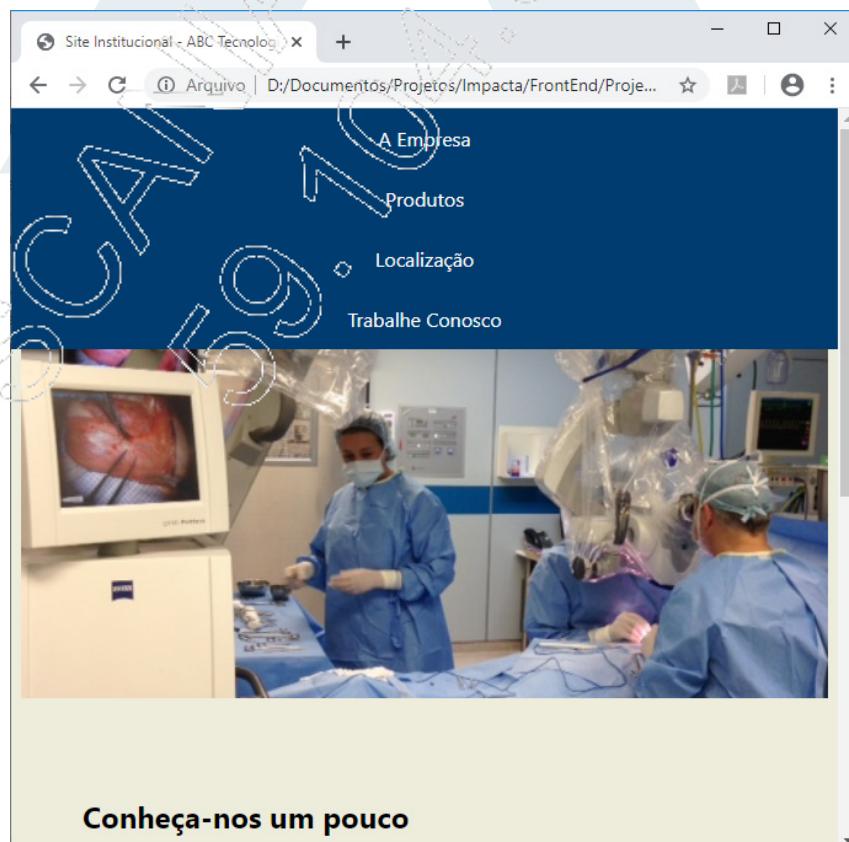
A – Definindo a responsividade para o menu

Na página **index.html**, o menu de opções não possui nenhuma responsividade, ou seja, se executarmos a aplicação em dispositivos menores, como tablets ou smartphones, a aparência é exatamente a mesma. Neste exercício, incluiremos os recursos no CSS para permitir certa variação e simplificação no layout do menu para dispositivos menores.

1. Abra o arquivo **menu.css** e inclua os seguintes elementos:

```
@media screen and (max-width: 760px) {  
    .menu li{  
        float: none;  
    }  
  
    .menu li a {  
        text-align: center;  
    }  
}
```

Para telas com o tamanho máximo de 760 pixels, colocamos o menu na posição vertical. A aparência da aplicação fica como a imagem a seguir:



2. Agora, vamos adicionar um pouco mais de interatividade na nossa página. Inclua, na pasta **css**, uma folha de estilos chamada **destaque.css**. Esse arquivo definirá estilos para incluir um ícone circular na página inicial, que também direcionará o usuário para a página **registro.html**. Nessa folha de estilos, inclua o seguinte conteúdo:

```
/* para o destaque do menu */
.registro {
    display: block;
    position: absolute;
    top: 400px;
    right: 35px;
    width: 160px;
    height: 100px;
    padding-top: 60px;
    opacity: 0.8;
    font-size: 1rem;
    color: #fff;
    text-align: center;
    text-decoration: none;
    text-transform: uppercase;

    /* Criação do círculo */
    border-radius: 100%;
    text-shadow: 0 1px 0 #000;

    /* Efeitos de rotação do texto no círculo */
    transform: rotate(6deg);

    /* Efeitos de Gradiente */
    background: -ms-linear-gradient(top, #a80000 0%,#740404 100%);
    background: -webkit-linear-gradient(top, #a80000 0%,#740404 100%);
    background: -moz-linear-gradient(top, #a80000 0%,#740404 100%);
    background: linear-gradient(top, #a80000 0%,#740404 100%);
    transition: transform 1s;
}

.registro:hover {
    color: #fff;

    /* Efeitos do Gradiente ao mover o mouse */
    background: linear-gradient(top, #bc0101 0%,#8c0909 100%);
    transform: rotate(16deg) scale(1.1,1.1);
    transition: transform 1s;
}
```

```
.registro:before /* Cria a borda tracejada */ {  
    display: block;  
    position: absolute;  
    top: -7px;  
    right: -7px;  
    height: 168px;  
    width: 168px;  
    content: "";  
    border: 3px dotted #740404;  
  
    /* Cantos circulares do círculo */  
    border-radius: 100%;  
}  
  
.registro .free{  
    font-size: 80%;  
}  
  
@media screen and (max-width: 760px) {  
    .registro {  
        display:none;  
    }  
}
```

3. Inclua o arquivo **destaque.css** em **index.html**:

```
<!DOCTYPE html>  
<html>  
  
<head>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>Site Institucional - ABC Tecnologia</title>  
    <link rel="stylesheet" href="css/estilos.css" />  
    <link rel="stylesheet" href="css/menu.css" />  
    <link rel="stylesheet" href="css/destaque.css">  
</head>
```

Desenvolvimento de responsividade com media queries

4. Na página **index.html**, inclua o elemento **<div>** indicado, dentro do elemento **<header>** existente:

```
<header>
  <div>
    <a class="registro" href="paginas/registro.html">
      Candidate-se<br/>
      <span class="free">• Hoje •</span>
    </a>
  </div>
  <h2>Conheça-nos um pouco</h2>
  <p>
    Desde sua fundação a ABC Tecnologia é uma
    empresa com foco na elaboração de soluções
    visando agilizar os processos relativos às
    tarefas de seus clientes. <br />
    A agilidade, redução de custos e satisfação
    dos nossos clientes é nossa maior meta.
  </p>
</header>
```

5. Execute a página e verifique o resultado.



6

Inclusão de funcionalidades JavaScript e jQuery



Atividade



Laboratório 1

A – Definindo mensagens personalizadas de validação para os formulários

Com JavaScript, podemos incluir mais interatividade com o usuário. Nesta parte do projeto, incluiremos um evento baseado em JavaScript para interceptar o evento **click** do botão **Incluir**, de modo a personalizar as mensagens de validação.

1. No arquivo **registro.html**, inclua um id no botão **Incluir**. Momentaneamente, altere o valor do atributo **type** para '**submit**', de modo que seja possível visualizar as mensagens de validação:

```
<div>
  <input type="submit" value="Incluir" id="incluirButton">
</div>
```

2. Na pasta **scripts**, inclua um arquivo chamado **validacao_registro.js** com o seguinte conteúdo:

```
let botao = document.getElementById('incluirButton');

botao.addEventListener('click', function (e) {

  let nome = document.getElementById('nome');
  let data = document.getElementById('data');
  let telefone = document.getElementById('telefone');
  let email = document.getElementById('email');

  nome.setCustomValidity(nome.value == '' ?
    'Informe corretamente o nome do candidato' : '');
  data.setCustomValidity(data.value == '' ?
    'Informe corretamente a data de nascimento do candidato' : '');
  telefone.setCustomValidity(telefone.value == '' ?
    'Informe corretamente o telefone do candidato' : '');
  email.setCustomValidity(email.value == '' ?
    'Informe corretamente o email do candidato' : '');

});
```

3. Inclua esse arquivo no final da página, como última instrução (é importante esta posição, pois as instruções presentes no arquivo fazem referência aos elementos HTML e serão validadas quando o DOM estiver completo):

```
</form>
</div>
<script src="../scripts/validacao_registro.js"></script>
</body>
```

4. Execute a página **registro.html**. Clique no botão **Incluir**, sem fornecer informações nos campos **nome**, **data**, **telefone** ou **email**. O que você consegue visualizar?

5. Retorne o valor do atributo **value** do botão para '**button**'.

B – Melhorando a responsividade do menu

O nosso menu de opções na página **index.html** já está responsivo, mas falta um detalhe: quando ele é exibido em dispositivos menores, o menu cobre toda a tela. Nossa proposta é justamente criar um recurso para que os elementos do menu se contraiam e, por meio do clique em um ícone a ser adicionado, ele mostre as opções.

1. Altere o arquivo **menu.css** onde há a indicação de **@media screen**. O conteúdo anterior permanece no arquivo, devidamente comentado para sua orientação:

```
@media screen and (max-width: 760px) {  
    /* .menu li{  
        float: none;  
    }  
  
    .menu li a {  
        text-align: center;  
    } */  
  
    .menu li:not(:first-child) {  
        display: none;  
    }  
  
    .menu li.icon {  
        float: right;  
        display: block;  
    }  
  
.menu.responsive {  
    position: relative;  
}  
  
.menu.responsive li.icon {  
    position: absolute;  
    right: 0;  
    top: 0;  
}
```

```
.menu.responsive li {  
    float: none;  
    display: inline;  
}  
  
.menu.responsive li a {  
    display: block;  
    text-align: left;  
}  
  
}
```

2. Na página **index.html**, realize estas alterações, na parte que define o menu:

```
<nav>  
    <ul class="menu" id="menuTopo">  
        <li><a href="#">A Empresa</a></li>  
        <li><a href="#">Produtos</a></li>  
        <li><a href="#">Localização</a></li>  
        <li><a href="paginas/registro.html">Trabalhe Conosco</a></li>  
  
        <li class="icon">  
            <a href="javascript:void(0);" onclick="mostrarMenu()">  
                &#9776;</a>  
        </li>  
    </ul>  
</nav>
```

3. No final da página, antes do fechamento **</body>**, inclua este código JavaScript:

```
<script type="text/javascript">  
    function mostrarMenu(){  
        let elemento = document.getElementById("menuTopo");  
        if(elemento.className === "menu"){  
            elemento.className += " responsive";  
        } else {  
            elemento.className = "menu";  
        }  
    }  
</script>  
</body>
```

4. Ainda no arquivo **menu.css**, inclua este estilo antes do item **@media screen**:

```
.menu li.icon {  
    display: none;  
}  
  
@media screen and (max-width: 760px) {
```

5. Execute a página, reduza o tamanho e veja o resultado.

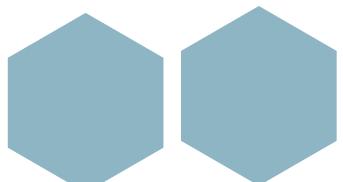


7

Consumo de serviços REST



Atividade



Laboratório 1

A – Mostrando a lista de vagas dinamicamente

Em aplicações Web, é bastante comum que determinadas informações sejam obtidas de fontes externas. Para que seja possível obtê-las, devemos usar recursos que viabilizem essa tarefa. Os Web services são os recursos mais comuns para essa finalidade.

A criação e a manutenção de Web services não são tarefa do desenvolvedor front end, porém seu consumo é, uma vez que o JavaScript nos permite consumi-los por meio de funções, como é o caso da função `fetch`.

Neste projeto, apresentaremos recursos para o consumo de Web services padrão REST. O problema é que não temos o Web service disponível!

Para contornar esse problema, utilizaremos um componente do Node.js chamado `json-server`. Esse componente permite a simulação de um Web service, tomando como base informações armazenadas em arquivos (como se estes fossem, de fato, os elementos contendo os dados para nossa aplicação). Iniciaremos com a instalação desse módulo e, na sequência, criaremos um arquivo representando os dados a serem consumidos. A lista de vagas apresentada no arquivo `registro.html` será obtida por meio deste serviço.

1. No prompt de comandos, instale o módulo `json-server` usando a ferramenta `npm` disponibilizada com o Node.js:

```
npm install -g json-server
```

A opção `-g` indica instalação global (para todos os usuários).

2. Crie uma pasta chamada `dados` no projeto;

3. Na pasta **dados**, crie um arquivo chamado **dados.json** com o seguinte conteúdo:

```
{  
    "vagas": [  
        {  
            "id": 1,  
            "titulo": "Programador Java Pleno",  
            "tipo": "1"  
        },  
        {  
            "id": 2,  
            "titulo": "Analista Comercial",  
            "tipo": "2"  
        },  
        {  
            "id": 3,  
            "titulo": "Instrutor Excel Avançado",  
            "tipo": "1"  
        },  
        {  
            "id": 4,  
            "titulo": "Gerente de Projetos para .NET",  
            "tipo": "2"  
        },  
        {  
            "id": 5,  
            "titulo": "Consultor de Desenvolvimento Agil",  
            "tipo": "2"  
        },  
        {  
            "id": 6,  
            "titulo": "Programador VB.Net",  
            "tipo": "1"  
        }  
    ]  
}
```

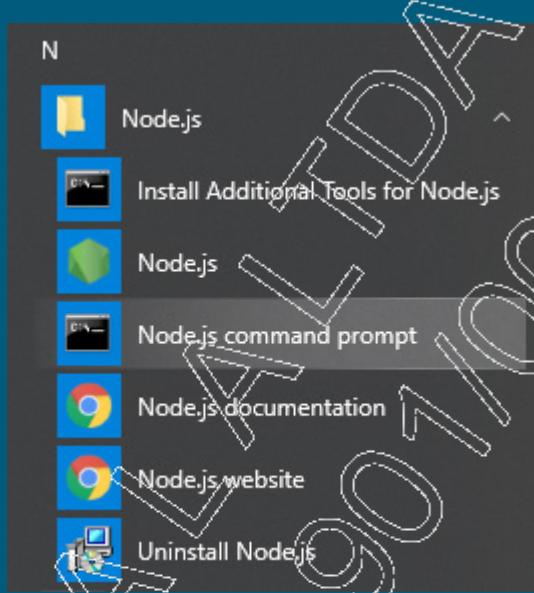
4. Usando o prompt de comandos, entre na pasta em que você criou o arquivo **dados.json** (pasta **dados**);

5. Execute este comando:

```
json-server dados.json
```

Observação:

Dependendo do nível de acesso do usuário, pode ser necessário executar o prompt do próprio Node.js (**Node.js command prompt**), como mostrado na ilustração adiante:



O importante é que o acesso à execução do módulo **json-server** esteja disponível.

6. A execução do comando anterior deverá nos fornecer um acesso semelhante ao mostrado a seguir:

```
cmd C:\Windows\System32\cmd.exe - json-server dados.json
D:\Documentos\Projetos\Impacta\FrontEnd\ProjetoEmpresaRH\dados>json-server dados.json
\{^_^\}/ hit
Loading dados.json
Done

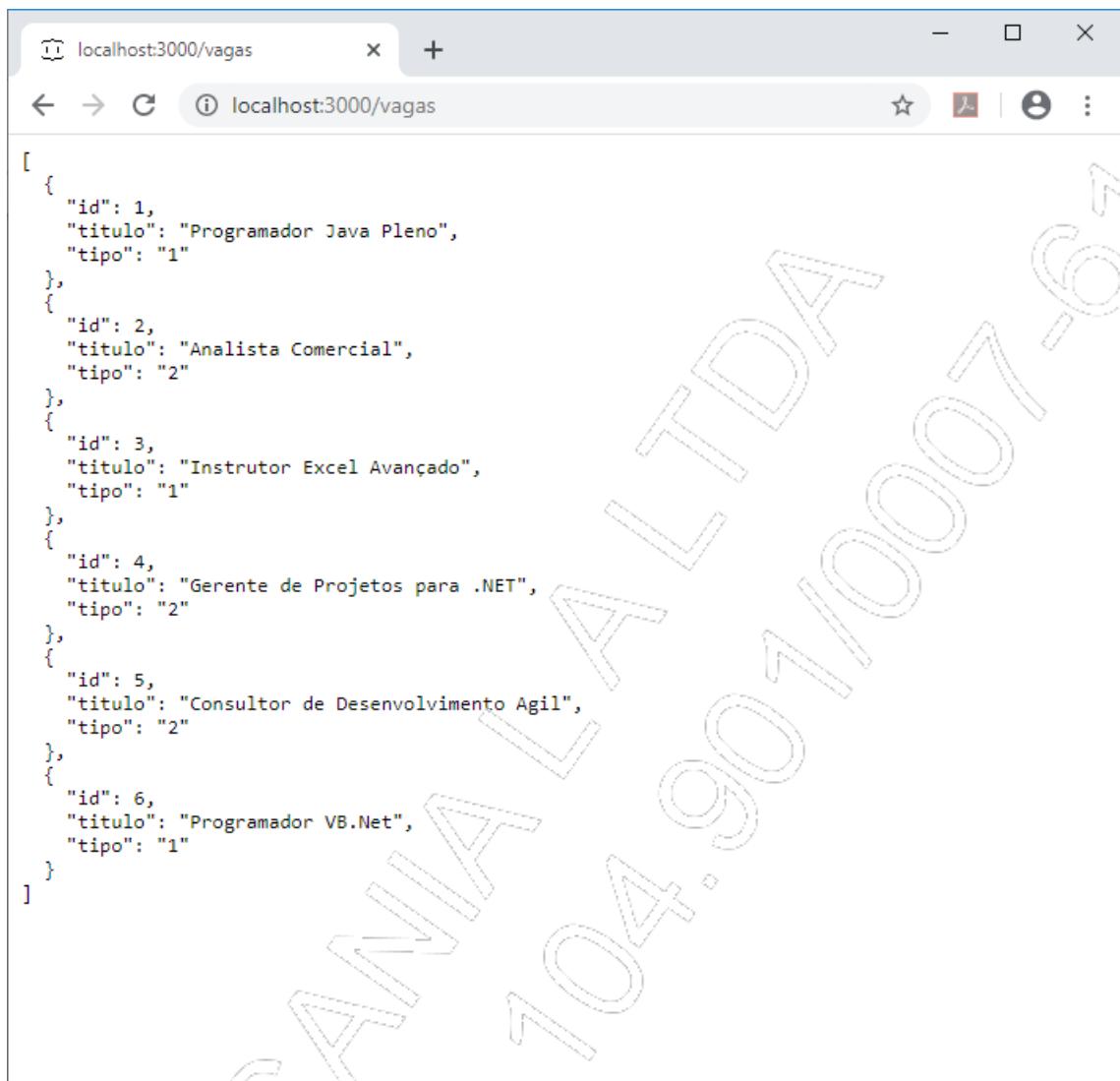
Resources
http://localhost:3000/vagas

Home
http://localhost:3000

Type s + enter at any time to create a snapshot of the database
```

A screenshot of a Windows Command Prompt window. The title bar says 'cmd C:\Windows\System32\cmd.exe - json-server dados.json'. The command 'json-server dados.json' is entered and executed. The output shows the server loading the 'dados.json' file from the current directory and starting a local development server at 'http://localhost:3000'. It also provides a keybinding for creating a database snapshot.

7. Observe que o json-server nos apresentou uma URL chamada `http://localhost:3000/vagas`. Acesse essa URL e veja o resultado:



The screenshot shows a web browser window with the address bar containing "localhost:3000/vagas". The page content displays a JSON array of six job posts:

```
[{"id": 1, "titulo": "Programador Java Pleno", "tipo": "1"}, {"id": 2, "titulo": "Analista Comercial", "tipo": "2"}, {"id": 3, "titulo": "Instrutor Excel Avançado", "tipo": "1"}, {"id": 4, "titulo": "Gerente de Projetos para .NET", "tipo": "2"}, {"id": 5, "titulo": "Consultor de Desenvolvimento Agil", "tipo": "2"}, {"id": 6, "titulo": "Programador VB.Net", "tipo": "1"}]
```

Esse é nosso Web service simulado! Nós o utilizaremos na sequência do projeto.

8. Na página `registro.html`, remova as vagas do elemento `<select>`. Ele deve ficar assim:

```
<!-- Lista de vagas -->
<div>
  <label for="vaga">Vaga desejada:</label>
  <div>
    <select name="vaga" id="vaga">
      </select>
    </div>
  </div>
```

9. Crie, na pasta **scripts**, um novo arquivo JavaScript chamado **listaVagas.js**. Seu conteúdo deve ser o mostrado adiante:

```
//lista de vagas, a ser preenchida com os
//dados do Web service
let vagas = [];

//URL obtida a partir do componente json-server
let url = "http://localhost:3000/vagas";
let lista = document.getElementById("vaga");

//função para construir a lista de vagas na página
function exibirVagas(){
    for (let i = 0; i < vagas.length; i++) {
        let option = document.createElement("option");
        option.textContent = vagas[i].titulo;
        option.setAttribute("value", vagas[i].id);

        lista.appendChild(option);
    }
}

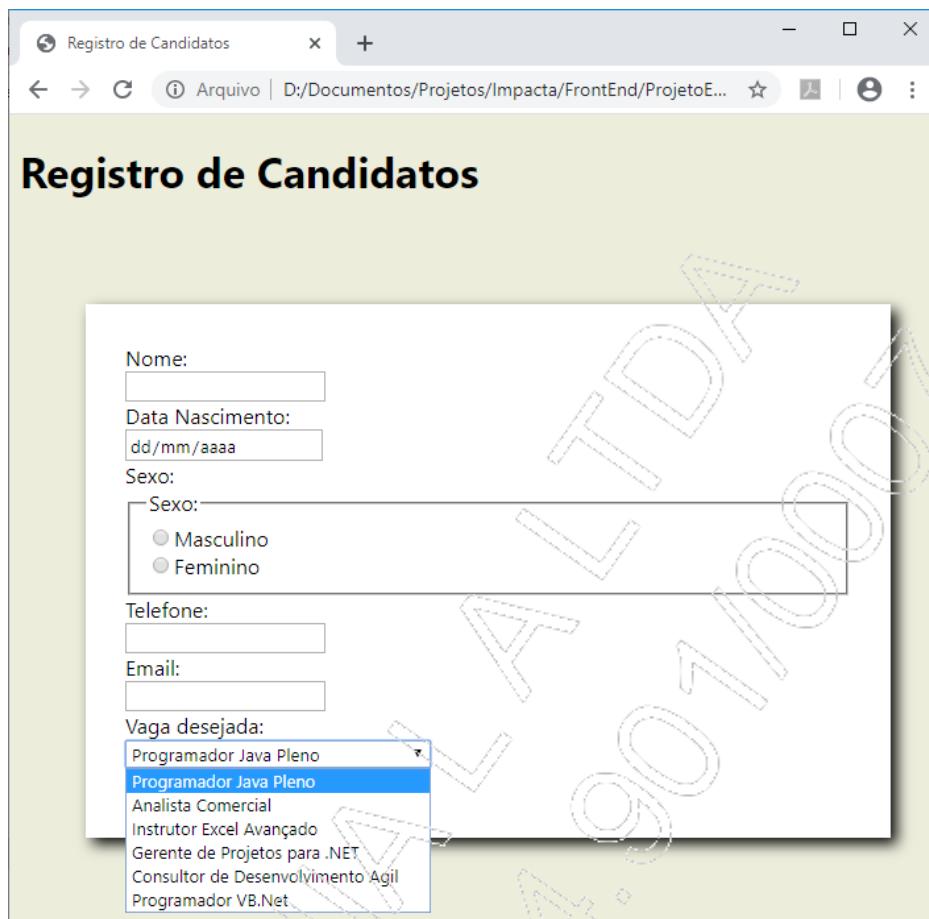
//acesso ao Web service
fetch(url)
    .then(res => res.json())
    .then(valor => {
        vagas = valor;
        exibirVagas();
    });
}
```

10. Inclua esse arquivo ao final da página, antes do fechamento **</body>**:

```
</form>
</div>
<script src="../scripts/validacao_registro.js"></script>
<script src="../scripts/listaVagas.js"></script>
</body>

</html>
```

11. Execute a página e verifique se a lista de vagas aparece de acordo com o código implementado no arquivo:



12. Agora, vamos incluir uma opção para o usuário selecionar as vagas por tipo. Na página **registro.html**, inclua o grupo de elementos imediatamente antes da lista de vagas:

```
<!-- Lista de vagas -->
<div>
  <label>Tipo de vaga:</label>
  <div>
    <input type="radio" id="desenv" name="tipo"
           value="desenv" checked="checked">Desenvolvimento
    <input type="radio" id="negocio" name="tipo"
           value="negocio">Negócios
  </div>
</div>

<div>
  <label for="vaga">Vaga desejada:</label>
  <div>
    <select name="vaga" id="vaga">
      </select>
  </div>
</div>
```

13. No arquivo **listaVagas.js**, acrescente a função **limparLista**:

```
//função para limpar a lista de vagas
function limparLista(){
    while(lista.firstChild){
        lista.removeChild(lista.firstChild);
    }
}
```

14. Realize as seguintes alterações na função **exibirVagas**:

```
//função para construir a lista de vagas na página
function exibirVagas(){

    let desenv = document.getElementById("desenv");
    let negocio = document.getElementById("negocio");

    limparLista();

    for (let i = 0; i < vagas.length; i++) {

        let tipo = vagas[i].tipo;
        var qualTipo = (desenv.checked && tipo == '1') ||
            (negocio.checked && tipo == '2');

        if(qualTipo){
            let option = document.createElement("option");
            option.textContent = vagas[i].titulo;
            option.setAttribute("value", vagas[i].id);

            lista.appendChild(option);
        }
    }
}
```

15. No final, adicione as instruções para o evento **click** dos elementos **checkbox**:

```
desenv.addEventListener("click", exibirVagas, false);
negocio.addEventListener("click", exibirVagas, false);
```

16. Execute a página e veja o resultado obtido ao selecionar o tipo de vaga.



8

Otimizações de layout com Bootstrap



Atividade



Laboratório 1

Para usarmos o Bootstrap, temos alguns caminhos:

- Fazendo o download dos arquivos no portal <getbootstrap.com>;
- Adicionando o conteúdo do Bootstrap por meio do npm, ferramenta disponível com o Node.js. Este procedimento requer que o projeto seja configurado com o padrão requerido pelo próprio Node.js;
- Usando a versão on-line, ou CDN (Content Delivery Network). A vantagem desta abordagem é que não precisamos de arquivos no projeto, além de permitir a mudança de versão, caso seja necessário. A desvantagem é que a execução requer uma conexão com a Internet.

A escolha pela melhor alternativa é do desenvolvedor. Neste projeto, usaremos a versão CDN. O Bootstrap está na versão 4.4 durante a elaboração deste material, mas você pode escolher a versão que julgar adequada, lembrando-se de manter a documentação sempre à disposição.

A – Incluindo estilos na página de registro baseados no Bootstrap

Nosso formulário de registros utilizou algumas classes CSS, mas não aplicamos nenhuma classe ao formulário propriamente dito, ou seja, aos campos do formulário. Vamos tomar como base a documentação do Bootstrap no link <<https://getbootstrap.com/docs/4.4/components/forms/>>.

1. Abra o arquivo **registro.html**:

2. No elemento **<head>**, antes da referência ao arquivo **estilos.css**, inclua a referência ao Bootstrap (essa referência foi copiada de <<https://getbootstrap.com/docs/4.4/getting-started/download/>>):

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
        content="width=device-width, initial-scale=1.0">
  <title>Registro de Candidatos</title>
  <link rel="stylesheet" href=
"https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" />
  <link rel="stylesheet" href="..../css/estilos.css">
</head>
```

Observe que removemos os atributos **integrity** e **crossorigin**.

Otimizações de layout com Bootstrap

3. Vamos adicionar classes Bootstrap nos elementos `<div>` que envolvem cada grupo de campos de entrada, nos elementos `<label>`, nos elementos `<div>` específicos dos campos de entrada e nos próprios campos de entrada. As alterações estão destacadas a seguir:

```
<form>
  <div class="container borda">
    <!-- Nome do candidato -->
    <div class="form-group row">
      <label for="nome"
            class="col-sm-2 col-form-label">Nome:</label>
      <div class="col-sm-10">
        <input type="text" id="nome" name="nome"
              required="required" autofocus="autofocus"
              class="form-control" />
      </div>
    </div>
  </div>

  <!-- Data de nascimento -->
  <div class="form-group row">
    <label for="data"
          class="col-sm-2 col-form-label">
      Data Nascimento:</label>
    <div class="col-sm-5">
      <input type="date" id="data" name="data"
            required="required"
            class="form-control" />
    </div>
  </div>

  <!-- Sexo -->
  <div class="form-group row">
    <label for="sexo"
          class="col-sm-2 col-form-label">Sexo:</label>
    <div class="col-sm-10">
      <fieldset>
        <legend>Sexo:</legend>
        <input type="radio" id="masculino" name="sexo"
              value="masculino">Masculino<br />
        <input type="radio" id="feminino" name="sexo"
              value="feminino">Feminino<br />
      </fieldset>
    </div>
  </div>
</form>
```

```
<!-- Telefone -->
<div class="form-group row">
    <label for="telefone"
        class="col-sm-2 col-form-label">Telefone:</label>
    <div class="col-sm-5">
        <input type="tel" id="telefone" name="telefone"
            required="required"
            class="form-control" />
    </div>
</div>

<!-- E-mail -->
<div class="form-group row">
    <label for="email"
        class="col-sm-2 col-form-label">Email:</label>
    <div class="col-sm-10">
        <input type="email" id="email" name="email"
            required="required"
            class="form-control" />
    </div>
</div>

<!-- Lista de vagas -->
<div class="form-group row">
    <label class="col-sm-2 col-form-label">Tipo de vaga:</label>
    <div class="col-sm-5">
        <input type="radio" id="desenv" name="tipo"
            value="desenv" checked="checked">Desenvolvimento
        <input type="radio" id="negocio" name="tipo"
            value="negocio">Negócios
    </div>
</div>

<div class="form-group row">
    <label for="vaga"
        class="col-sm-2 col-form-label">Vaga desejada:</label>
    <div class="col-sm-5">
        <select name="vaga" id="vaga"
            class="form-control">
        </select>
    </div>
</div>

<!-- Botão de comandos -->
<div class="form-group row">
    <div class="col-sm-5 col-sm-offset-2">
        <input type="button" value="Incluir"
            id="incluirButton"
            class="btn btn-primary" />
    </div>
</div>
</div>
</form>
```

Otimizações de layout com Bootstrap

4. O resultado da aplicação dessas classes produz o resultado adiante:

B – Definindo a página de produtos com Bootstrap

Vamos, agora, elaborar a página de apresentação de produtos da empresa utilizando os recursos do Bootstrap. Vamos desenvolver a página de forma que as informações relevantes estejam acessíveis na mesma página.

1. Crie, na pasta **paginas**, o arquivo **produtos.html**;

2. Adicione uma estrutura de arquivo HTML5:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1.0">
  <title>Apresentação dos produtos</title>
</head>
<body>

</body>
</html>
```

3. Inclua as referências ao Bootstrap como mostrado adiante:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
        content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet"
        href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/
bootstrap.min.css" />

        <title>Apresentação dos produtos</title>
</head>
<body>

    <script src="http://code.jquery.com/jquery-3.4.1.slim.min.js">
    </script>
    <script src=
"https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js">
    </script>
    <script src=
"https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"></
script>

</body>
</html>
```

Veja que incluímos, também, uma referência ao jQuery. As bibliotecas JavaScript são necessárias para executarmos eventos na página. No próximo tópico, adicionaremos um menu de opções, responsivo. A responsividade baseada no Bootstrap usa funções JavaScript previamente definidas na biblioteca, que, por sua vez, utilizam o jQuery.

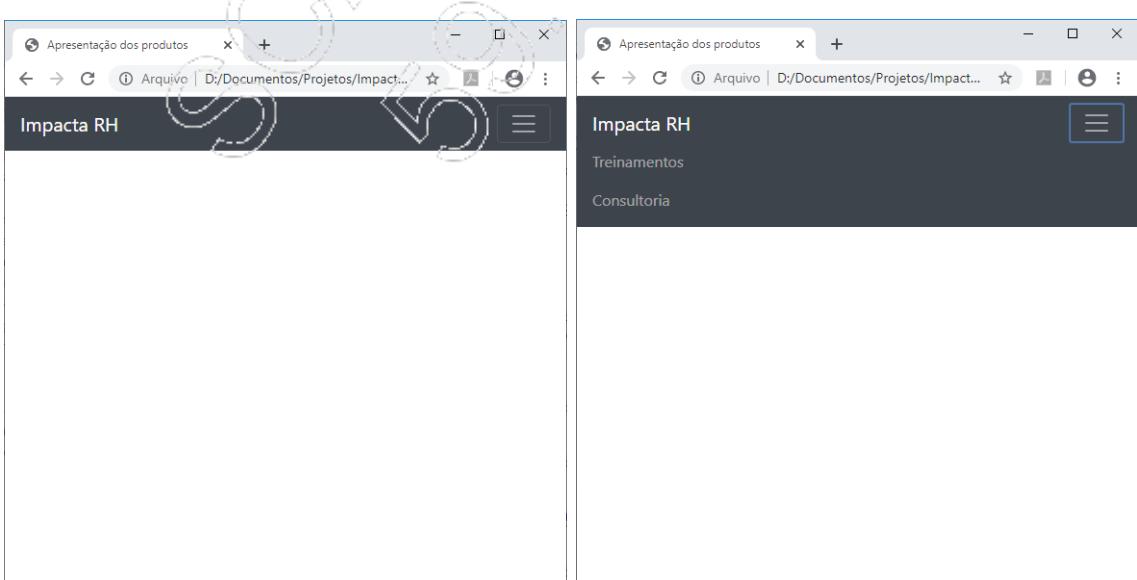
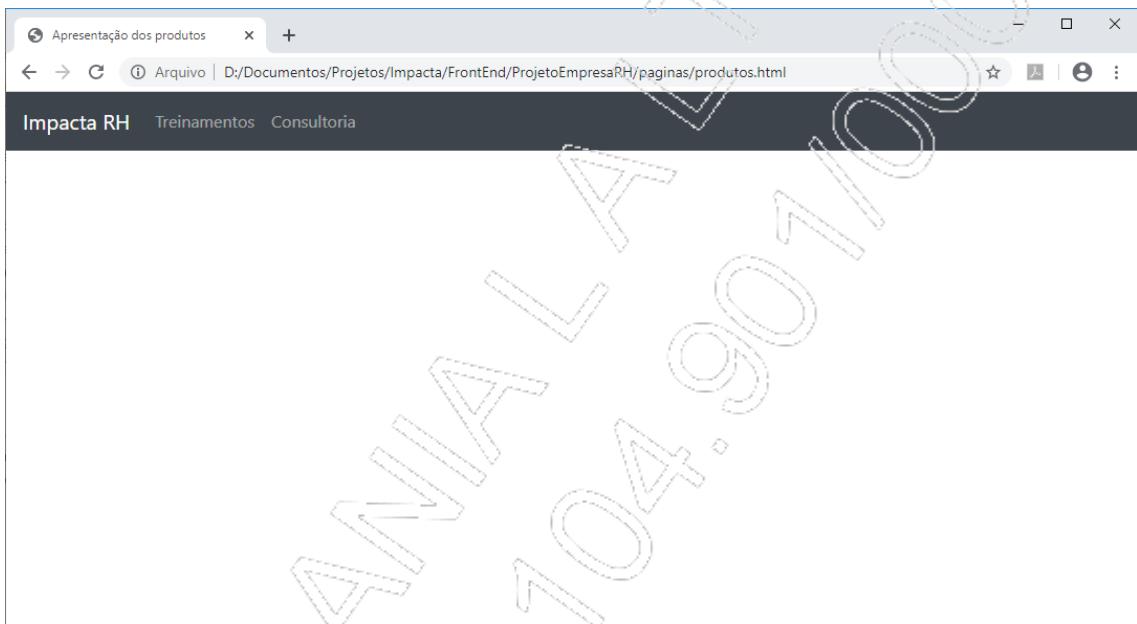
4. Vamos incluir um menu de opções. Logo abaixo de **<body>**, inclua este código (observe o uso dos elementos Bootstrap):

```
<!-- Menu de navegação -->
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <a class="navbar-brand" href="#">Impacta RH</a>
    <button class="navbar-toggler" type="button"
        data-toggle="collapse"
        data-target="#opcoes">
        <span class="navbar-toggler-icon"></span>
    </button>
```

Otimizações de layout com Bootstrap

```
<div class="collapse navbar-collapse" id="opcoes">
  <ul class="navbar-nav mr-auto">
    <li class="nav-item">
      <a class="nav-link" href="#treinamentos">Treinamentos</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#consultoria">Consultoria</a>
    </li>
  </ul>
</div>
</nav>
```

5. Teste a página, tanto em tamanho normal como em tamanho reduzido. Observe a responsividade do menu:



6. Vamos desenvolver duas seções: **treinamento** e **desenvolvimento**. Começando pela seção **treinamento**, serão necessárias três imagens retangulares, com dimensões 150x150 pixels. Essas imagens serão usadas para representar as modalidades de treinamento. Inclua essas imagens na pasta **imagens**. Nomeie essas imagens como: **curso-presencial.jpg**, **curso-online.jpg** e **curso-incompany.jpg**, respectivamente. Procure selecionar imagens condizentes com os nomes. Em seguida, insira a <div> logo abaixo do menu de navegação:

```
<!-- Conteúdo -->
<div class="container body-content clearfix">
</div>
```

7. Inclua um arquivo chamado **produtos.css** na pasta **css**. Após incluir o conteúdo seguinte, adicione a referência a este arquivo na página:

```
.body-content {
  clear:both;
  padding: 15px;
  margin-top: 50px;
}

.center-image {
  text-align:center;
}

.margin-image{
  margin: 10px;
}

.footer{
  bottom: 0;
  width: 100%;
  text-align:center;
  padding:20px;
  background-color:#7F7F7F;
  color: white;
}
```

8. Na div que você inseriu por último, inclua um elemento <section> com o seguinte conteúdo:

```
<!-- #treinamentos -->
<section id="treinamentos">
  <h1>Treinamentos</h1>
  <p>
    A Empresa oferece treinamentos em diversas
    áreas, nas seguintes modalidades:
  </p>
  <ul>
    <li>Presencial</li>
    <li>Online</li>
    <li><i>In-company</i></li>
  </ul>

  <div class="row center-image">
    <div class="col-md-4">
      <h3>Presencial</h3>
      
      <p>
        Nesta modalidade os cursos são oferecidos
        em nossa unidade localizada na Av. Paulista.
        Existem diversas opções de horários.<br />
        Consultar a grade de horários para maiores
        informações
      </p>
    </div>
    <div class="col-md-4">
      <h3>Online</h3>
      
      <p>
        Não tem tempo de ir até nossa unidade?
        Necessita estudar, e possui horários diversificados?
        Temos treinamentos também na modalidade
        <i>online</i>.<br />
        Assim, você estuda na hora e nas datas que
        estiver disponível.
      </p>
    </div>
    <div class="col-md-4">
      <h3><i>In-company</i></h3>
      
      <p>
        Se sua empresa necessita de treinamentos personalizados,
        atendendo a necessidades específicas, temos uma equipe
        altamente capacitada para ir até sua empresa e levar
        o conteúdo que você precisa.
      </p>
    </div>
  </div>
</section>
```

9. Até este ponto, o resultado deve ser similar a este:

The screenshot shows a web browser window with the title bar "Apresentação dos produtos". The address bar shows the URL "D:/Documentos/Projetos/Impacta/FrontEnd/ProjetoEmpresaRH/paginas/produtos.html". The page content is as follows:

Treinamentos

A Empresa oferece treinamentos em diversas áreas, nas seguintes modalidades:

- Presencial
- Online
- *In-company*

Presencial

Nesta modalidade os cursos são oferecidos em nossa unidade localizada na Av. Paulista. Existem diversas opções de horários.
Consultar a grade de horários para maiores informações

Online

Não tem tempo de ir até nossa unidade? Necessita estudar, e possui horários diversificados? Temos treinamentos também na modalidade *online*. Assim, você estuda na hora e nas datas que estiver disponível.

In-company

Se sua empresa necessita de treinamentos personalizados, atendendo a necessidades específicas, temos uma equipe altamente capacitada para ir até sua empresa e levar o conteúdo que você precisa.

10. Abaixo dessa **section**, fora da div incluída no item 6, acrescente outro elemento, **section consultoria**. Aqui, será necessária outra imagem, como mostrado no código a seguir:

```
</section>
</div>

<!-- #consultoria -->
<section id="consultoria" style="background-color:burlywood;">
  <div class="container body-content clearfix">
    <h1>Consultoria</h1>
    <div class="row">
      <div class="col-md-4">
        
      </div>
```

```

<div class="col-md-8">
  <p>
    De modo geral, toda empresa deseja obter lucros,
    a custos reduzidos.
    No que diz respeito ao gerenciamento de
    processos, temos consultores
    especializados para ajudar nossos clientes
    a atingirem esse objetivo.
    Como? Aplicando da melhor forma possível
    os recursos disponíveis,
    como infraestrutura, pessoas, sistemas,
    folhas de pagamento,
    entre outros.<br />
    Neste sentido a empresa colabora para
    que o principal objetivo do
    negócio seja consolidado, dispensando
    seus clientes de ocupações irrelevantes.
  </p>
</div>
</div>

</div>
</div>

</section>

```

11. O resultado é o mostrado a seguir:

Presencial



Nesta modalidade os cursos são oferecidos em nossa unidade localizada na Av. Paulista. Existem diversas opções de horários. Consultar a grade de horários para maiores informações

Online



Não tem tempo de ir até nossa unidade? Necessita estudar, e possui horários diversificados? Temos treinamentos também na modalidade *online*. Assim, você estuda na hora e nas datas que estiver disponível.

In-company



Se sua empresa necessita de treinamentos personalizados, atendendo a necessidades específicas, temos uma equipe altamente capacitada para ir até sua empresa e levar o conteúdo que você precisa.

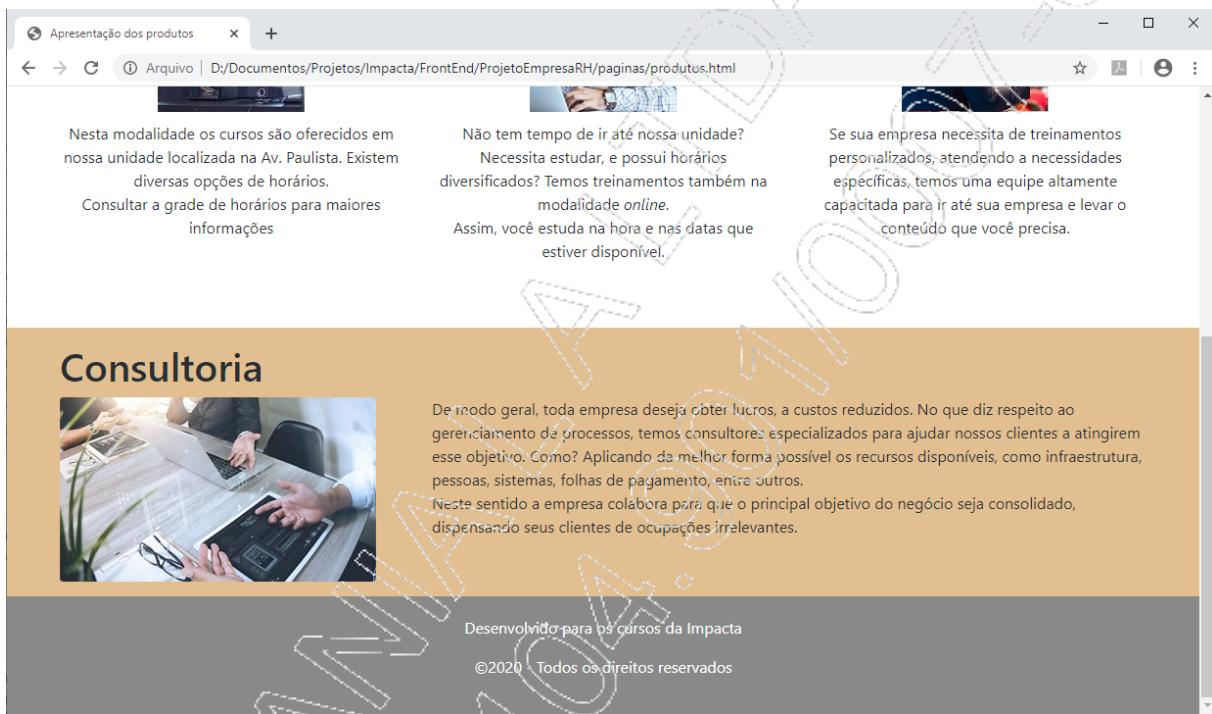
Consultoria



De modo geral, toda empresa deseja obter lucros, a custos reduzidos. No que diz respeito ao gerenciamento de processos, temos consultores especializados para ajudar nossos clientes a atingirem esse objetivo. Como? Aplicando da melhor forma possível os recursos disponíveis, como infraestrutura, pessoas, sistemas, folhas de pagamento, entre outros. Neste sentido a empresa colabora para que o principal objetivo do negócio seja consolidado, dispensando seus clientes de ocupações irrelevantes.

12. Defina, agora, o rodapé da página, após o elemento `<section>` recém-incluído:

```
<footer class="footer">
  <p>
    Desenvolvido para os cursos da Impacta
  </p>
  <p>
    ©2020 - Todos os direitos reservados
  </p>
</footer>
```



13. No arquivo `index.html`, atualize o menu de opções para incluir a página `produtos.html`:

```
<nav>
  <ul class="menu" id="menuTopo">
    <li><a href="#">A Empresa</a></li>
    <li><a href="paginas/produtos.html">Produtos</a></li>
    <li><a href="#">Localização</a></li>
    <li><a href="paginas/registro.html">Trabalhe Conosco</a></li>

    <li class="icon">
      <a href="javascript:void(0);"
         onclick="mostrarMenu()">#9776;</a>
    </li>
  </ul>
</nav>
```

14. Execute a página e verifique seu conteúdo.

C – Definindo a página de localização

1. Inclua, na pasta **paginas**, um arquivo chamado **localizacao.html**;

2. Insira a referência ao Bootstrap e ao arquivo **estilos.css**:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1.0">
  <title>Localização</title>
  <link rel="stylesheet" href=
  "https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.
  css" />

  <link rel="stylesheet" href="../css/estilos.css">
</head>
<body>

</body>
</html>
```

3. Insira uma **<div>** com a classe **container** e, dentro dela, outra **<div>** com a classe **borda**. O conteúdo do arquivo deve ser semelhante a este:

```
<body>
  <div class="container">
    <h1>Mapa de localização da empresa</h1>
    <div class="borda">
    </div>
  </div>
</body>
```

4. Nesta última **<div>**, inclua o endereço da empresa:

```
<body>
  <div class="container">
    <h1>Mapa de localização da empresa</h1>
    <div class="borda">
      <p>
        Avenida Paulista, 1009 - 17º Andar. <br />
        São Paulo. SP <br/>
        CEP 01311-100
      </p>
    </div>
  </div>
</body>
```

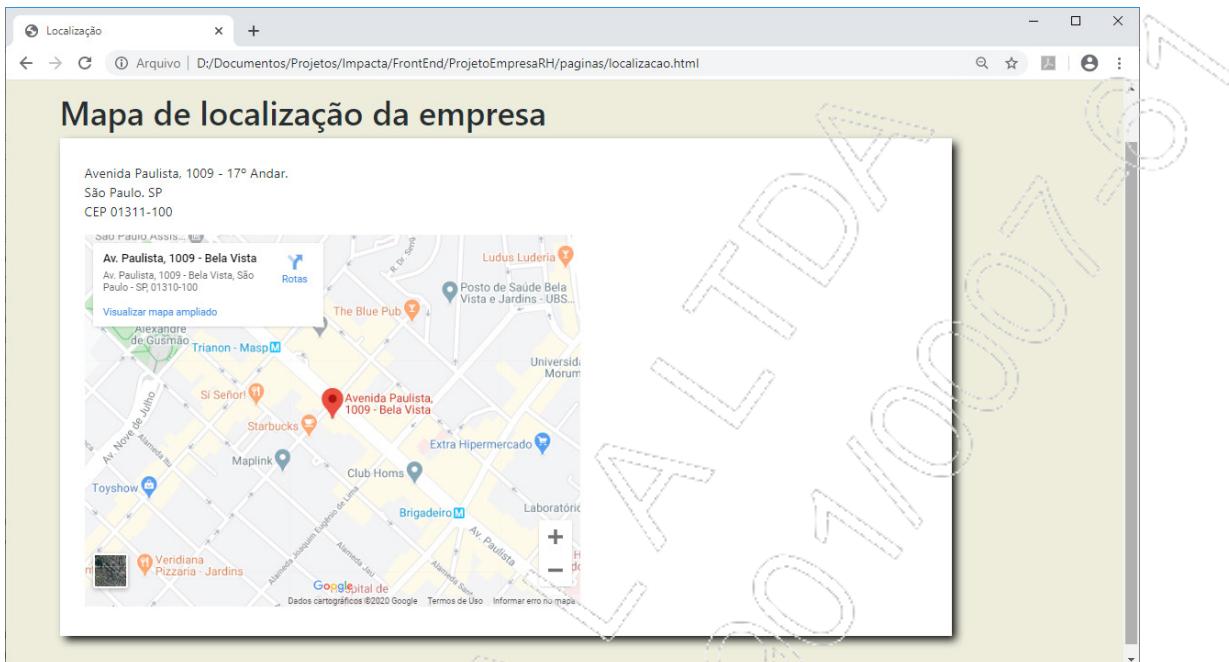
5. No navegador, abra o Google Maps;
6. Inclua o endereço da empresa;
7. Clique no menu e, em seguida, na opção **Compartilhar ou Incorporar Mapa**;
8. Escolha **Incorporar Mapa**;
9. Copie o código HTML de incorporação dentro de uma nova div:

```
<body>
  <div class="container">
    <h1>Mapa de localização da empresa</h1>
    <div class="borda">
      <p>
        Avenida Paulista, 1009 - 17º Andar. <br />
        São Paulo. SP <br />
        CEP 01311-100
      </p>

      <div>
        <iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d365
7.084187565715!2d-46.65452158543306!3d-23.565419684680933!2m3!1f0!2f0!3f0!3m2!
1i1024!2i768!4f13.1!3m3!1m2!1s0x94ce59c7947976cb%3A0xcfcae32dae64ebfda!2sAv.%20
Paulista%2C%201009%20-%20Bela%20Vista%2C%20S%C3%A3o%20Paulo%20-%20SP%2C%20
01310-100!5e0!3m2!1spt-BR!2sbr!4v1587082422616!5m2!1spt-BR!2sbr"
          width="600" height="450"
          frameborder="0" style="border:0;"
          allowfullscreen="" aria-hidden="false" tabindex="0"></iframe>
      </div>
    </div>
  </div>
</body>
```

Otimizações de layout com Bootstrap

10. Ajuste o código e visualize o mapa, executando a página. O resultado será semelhante a este:



11. Atualize o menu de opções em **index.html**:

```
<nav>
  <ul class="menu" id="menuTopo">
    <li><a href="#">A Empresa</a></li>
    <li><a href="paginas/produtos.html">Produtos</a></li>
    <li><a href="paginas/localizacao.html">Localização</a></li>
    <li><a href="paginas/registro.html">Trabalhe Conosco</a></li>

    <li class="icon">
      <a href="javascript:void(0);"
        onclick="mostrarMenu()">#</a>
    </li>
  </ul>
</nav>
```




9

Inclusão e listagem de registros com Web services



Atividade



Laboratório 1

Neste projeto, incluiremos um recurso para adicionar os dados dos candidatos em um banco de dados simulado com o componente **json-server**, assim como fizemos para buscar a lista de vagas.

A – Criando arquivo de dados e configurando eventos para adicionar registros via JavaScript

1. Abra o arquivo **dados.json**. Acrescente uma lista vazia chamada **candidatos**. Essa lista será usada para receber os novos registros:

```
{  
  "vagas": [  
    {  
      "id": 1,  
      "titulo": "Programador Java Pleno",  
      "tipo": "1"  
    },  
    {  
      "id": 2,  
      "titulo": "Analista Comercial",  
      "tipo": "2"  
    },  
    {  
      "id": 3,  
      "titulo": "Instrutor Excel Avançado",  
      "tipo": "1"  
    },  
    {  
      "id": 4,  
      "titulo": "Gerente de Projetos para .NET",  
      "tipo": "2"  
    },  
    {  
      "id": 5,  
      "titulo": "Consultor de Desenvolvimento Agil",  
      "tipo": "2"  
    },  
    {  
      "id": 6,  
      "titulo": "Programador VB.Net",  
      "tipo": "1"  
    }  
  ],  
  "candidatos": [ ]  
}
```

Inclusão e listagem de registros com Web services

2. Para este projeto, usaremos o jQuery para acessar o serviço, como alternativa à função `fetch`. Na pasta `scripts`, crie um arquivo chamado `cadastro.js`. Adicione o seguinte conteúdo:

```
$(document).ready(function () {
  $('#incluirButton').click(function(){
    //determinação do sexo com base
    //no item selecionado
    let sexo = '';
    if($('#masculino').prop('checked')){
      sexo = 'masculino';
    } else {
      sexo = 'feminino';
    }

    //obtenção do texto referente à vaga
    //selecionada
    let vaga = $('#vaga').find(":selected").text();

    $.ajax({
      url: 'http://localhost:3000/candidatos',
      method: 'POST',
      data: {
        nome: $('#nome').val(),
        data: $('#data').val(),
        sexo: sexo,
        telefone: $('#telefone').val(),
        email: $('#email').val(),
        vaga: vaga
      },
      success: function(resposta){
        window.alert('Dados incluídos');
      },
      error: function(erro){
        window.alert('Erro: ' + erro.responseText);
      }
    });
  });
});
```

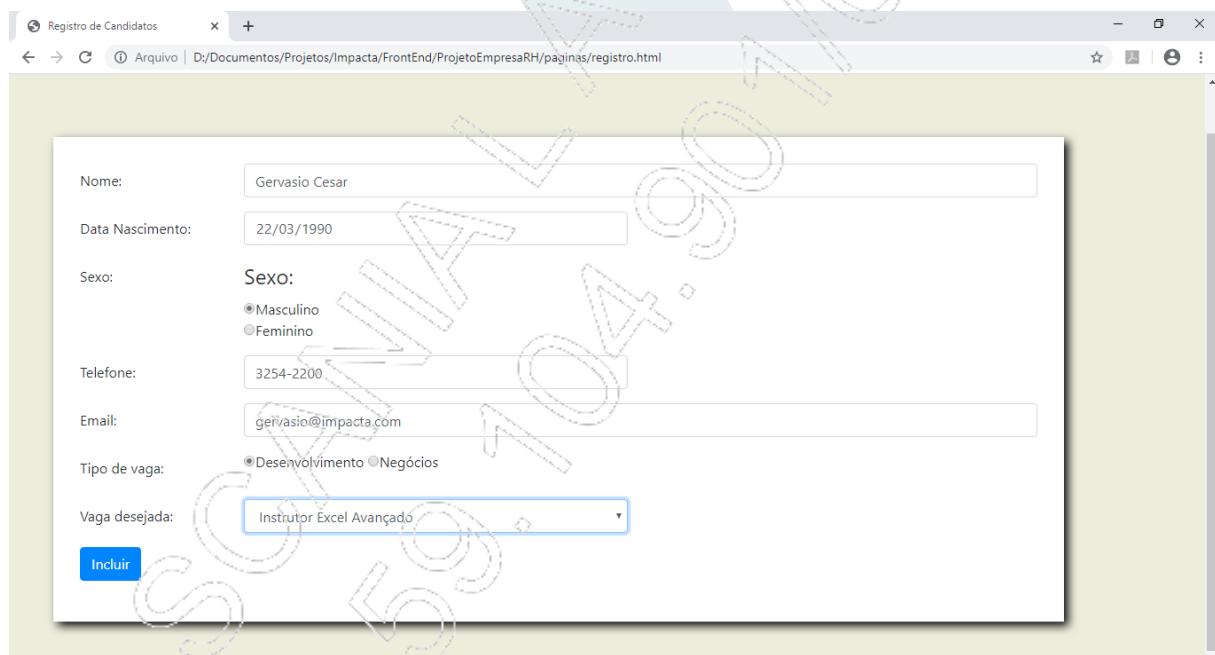
3. No arquivo **registro.html**, adicione a referência ao jQuery e, em seguida, a referência ao arquivo **cadastro.js**. É importante que a referência ao jQuery seja incluída antes de ser usada:

```
<script src="http://code.jquery.com/jquery-3.4.1.min.js"></script>
<script src="../scripts/validacao_registro.js"></script>
<script src="../scripts/listaVagas.js"></script>
<script src="../scripts/cadastro.js"></script>
</body>

</html>
```

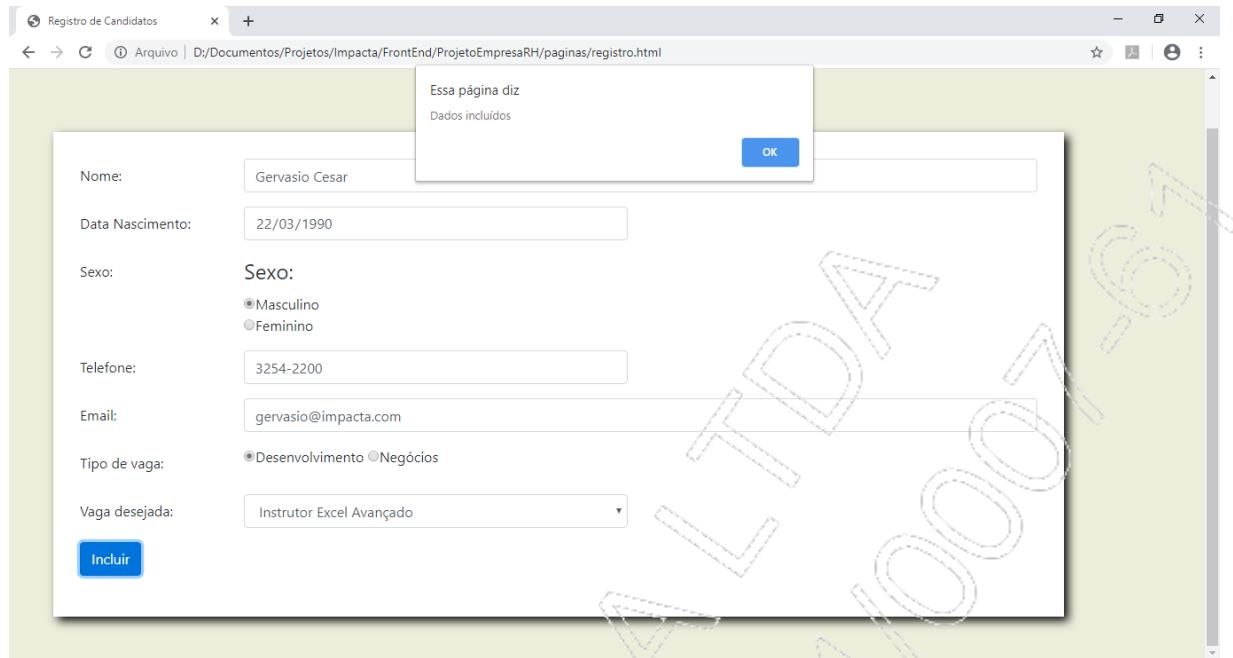
4. Mantenha o Web service em execução, acionando a instrução **json-server dados.json** no prompt de comandos (lembre-se que essa instrução deve ser executada na pasta onde se encontra o arquivo **dados.json**);

5. Execute o arquivo **registro.html** no browser. Informe alguns dados no formulário, como no exemplo:



Inclusão e listagem de registros com Web services

6. Ao clicar no botão **Incluir**, receberemos a mensagem de sucesso (claro, se o código estiver correto!):



7. Observe o arquivo **dados.json**. As informações fornecidas no formulário foram gravadas a partir do componente **json-server**:

```
{  
  "vagas": [  
    {  
      "id": 1,  
      "titulo": "Programador Java Pleno",  
      "tipo": "1"  
    },  
    {  
      "id": 2,  
      "titulo": "Analista Comercial",  
      "tipo": "2"  
    },  
    {  
      "id": 3,  
      "titulo": "Instrutor Excel Avançado",  
      "tipo": "1"  
    },  
    {  
      "id": 4,  
      "titulo": "Gerente de Projetos para .NET",  
      "tipo": "2"  
    },  
  ]}
```

```
{  
    "id": 5,  
    "titulo": "Consultor de Desenvolvimento Agil",  
    "tipo": "2"  
},  
{  
    "id": 6,  
    "titulo": "Programador VB.Net",  
    "tipo": "1"  
}  
],  
"candidatos": [  
{  
    "nome": "Gervasio Cesar",  
    "data": "1990-03-22",  
    "sexo": "masculino",  
    "telefone": "3254-2200",  
    "email": "gervasio@impacta.com",  
    "vaga": "Instrutor Excel Avançado",  
    "id": 1  
}  
]  
}
```

O atributo **id** foi inserido pelo próprio **json-server**. Esse atributo é requerido por esse componente e representa a chave primária do registro, mesmo que de forma simulada.

B – Apresentando a lista de candidatos

1. Na pasta **paginas**, inclua o arquivo **listaCandidatos.html**. Escreva a estrutura para o conteúdo HTML5 e já inclua a referência ao Bootstrap e ao jQuery:

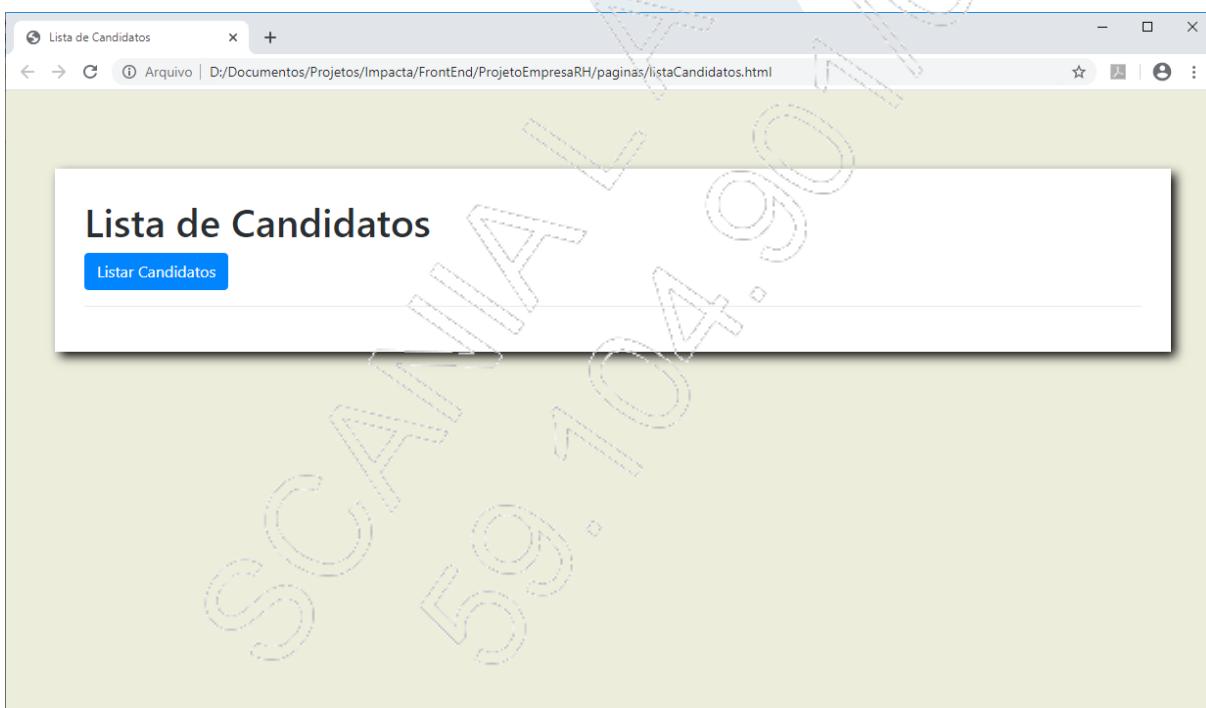
```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <meta name="viewport"  
        content="width=device-width, initial-scale=1.0">  
    <title>Lista de Candidatos</title>  
    <link rel="stylesheet" href=  
"https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" />  
        <link rel="stylesheet" href="..../css/estilos.css">  
</head>  
<body>  
  
    <script src="http://code.jquery.com/jquery-3.4.1.min.js"></script>  
</body>  
</html>
```

2. Nessa nova página, apresentaremos a lista de candidatos cadastrados quando o usuário clicar em um botão com o título **Listar Candidatos**. Vamos, então, escrever a estrutura básica para essa página:

```
<body>
  <div class="container borda">
    <h1>Lista de Candidatos</h1>
    <div>
      <button type="button" id="listarButton"
        class="btn btn-primary">Listar Candidatos</button>
      <hr />
    </div>
  </div>

  <script src="http://code.jquery.com/jquery-3.4.1.min.js"></script>
</body>
```

3. O resultado da execução produz o resultado adiante:



4. Abaixo do botão e da linha horizontal, escreva a estrutura para uma tabela, usando as classes do Bootstrap:

```
<body>
  <div class="container borda">
    <h1>Lista de Candidatos</h1>
    <div>
      <button type="button" id="listarButton"
        class="btn btn-primary">Listar Candidatos</button>
      <hr />

      <table class="table table-striped" id="tabela">
        <thead>
          <tr>
            <th>ID</th>
            <th>NOME</th>
            <th>TELEFONE</th>
          </tr>
        </thead>
        <tbody>
          </tbody>
      </table>

    </div>
  </div>

  <script src="http://code.jquery.com/jquery-3.4.1.min.js"></script>
</body>
```

5. No arquivo **cadastro.js**, adicione ao final uma função que execute a tarefa de consumir o Web service. Em seguida, execute essa função no evento click do botão inserido nessa página:

```
$(document).ready(function () {
  $('#incluirButton').click(function () {
    //continuação do código, omitido por conveniência
  });

  //listando os candidatos
  $('#listarButton').click(function () {
    $.ajax({
      dataType: 'json',
      url: 'http://localhost:3000/candidatos',
      method: 'GET',
      success: function (resposta) {
        $('#tabela > tbody').html('');
        $.each(resposta, function (index, item) {
          let linha = $('<tr>');
          let coluna = $('<td>' + item.NOME + '</td>');
          linha.append(coluna);
          $('#tabela > tbody').append(linha);
        });
      }
    });
  });
});
```

```
let colunas = '<td>' + item.id + '</td>';
colunas += '<td>' + item.nome + '</td>';
colunas += '<td>' + item.telefone + '</td>';

linha.append(colunas);

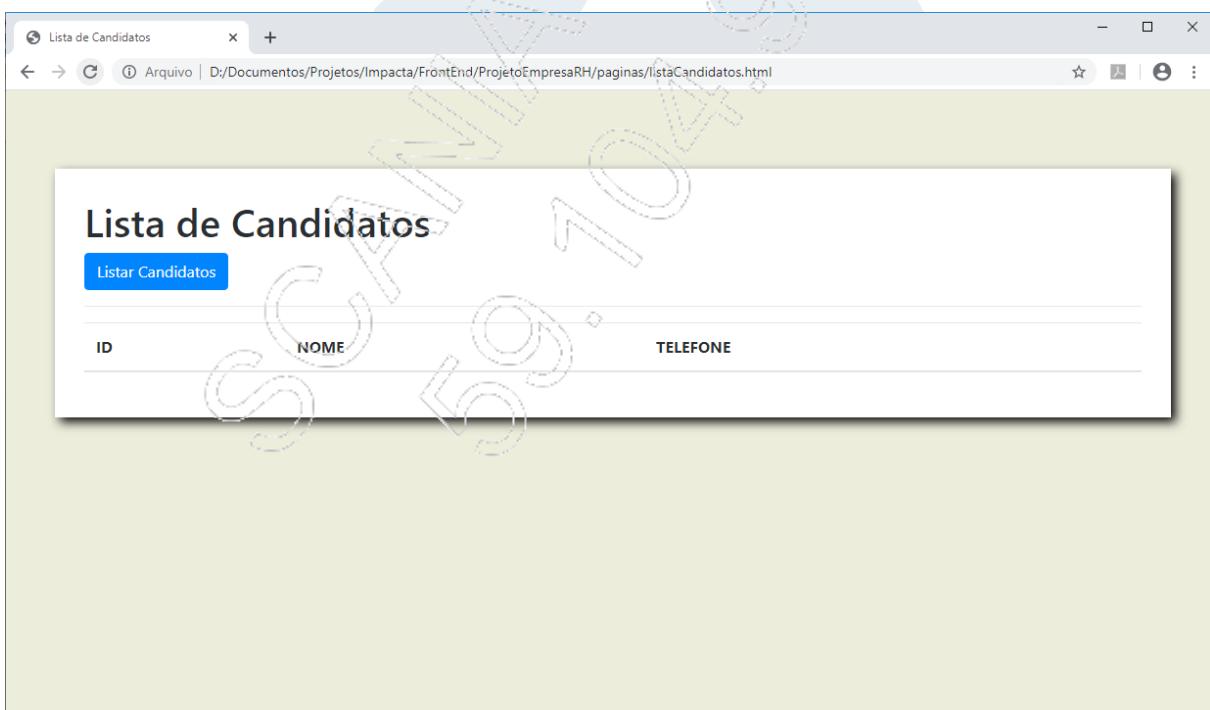
$('#tabela > tbody').append(linha);
},
error: function (erro) {
  console.log(erro.responseText);
}
});
});

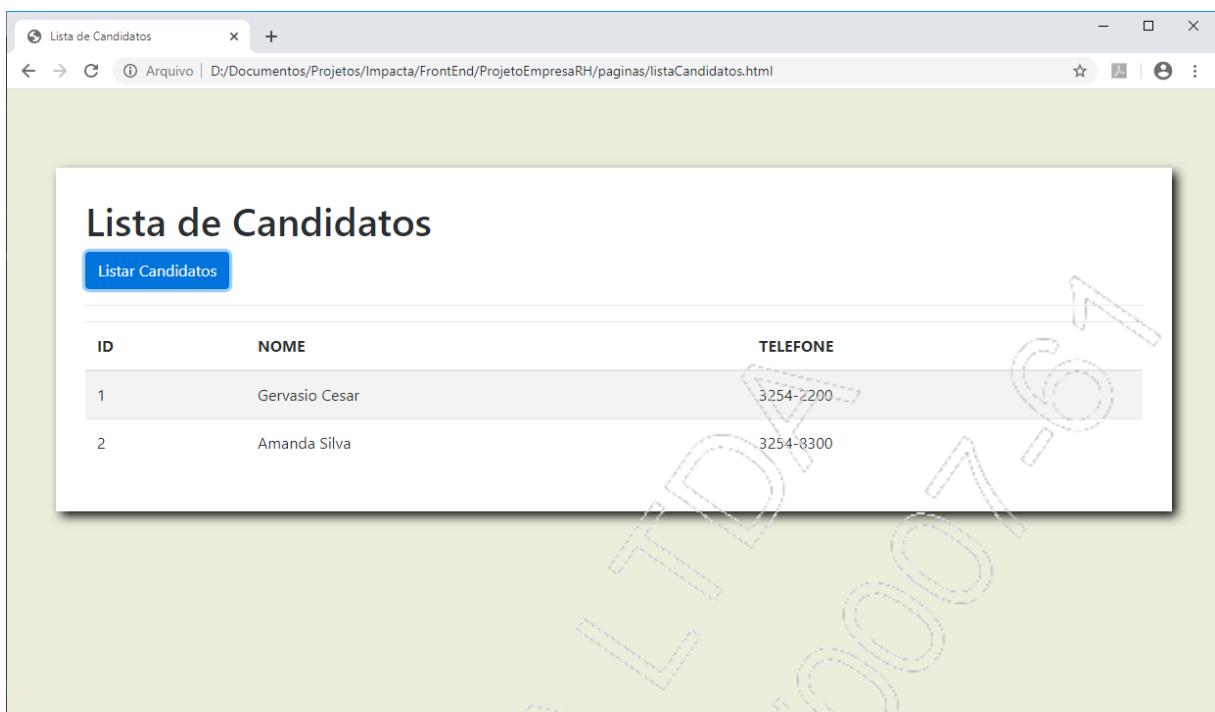
});
```

6. Adicione a referência ao arquivo **cadastro.js** em **listaCandidatos.html**:

```
<script src="http://code.jquery.com/jquery-3.4.1.min.js"></script>
<script src="../scripts/cadastro.js"></script>
</body>
```

7. Execute a página **listaCandidatos.html**. Com o Web service em execução, clique no botão **Listar Candidatos**:





8. Atualize o arquivo **index.html** para incluir um link para a listagem de candidatos:

```
<ul class="menu" id="menuTopo">
    <li><a href="#">A Empresa</a></li>
    <li><a href="paginas/produtos.html">Produtos</a></li>
    <li><a href="paginas/localizacao.html">Localização</a></li>
    <li><a href="paginas/registro.html">Trabalhe Conosco</a></li>
    <li><a href="paginas/listaCandidatos.html">Listar Candidatos</a></li>

    <li class="icon">
        <a href="javascript:void(0);" onclick="mostrarMenu()">#</a>
    </li>
</ul>
```



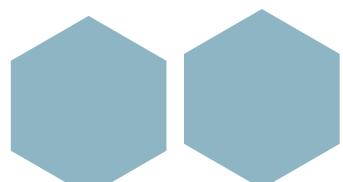
10

Aplicação de Canvas API



Atividade

ALTA
SCA
59.
104.
10007-67



Laboratório 1

A – Definindo um relógio na página de registro de candidatos

Adicionaremos um código baseado em Canvas API para mostrar um relógio na página **registro.html**. O código referente a esse relógio foi obtido em <https://developer.mozilla.org/pt-BR/docs/Web/Guide/HTML/Canvas_tutorial/Basic_animations>.

O objetivo é ter a oportunidade de incluir um elemento usando esta API que, quando bem explorada, oferece recursos bastante interessantes para nossa aplicação.

1. Crie, na pasta **scripts**, um arquivo chamado **relogio.js**;
2. Nesse arquivo, inclua este código (se preferir, pode buscar no link indicado anteriormente):

```
function init() {
    clock();
    setInterval(clock, 1000);
}

function clock() {
    var now = new Date();
    var ctx = document.getElementById('relogio').getContext('2d');

    ctx.save();
    ctx.clearRect(0, 0, 150, 150);
    ctx.translate(75, 75);
    ctx.scale(0.4, 0.4);
    ctx.rotate(-Math.PI / 2);
    ctx.strokeStyle = "black";
    ctx.fillStyle = "white";
    ctx.lineWidth = 8;
    ctx.lineCap = "round";

    // Hour marks
    ctx.save();
    for (var i = 0; i < 12; i++) {
        ctx.beginPath();
        ctx.rotate(Math.PI / 6);
        ctx.moveTo(100, 0);
        ctx.lineTo(120, 0);
        ctx.stroke();
    }
    ctx.restore();
}
```

```
// Minute marks
ctx.save();
ctx.lineWidth = 5;
for (i = 0; i < 60; i++) {
    if (i % 5 != 0) {
        ctx.beginPath();
        ctx.moveTo(117, 0);
        ctx.lineTo(120, 0);
        ctx.stroke();
    }
    ctx.rotate(Math.PI / 30);
}
ctx.restore();

var sec = now.getSeconds();
var min = now.getMinutes();
var hr = now.getHours();
hr = hr >= 12 ? hr - 12 : hr;

ctx.fillStyle = "black";

// write Hours
ctx.save();
ctx.rotate(hr * (Math.PI / 6) + (Math.PI / 360) * min +
           (Math.PI / 21600) * sec)
ctx.lineWidth = 14;
ctx.beginPath();
ctx.moveTo(-20, 0);
ctx.lineTo(80, 0);
ctx.stroke();
ctx.restore();

// write Minutes
ctx.save();
ctx.rotate((Math.PI / 30) * min + (Math.PI / 1800) * sec)
ctx.lineWidth = 10;
ctx.beginPath();
ctx.moveTo(-28, 0);
ctx.lineTo(112, 0);
ctx.stroke();
ctx.restore();
```

```
// Write seconds
ctx.save();
ctx.rotate(sec * Math.PI / 30);
ctx.strokeStyle = "#D40000";
ctx.fillStyle = "#D40000";
ctx.lineWidth = 6;
ctx.beginPath();
ctx.moveTo(-30, 0);
ctx.lineTo(83, 0);
ctx.stroke();
ctx.beginPath();
ctx.arc(0, 0, 10, 0, Math.PI * 2, true);
ctx.fill();
ctx.beginPath();
ctx.arc(95, 0, 10, 0, Math.PI * 2, true);
ctx.stroke();
ctx.fillStyle = "rgba(0,0,0,0)";
ctx.arc(0, 0, 3, 0, Math.PI * 2, true);
ctx.fill();
ctx.restore();

ctx.beginPath();
ctx.lineWidth = 14;
ctx.strokeStyle = '#325FA2';
ctx.arc(0, 0, 142, 0, Math.PI * 2, true);
ctx.stroke();

ctx.restore();
}
init();
```

3. Em `registro.html`, inclua o elemento `<canvas>` logo após `<div class="container borda">`:

```
<div class="container borda">
<canvas id="relogio"></canvas>
<!-- Nome do candidato --&gt;</pre>
```

4. Inclua a referência ao arquivo `relogio.js` ao final da página:

```
<script src="http://code.jquery.com/jquery-3.4.1.min.js"></script>
<script src="../scripts/validacao_registro.js"></script>
<script src="../scripts/listaVagas.js"></script>
<script src="../scripts/cadastro.js"></script>
<script src="../scripts/relogio.js"></script>
</body>
```

5. Execute a página e visualize o relógio.



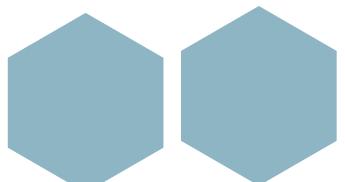
11

Uso de áudio e vídeo



Atividade

ALTA
SCAMIA
59.
10007-67



Laboratório 1

A – Incluindo um áudio e um vídeo na aplicação

O propósito desta etapa é adicionar um vídeo na página inicial e um áudio convidando os candidatos a se inscreverem no site.

1. Selecione um vídeo. Procure colocar um vídeo que possua algum significado para a aplicação;
2. No projeto, crie uma pasta chamada **vídeos** e inclua o vídeo que você selecionou;
3. Na página **index.html**, localize o elemento **<article>**. Inclua o elemento a seguir como último item de **<article>**:

```
<h2>Assista ao nosso vídeo, e conheça melhor a empresa</h2>
<div id="video" style="text-align:center;">
  <video controls="controls" width="400" height="300">
    <source src="videos/video.mp4" type="video/mp4" />
    <p>
      Seu navegador não tem suporte ao elemento video
    </p>
  </video>
</div>
</article>
```

4. Agora, use o recurso do Windows para gravar um áudio. Apresente uma mensagem de boas-vindas para os candidatos. Chame o áudio de **welcome.mp3**;

5. Crie uma pasta chamada **audios** e inclua seu áudio nessa pasta;
6. Inclua esse áudio na página **registro.html**, logo após o relógio:

```
<canvas id="relogio"></canvas>

<div>
  <audio src="../audios/welcome.mp3" controls autoplay loop></audio>
</div>
```

7. Se preferir, use os recursos apresentados na aula para incluir botões personalizados.



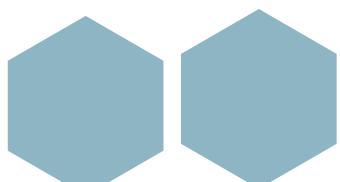
12

Armazenamento local



Atividade

SCALVIM ALTA 59. 104. 9007-67



Laboratório 1

A – Criando um banco de dados IndexedDb e recursos para inserir e listar registros

Nesta etapa do projeto, desenvolveremos um registro local para armazenamento dos candidatos cadastrados, como alternativa ao cadastro inserido no Web service que desenvolvemos. O propósito é treinar o uso do componente JavaScript **indexedDb**. Serão realizadas as devidas verificações a respeito da possibilidade de acessá-lo. Para completar essa tarefa, realizaremos algumas alterações em arquivos existentes.

1. No arquivo **registro.html**, inclua um elemento `<div>` para apresentar a mensagem do banco de dados. Observe as alterações realizadas:

```
<div class="container borda">
  <div class="row">
    <div class="col-sm-6 text-center">
      <canvas id="relogio"></canvas>
    </div>
    <div class="col-sm-6 text-center">
      <div id="mensagemdb"></div>
    </div>
  </div>
```

2. Acrescente um novo botão ao lado do botão **Incluir** com o título **Enviar para o IndexedDb**, como no exemplo a seguir:

```
<!-- Botão de comandos -->
<div class="form-group row">
  <div class="col-sm-5 col-sm-offset-2">
    <input type="button" value="Incluir" id="incluirButton"
           class="btn btn-primary">
    <input type="button" value="Enviar para o IndexedDb"
           id="incluirDbButton" class="btn btn-primary">
  </div>
</div>
```

3. Altere o conteúdo do arquivo **estilos.css**, acrescentando os elementos listados a seguir (estes elementos serão usados pelo código jQuery):

```
/*
  Classes para mensagem de erro ou sucesso
  no cadastro de candidatos via indexedDb
*/
.erro_db{
  color:red;
}
.sucesso_db {
  color:blue;
}
```

4. Na pasta **paginas**, inclua uma nova página chamada **verRegistros.html**. O conteúdo dessa nova página deve ser o seguinte:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
        content="width=device-width, initial-scale=1,0">
  <title>Registros do IndexedDb</title>
  <link rel="stylesheet" href=
"https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" />
  <link rel="stylesheet" href="../css/estilos.css">
</head>
<body>
  <div class="container borda">
    <h1>Listagem de Candidatos</h1>
    <div>
      <button type="button" id="listarDbButton"
             class="btn btn-primary">
        Ver os candidatos
      </button>
      <hr />
    </div>
    <div>
      <ul id="listaCandidatos"></ul>
    </div>
  </div>

  <script src="http://code.jquery.com/jquery-3.4.1.min.js"></script>
</body>
</html>
```

5. Na pasta **scripts**, crie um novo arquivo chamado **database.js**. Em seguida, inclua a referência a esse arquivo em **verRegistros.html**:

```
<script src="http://code.jquery.com/jquery-3.4.1.min.js"></script>
<script src="../scripts/database.js"></script>
</body>
```

6. Nesse arquivo, teremos instruções para criar / abrir um banco de dados chamado **DBCandidatos**, com um registro (tabela) chamado **candidatos**. Analise a estrutura e inclua este conteúdo no arquivo:

```
let request, db;

request = window.indexedDB.open("DBCandidatos", 1);
request.onerror = function(event) {
    $("#mensagemdb").addClass("erro_db");
    $("#mensagemdb").html("Erro ao abrir o banco de dados");
    $("#btnEnviar").prop("disabled", true);
}
request.onupgradeneeded = function(event) {
    $("#mensagemdb").addClass("sucesso_db");
    $("#mensagemdb").html("Banco de dados preparado para uso");

    db = event.target.result;
    let objectStore = db.createObjectStore("candidatos",
        { keyPath: "email" });
};

request.onsuccess = function(event) {
    $("#mensagemdb").addClass("sucesso_db");
    $("#mensagemdb").html("Banco de dados aberto com sucesso");

    db = event.target.result;
}

//evento para incluir um novo candidato
$("#incluirDbButton").click(function(){
    let nome = $("#nome").val();
    let datanasc = $("#data").val();
    let sexo;
    if ($("#masculino").is(':checked')) {
        sexo = "masculino";
    } else {
        sexo = "feminino";
    }
    let telefone = $("#telefone").val();
    let email = $("#email").val();
    let vaga = $('#vaga').find(":selected").text();

    let transaction = db.transaction(["candidatos"], "readwrite");
```

```

transaction.oncomplete = function (event) {
    $(location).attr("href", "/paginas/registroOk.html");
};

transaction.onerror = function (event) {
    alert("Ocorreu um erro ao incluir o registro");
};

var objStore = transaction.objectStore("candidatos");
objStore.add({
    email: email, nome: nome, data: datanasc, sexo: sexo,
    telefone: telefone, vaga: vaga
});
});

//evento para listar os candidatos, adicionando-os em uma lista <li>
$("#listarDbButton").click(function () {
    let request = db.transaction(["candidatos"], "readonly")
        .objectStore("candidatos");

    request.openCursor().onsuccess = function (event) {
        let cursor = event.target.result;
        if (cursor) {
            $("#listaCandidatos").append("<li>" + cursor.value.nome
                + "</li>");
            cursor.continue();
        }
    };
});

```

7. Na página **registro.html**, inclua a referência ao arquivo **database.js**:

```

<script src="http://code.jquery.com/jquery-3.4.1.min.js"></script>
<script src="../scripts/validacao_registro.js"></script>
<script src="../scripts/listaVagas.js"></script>
<script src="../scripts/cadastro.js"></script>
<script src="../scripts/relogio.js"></script>
<script src="../scripts/database.js"></script>

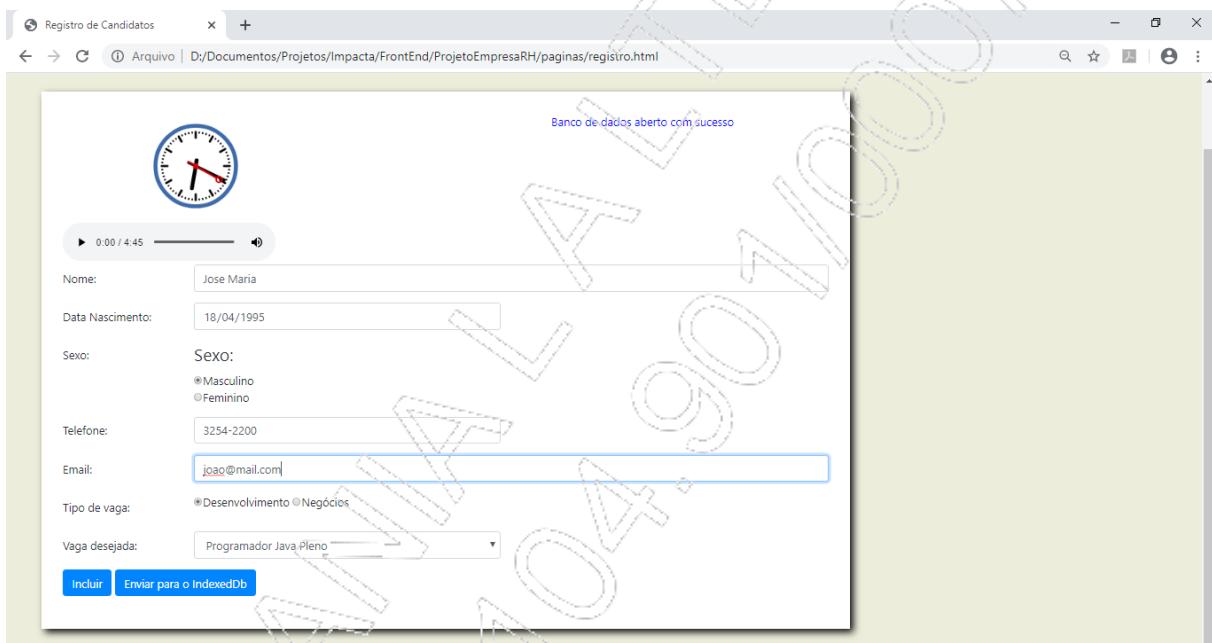
```

8. Na página **index.html**, adicione uma opção ao menu para listar os candidatos, contemplando a página recém-criada:

```
<ul class="menu" id="menuTopo">
    <li><a href="#">A Empresa</a></li>
    <li><a href="paginas/produtos.html">Produtos</a></li>
    <li><a href="paginas/localizacao.html">Localização</a></li>
    <li><a href="paginas/registro.html">Trabalhe Conosco</a></li>
    <li><a href="paginas/listaCandidatos.html">Listar Candidatos</a></li>
    <li><a href="paginas/verRegistros.html">Ver Candidatos</a></li>

    <li class="icon">
        <a href="javascript:void(0);" onclick="mostrarMenu()">#9776;</a>
    </li>
</ul>
```

9. Teste a aplicação, inclua alguns registros e liste-os:



A inclusão desse registro produz o resultado mostrado no browser (foi utilizado o Google Chrome):

#	Key (Key path: "email")	Value
0	"jose@mail.com"	{email: "jose@mail.com", nome: "Jose Maria", data: "1995-04-18", email: "jose@mail.com", nome: "Jose Maria", sexo: "masculino", telefone: "3254-2200", vaga: "Programador Java Pleno"}



13

Complemento: PhoneGap



Atividade

SCALDA
59.
104.
10007-67



Laboratório 1

A – Criando um aplicativo para cadastro de contato com PhoneGap

Este é um projeto que representa o cadastro de contatos por meio de um aplicativo, a ser usado como complemento do curso. A proposta é mostrar o caminho para a elaboração do aplicativo.

1. Abra o prompt de comandos apontando para uma pasta diferente da pasta **ProjetoEmpresaRH**;
2. No prompt aberto, crie um projeto do tipo PhoneGap digitando este comando:

```
phonegap create app projeto.complemento ProjetoPhonegap
```

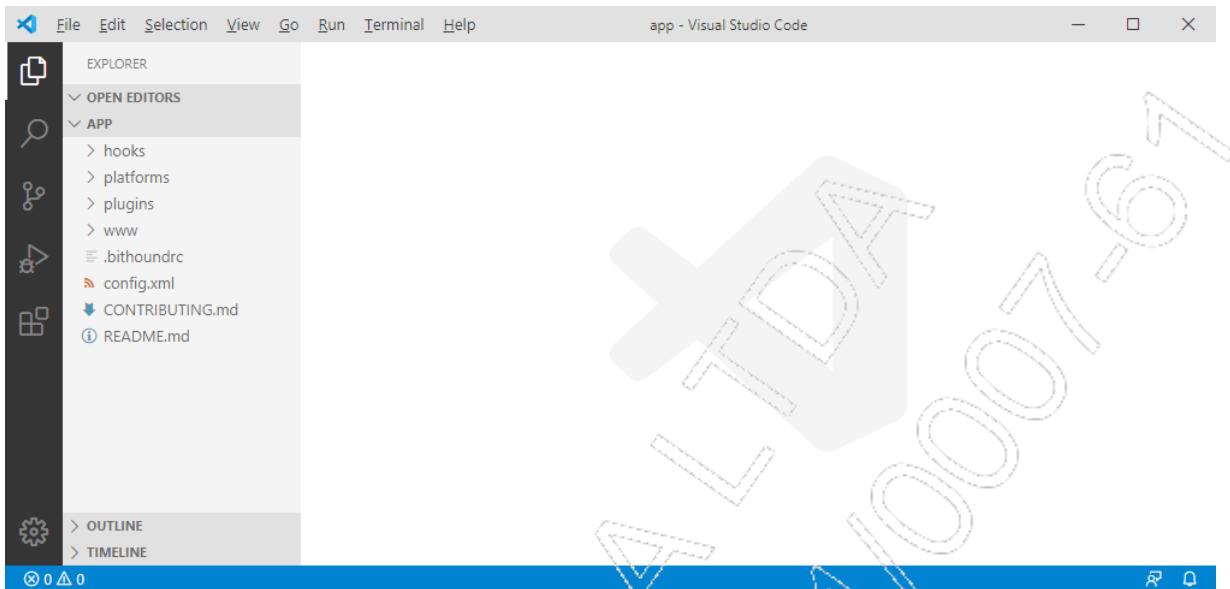


```
C:\Windows\System32\cmd.exe
Microsoft Windows [versão 10.0.18363.778]
(c) 2019 Microsoft Corporation. Todos os direitos reservados.

D:\Documentos\Projetos\Impacta\FrontEnd>phonegap create app projeto.complemento ProjetoPhonegap
```

Observe que o nome do projeto é **ProjetoPhonegap**.

3. Abra o Visual Studio Code apontando para a pasta do novo projeto. A pasta se chama **app**:



4. Na pasta **js**, adicione o arquivo **scripts.js** com o seguinte conteúdo:

```
var request, db;

request = window.indexedDB.open("DBContatos", 1);
request.onerror = function (event) {
    window.alert('Erro ao abrir banco de dados');
}
request.onupgradeneeded = function (event) {
    db = event.target.result;
    var objectStore = db.createObjectStore("contatos",
        { keyPath: "nome" });
}
request.onsuccess = function (event) {
    db = event.target.result;
}

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

```
//buscar o endereço pelo cep
$("#cep").blur(function(){
    let cep = $(this).val();

    let url = `http://viacep.com.br/ws/${cep}/json/`;

    let resposta = {};

    fetch(url)
        .then(res => {
            let x = res.json();
            return x;
        })
        .then(valor => {
            resposta = valor;
            $('#rua').val(resposta.logradouro);
            $('#cidade').val(resposta.localidade);
            $('#numero').focus();
        });
});

//incluir o registro no indexeddb
$("#btnEnviar").click(function () {
    var nome = $("#nome").val();
    var telefone = $("#telefone").val();
    var cep = $("#cep").val();
    var numero = $("#numero").val();

    var transaction = db.transaction(["contatos"], "readwrite");

    transaction.oncomplete = function (event) {
        window.alert('Registro incluído com sucesso');
    };
    transaction.onerror = function (event) {
        window.alert("Ocorreu um erro ao incluir o registro");
    };
    var objStore = transaction.objectStore("contatos");
    objStore.add({
        nome: nome, telefone: telefone, cep: cep, numero: numero
    });
});
});
```

5. Abra o arquivo **index.html** na pasta **www**. Deixe o conteúdo como mostrado no exemplo:

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />

    <meta name="viewport" content="initial-scale=1,
        width=device-width" />

    <title>Contatos</title>
    <link rel="stylesheet" href=
"https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"/>
</head>

<body>
    <div class="container">
        <h2>Registro de Contatos</h2>
        <h4>Dados Pessoais</h4>
        <div>
            <div class="form-group">
                <label>Nome:</label>
                <input type="text" class="form-control" id="nome" />
            </div>
            <div class="form-group">
                <label>Telefone:</label>
                <input type="text" class="form-control" id="telefone" />
            </div>
        </div>
        <h4>Dados do Endereço</h4>
        <div>
            <div class="form-group">
                <label>Informe o CEP:</label>
                <input type="text" class="form-control" id="cep" />
            </div>
            <div class="form-group">
                <label>Rua:</label>
                <input type="text" class="form-control" id="rua"
                    readonly="readonly" />
            </div>
        </div>
    </div>
</body>
```

```
<div class="form-group">
    <label>Cidade:</label>
    <input type="text" class="form-control" id="cidade"
        readonly="readonly" />
</div>
<div class="form-group">
    <label>Informe o Número:</label>
    <input type="text" class="form-control" id="numero" />
</div>
<button type="button" class="btn btn-primary"
    id="btnEnviar">Enviar</button>
</div>

</div>
<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
<script src="js/scripts.js"></script>
</body>

</html>
```

6. Abra o arquivo **config.xml**. Altere o início do arquivo com estes dados:

```
<?xml version='1.0' encoding='utf-8'?>
<widget id="projeto.complemento" version="1.0.0"
    xmlns="http://www.w3.org/ns/widgets"
    xmlns:gap="http://phonegap.com/ns/1.0">

    <name>ProjetoPhonegap</name>
    <description>
        Cadastro de Contatos.
    </description>
    <author email="seuemail@seuprovedor.com"
        href="http://www.impacta.com.br">
        FrontEnd Impacta
    </author>
```

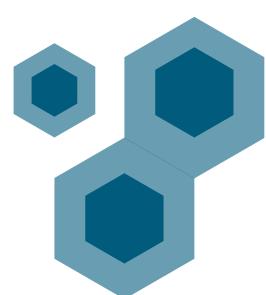
7. Com essas alterações, compacte (zipe) a pasta **app**, gerando o arquivo **app.zip**;

8. Use suas credenciais no portal <<https://build.phonegap.com/>>;

9. Faça o upload do arquivo **app.zip**;

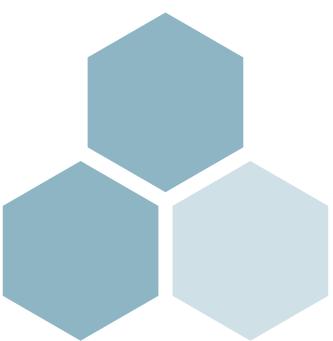
10. Aguarde o tempo de compilação. Em seguida, obtenha o arquivo **.apk** gerado e transfira para o seu celular Android;

11. Visualize a execução no dispositivo.



Mão à obra!

SCANNIK ALTDA
59.704.071/0007-67



Editora
IMPACTA





1

Criando uma página HTML



Mãos à obra!



Laboratório 1

A - Elaborando um currículo pessoal

1. Usando o Visual Studio Code, crie uma nova pasta chamada **Laboratorios**;
2. Em seguida, adicione uma pasta chamada **Laboratorio01**;
3. Na pasta **Laboratorio01**, crie uma pasta chamada **pessoal**;
4. Na pasta **pessoal**, inclua um arquivo HTML chamado **curriculum.html**;
5. Pesquise informações a respeito de um curriculum vitae;
6. Use, entre outros, os seguintes elementos:
 - **Html**;
 - **Head**;
 - **Body**;
 - **Header**;
 - **Article**;
 - **Div**;
 - **h1, h2**;
 - **P**;
 - **Section**;
 - **Footer**;
 - **Img**.
7. Elabore seu currículo considerando, pelo menos, as seguintes informações:
 - **Dados pessoais**: Nome, CPF, data de nascimento, telefone, e-mail;
 - **Dados de localização**: Endereço completo;
 - **Experiência profissional**: Destaque todos os itens relevantes referentes à sua experiência profissional;
 - **Pretensão salarial**;
 - **Cargos pretendidos**;
 - **Foto recente**.
8. Para destacar a experiência profissional, use listas, ou seja, a combinação dos elementos **** e ****;
9. Execute a página e verifique se ficou apresentável. Se não ficou, retorne e refaça as etapas das quais não tenha gostado.



2

Desenvolvendo formulários



Mãos à obra!



Laboratório 1

A - Criando um formulário para inclusão de currículo

1. Na pasta **Laboratorios**, crie uma nova pasta chamada **Laboratorio02**;
2. Na pasta **Laboratorio02**, copie o conteúdo da pasta **Laboratorio01**;
3. Na pasta **Laboratorio02**, inclua uma pasta chamada **forms**;
4. Na pasta **forms**, crie um arquivo HTML chamado **formCurriculum.html**;
5. Crie um título, usando o elemento **h1**;
6. Escreva o conteúdo do formulário usando caixas de texto adequadas para cada finalidade. As informações que devem constar no curriculum vitae, juntamente com as sugestões de componentes, estão listadas a seguir:
 - Dados pessoais (**fieldset**): CPF, nome, data de nascimento, sexo, telefone, e-mail, foto (**input** e **img**);
 - Endereço (**fieldset**): Logradouro, número, complemento, bairro, cidade, estado, CEP (**input**, e **select** para listar os estados);
 - Experiência profissional (**fieldset**): Resumo da experiência profissional (**textarea**);
 - Pretensões (**fieldset**): Cargo pretendido (**select** – incluir três elementos), salário pretendido (**input**).
7. Inclua um botão do tipo **submit** e outro do tipo **reset**;
8. Para todos os campos, inclua validadores:
 - Todos os campos devem ser obrigatórios;
 - O campo **Nome** deve ter, no máximo, 50 caracteres;
 - O campo **Telefone** deve ter, no máximo, 20 caracteres.
9. Teste seu formulário.



3

Aplicando estilos



Mãos à obra!



Laboratório 1

A - Aperfeiçoando o currículo com estilos

1. Na pasta **Laboratorios**, crie uma nova pasta chamada **Laboratorio03**;
2. Na pasta **Laboratorio03**, copie o conteúdo da pasta **Laboratorio02**;
3. Na pasta **Laboratorio03**, inclua uma nova pasta chamada **css**;
4. Na pasta **css**, inclua um arquivo chamado **estilos.css**;
5. Analise seu arquivo **curriculum.html**;
6. Planeje um formato bem atraente para ele:
 - Seu nome em destaque, com fontes maiores;
 - Os dados pessoais logo abaixo do nome;
 - A sua foto preferencialmente do lado esquerdo;
 - Sua experiência profissional em um retângulo (**div**) com cantos arredondados, uma cor de fundo diferenciada, com destaque;
 - As informações de contato fáceis de encontrar: procure colocar em fontes coloridas, em negrito.
7. Planeje as classes **css** e as inclua no arquivo criado **estilos.css**;
8. Altere o arquivo **curriculum.html**, de modo que os elementos HTML contemplem essas classes;
9. Se for necessário, realize uma manutenção no seu HTML de modo a atender aos estilos definidos;
10. Inclua a referência a esse css na sua página;
11. Teste a página. Certamente, haverá a necessidade de realizar algumas alterações até que fique com uma apresentação amigável.

B - Aperfeiçoando o formulário com estilos

1. Vamos, agora, deixar o formulário referente ao laboratório anterior mais atraente. Para isso, abra a página **formCurriculum.html**;
2. Verifique seu conteúdo e planeje uma aparência funcional e amigável do ponto de vista do usuário;
3. Inclua, no arquivo **estilos.css**, os estilos necessários para:
 - Configurar as fontes das caixas de texto, todas com o mesmo nome, tamanho e cor;
 - Alinhar os componentes do formulário;
 - Incluir o conteúdo do formulário em um elemento capaz de dar um destaque, como uma cor de fundo diferenciada, bordas sombreadas e arredondadas.
4. Execute a página e analise seu resultado.

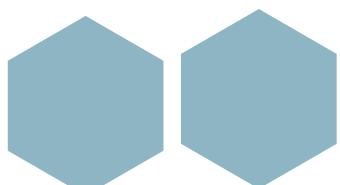


4

Aplicando responsividade



Mãos à obra!





Laboratório 1

A - Melhorando a aparência do currículo em dispositivos menores

1. Na pasta **Laboratorios**, crie uma nova pasta chamada **Laboratorio04**;
2. Na pasta **Laboratorio04**, copie o conteúdo da pasta **Laboratorio03**;
3. Abra os arquivos **curriculum.html** e **estilos.css**;
4. O curriculum vitae, inicialmente, estava com a foto ao lado das informações referentes aos dados pessoais. Complemente o seu css de forma que, para dispositivos menores, a foto fique acima dos dados pessoais;
5. Dependendo de como você organizou seu formulário na ocasião da aplicação do css, deixe todas as informações em um fluxo linear vertical.



5

JavaScript e jQuery



Mãos à obra!



Laboratório 1

A - Adicionando funcionalidade ao formulário

1. Na pasta **Laboratorios**, crie uma nova pasta chamada **Laboratorio05**;
2. Na pasta **Laboratorio05**, copie o conteúdo da pasta **Laboratorio04**;
3. Na pasta **Laboratorio05**, inclua uma pasta chamada **scripts**;
4. Na pasta **scripts**, inclua um novo arquivo chamado **validacao.js**;
5. Se os elementos do seu formulário ainda não possuem um id, inclua-o em todos os campos de formulário, incluindo o botão que permite a inclusão do registro;
6. Altere o modo do botão de **submit** para **button**;
7. No arquivo **validacao.js**, inclua códigos baseados em jQuery que realizem a sequência de tarefas adiante:
 - Definir o evento **click** para o botão, tomando como base o seu **id**;
 - Na função **call-back** correspondente ao evento, definir variáveis que obtenham todas as informações dos campos de entrada;
 - Criar uma variável cujo conteúdo seja formado com todas as informações dos campos de entrada, devidamente armazenados em variáveis, conforme item anterior;
 - Executar a função **alert()** para que essa variável seja exibida para o usuário.

 As outras funcionalidades serão descritas mais adiante.

8. Em um local do seu formulário (acima dos campos de entrada, ao lado ou abaixo dos botões, ou em outro lugar mais conveniente), inclua um elemento `<div>`, devidamente configurado no arquivo `estilos.css`. Esse `<div>` deve se apresentar com uma borda destacada, uma cor de fundo e fontes diferentes do formulário, um botão contendo um `x` para que seja fechado pelo usuário por meio do mouse, e uma largura menor que a largura do formulário. Atribua um `id` para essa nova `<div>`;

9. No arquivo `validacao.js`, continue com a codificação, seguindo mais estes passos:

- Usando a função de animação `fadeIn()`, escreva um grupo de instruções que, logo após a apresentação da mensagem (`alert()`), promova a aparição do seu novo `div`, suavemente, em três segundos;
- É importante que a `div` recém-inserida esteja escondida (invisível) assim que a página for carregada. Para isso, use a função `hide()` ao `div` por meio do seu `id`;
- Codifique um evento `click` para o botão localizado na `div` da mensagem, de forma que, clicando nele, a mensagem desapareça também suavemente, com o uso da função `fadeOut()`.

10. Teste essas funcionalidades no seu novo formulário.



6

Comunicação com o servidor



Mãos à obra!



Laboratório 1

A - Criando uma nova página contendo uma lista de estados e cidades

1. Na pasta **Laboratorios**, crie uma nova pasta chamada **Laboratorio06**;
2. Na pasta **Laboratorio06**, copie o conteúdo da pasta **Laboratorio05**;
3. Na pasta **Laboratorio06**, insira uma nova página chamada **cidades.html** na pasta **forms**;
4. Na pasta **scripts**, inclua um arquivo chamado **cidades.js**;
5. Planeje a página **cidades.html** de modo que tenha dois elementos `<select>`. O primeiro deverá representar uma lista de estados do Brasil, e o outro, uma lista de cidades de cada estado. Considere o seguinte:
 - Quando o formulário for carregado, a lista de estados deverá ser preenchida no primeiro elemento;
 - Assim que o usuário escolher um estado, a lista de cidades deverá aparecer no segundo elemento.

6. No arquivo **cidades.js**, elabore um código JavaScript que realize as seguintes tarefas:

- Definir uma estrutura baseada em JSON representando um array em que cada elemento possui duas informações, estado e cidade:

```
var estados = [  
    { "id": "1", "estado": "SP" },  
    { "id": "2", "estado": "RJ" },  
    { "id": "3", "estado": "MG" },  
    { "id": "4", "estado": "BA" },  
];  
  
var cidades = [  
    { "id": "1", "idestado": "1", "cidade": "CAMPINAS" },  
    { "id": "2", "idestado": "1", "cidade": "SOROCABA" },  
    { "id": "3", "idestado": "2", "cidade": "NITEROI" },  
    { "id": "4", "idestado": "2", "cidade": "CABO FRIO" },  
    { "id": "5", "idestado": "2", "cidade": "ANGRA" },  
    { "id": "6", "idestado": "3", "cidade": "BELO  
HORIZONTE"},  
    { "id": "7", "idestado": "3", "cidade": "BETIM" },  
    { "id": "8", "idestado": "3", "cidade": "EXTREMA" },  
    { "id": "9", "idestado": "4", "cidade": "SALVADOR" },  
    { "id": "10", "idestado": "4", "cidade": "PORTO  
SEGURO" },  
];
```

- Escrever uma função que, quando chamada na carga da página, realize a leitura desse conteúdo e inclua os estados, representados pela propriedade **ufs**;
- Escrever uma função que, tomando o estado como base, monte uma lista de cidades, juntamente com o respectivo elemento **<option>** a ser inserido dinamicamente no **<select>** existente;
- Prever um item no primeiro elemento (correspondente aos estados) com o conteúdo **TODOS**. Quando selecionado, deverá mostrar todas as cidades, independente do estado.



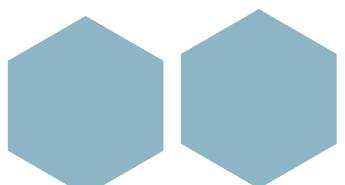
7

Suporte off-line



Mãos à obra!

SCALP
50.
907
104
907
ALTA
10007-67



Laboratório 1

A - Criando um formulário para inclusão de usuários

1. Na pasta **Laboratorios**, crie uma nova pasta chamada **Laboratorio07**;
2. Na pasta **Laboratorio07**, copie o conteúdo da pasta **Laboratorio06**;
3. Na pasta **Laboratorio07**, insira uma nova página chamada **usuarios.html** na pasta **forms**;
4. Na pasta **scripts**, inclua um arquivo chamado **usuarios.js**. O arquivo **usuarios.html** deverá ter um formulário contendo campos de entrada para o nome do usuário e uma senha, além de um botão para incluir um novo usuário;
5. No arquivo **usuarios.js**, escreva instruções para armazenar um usuário no objeto **localStorage**. As etapas para a elaboração desse código são as seguintes:
 - Leitura dos campos de entrada e armazenamento em variáveis;
 - Definição de chaves a serem associadas a cada variável do item anterior;
 - Utilização do objeto **localStorage** para armazenar as duas informações no browser;
 - Mensagem de confirmação de inclusão com sucesso ou erro, se for o caso.

B - Criando um formulário para login

1. Defina um novo formulário na pasta **forms**, só que, desta vez, para login. A página deve se chamar **login.html**. Os campos de entrada são essencialmente os mesmos usados para o cadastro de usuários;
2. Em seguida, no arquivo **usuarios.js**, escreva um conjunto de instruções que realizem a tarefa de validação de usuários. Essa tarefa deve ser formada pelas seguintes etapas:
 - Leitura dos campos de texto e armazenamento em variáveis;
 - Comparação dos campos de entrada lidos com os respectivos valores armazenados em **localStorage**;
 - Se os valores forem encontrados, armazená-los em **sessionStorage**.

3. Altere a página **formCurriculum.html** para que, quando carregada, verifique se existe algum dado armazenado em **sessionStorage** (o mesmo usado no item anterior). Se existir, apresenta a página normalmente. Se não existir, desabilita o botão que permite salvar um currículo.

C - Armazenando um currículo no banco de dados IndexedDb

1. Na pasta **scripts**, crie um arquivo chamado **database.js**;

2. No arquivo **database.js**, escreva um conjunto de instruções para armazenar as informações de um currículo no banco de dados **IndexedDb**. As etapas para elaboração dessas instruções são as seguintes:

- Ler as informações do formulário e armazená-las em variáveis;
- Definir o nome de um banco de dados;
- Abrir o banco de dados;
- Definir uma estrutura para o registro, o que, nos bancos de dados relacionais, é chamado de **tabela**;
- Por meio de uma estrutura baseada em **JSON**, tomar as variáveis lidas e persisti-las no registro;
- Substituir a mensagem que já havia neste formulário, mostrando todos os dados, por uma mensagem de confirmação ou de erro.

3. Teste as três páginas:

- Crie um usuário;
- Chame a tela de login. Se o usuário estiver validado, será direcionado para **formCurriculum**;
- Inclua alguns registros.

D - Listando os nomes das pessoas armazenadas no banco de dados

1. Na pasta **forms**, inclua um novo arquivo chamado **listaCurriculum.html** cujo conteúdo, além do título, é um elemento ****;

2. No arquivo **database.js**, complemente a informação necessária para buscar todos os registros armazenados e, em seguida, exibi-los na página. A exibição deverá contemplar a inclusão da informação em elementos ****, a serem inseridos dinamicamente no elemento ****.



8

PhoneGap

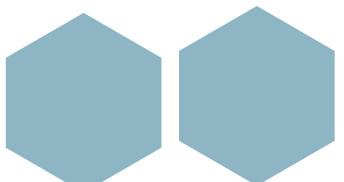


Mãos à obra!

SCALING ALTDAS
50°



Editora
IMPACTA



Laboratório 1

A - Gerando um aplicativo baseado em Android com o PhoneGap

Com base nos procedimentos indicados, elabore a aplicação. Este laboratório requer o uso do NodeJS ou do PhoneGap Builder. O último é a forma mais simples de testar, uma vez que o portal da Adobe disponibiliza um compilador para suas aplicações.

1. Usando instruções de linha de comando do Cordova, crie um projeto chamado **app**;
2. Defina a página inicial (**index.html**) com o mesmo conteúdo que você definiu na página **curriculum.html**;
3. Inclua o arquivo css necessário na pasta correspondente ao projeto Cordova que você criou;
4. Compacte a pasta **app**, cujo conteúdo é o seu projeto. Guarde esse arquivo compactado para uma etapa posterior;
5. Para este laboratório, vamos usar o PhoneGap Build para gerar nosso pacote. Para tanto, crie uma conta no site da Adobe, proprietária do PhoneGap. Siga os passos adiante:
 - Entre no site <<https://build.phonegap.com/>>;
 - Registre-se como novo usuário. Use a conta **free** (existem opções pagas);
 - Em seguida, realize o login;
 - Na página que surgir, clique na aba **private** (na aba open source você será direcionado a usar uma aplicação no GitHub). Como usuário free, é possível criar apenas uma conta **private**;
 - Faça o upload do projeto compactado. Quando estiver pronto, você terá a opção de fazer o download do pacote de instalação tanto para Android como para iOS.
6. Proceda com a instalação no seu dispositivo, preferencialmente, Android.