

PADRÕES WEB

- **Boas Práticas**
- **Responsividade**
- **HTML**
- **CSS**
- **JavaScript**

Autor: Grazielle Ferreira Antunes


Graduada em Sistemas de
Informação e Análise e
Desenvolvimento de Sistemas

Atua como Desenvolvedora Full
Stack na Teknisa com softwares
voltados para o ramo alimentício.






BOAS PRÁTICAS

- 
- Boa Semântica
 - Quebras de Linhas
 - Linguagem Clara
 - Uso de Tabelas no html
 - Indentação
 - Utilização de chaves
 - Inserir o script ao final da página
 - Declare variáveis fora do loop
 - Use ;
 - Comentários nos códigos



RESPONSIVIDADE

- 
- Responsividade se refere a disposição dos elementos e o conteúdo que se adaptam de acordo com o tamanho da tela do usuário. Isso significa que, independentemente do dispositivo que utilizar, o layout daquele website será carregado sem erros, mantendo a facilidade de se encontrar o que deseja, sempre com uma navegação simples e intuitiva;
 - Compatibilidade com novos dispositivos;
 - Menor taxa de rejeição;
 -



HTML

CONCEITOS

- HTML (HyperText Markup Language ou Linguagem de Marcação de HiperTexto) é uma linguagem de marcação e define o significado e a estrutura do conteúdo da web.
- "Hipertexto" refere-se aos links que conectam páginas da Web entre si, seja dentro de um único site ou entre sites. "Hipertexto" refere-se aos links que conectam páginas da Web entre si, seja dentro de um único site ou entre sites. Links são um aspecto fundamental da web. Ao carregar conteúdo na Internet e vinculá-lo a páginas criadas por outras pessoas, você se torna um participante ativo na world wide web.
- A marcação HTML inclui "elementos" especiais, como <head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, <div>, , , <audio>, <nav>, <video>, , , e muitos outros.



CSS

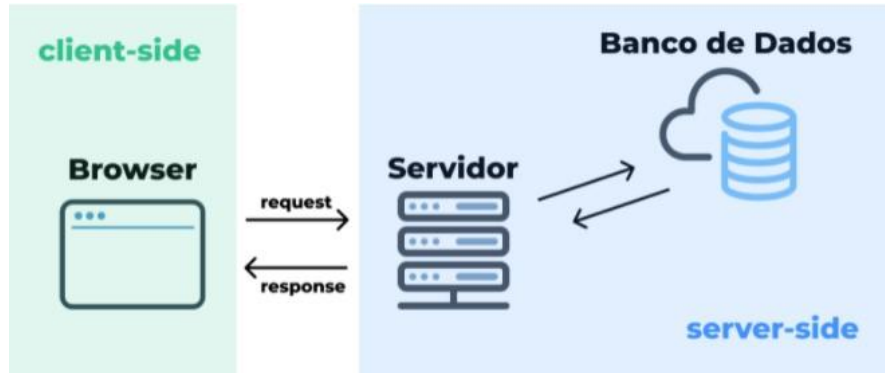
CONCEITOS

- CSS (Cascading Style Sheets ou Folhas de Estilo em Cascata) é uma linguagem de estilo usada para estilizar a apresentação de um documento escrito em HTML ou em XML.

Java Script

CONCEITOS

- Javascript, ou simplesmente JS, é uma linguagem de programação de uso geral muito popular entre os programadores, aplicada principalmente para desenvolvimento web e desenvolvimento de software.
- O Js atua na programação front-end, — a parte “visual” de uma aplicação (geralmente um site ou um app). Neste caso, o Javascript é usado junto com outras duas linguagens iniciais, o HTML e CSS.



- No seu navegador, que é o lado “do cliente”, o usuário, visualiza a parte estática (HTML + CSS) e também a parte dinâmica (Javascript). Toda a comunicação que clientes não vêem entre uma aplicação e, por exemplo, um banco de dados, é do “lado do servidor”.

Criando uma página web



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="author" content="Grazielle Ferreira" />
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Padrões Web</title>
</head>
<body>

</body>
</html>
```

- Vamos iniciar o conteúdo do site dentro do nosso body
- Criaremos um header no qual estará contido todo o nosso filtro
- Criaremos uma seção para este header
- Na nossa primeira div teremos o nosso input para pesquisar os desenvolvedores por nome

```
<header>
  <h1>Desenvolvedores</h1>
  <section>
    <div id="name">
      <label>Por nome:</label>
      <input id="nameSearch" placeholder="Digite aqui..."
type="text" />
    </div>
  </section>
</header>
```


- Ainda dentro da section e abaixo da div para pesquisa por nome criaremos a div de pesquisa por linguagem
- Note que usaremos outro tipo de input

```
<div id="language">
  <label>Por linguagem:</label>
  <div id="" condition>
    <label>
      <input id="1" type="radio" name="option" checked />
      <span>E</span>
    </label>
    <label>
      <input id="2" type="radio" name="option" />
      <span>OU</span>
    </label>
  </div>
</div>
```

- Ainda dentro da section e abaixo da div para pesquisa por linguagem criaremos a div de pesquisa por nome da linguagem
- Note que usaremos outro tipo de input
- Repita os labels para as linguagens PHP e Python (se atente aos nomes e ids)

```
<div id="languageOptions">
  <label>
    <input id="java" name="java" type="checkbox" checked />
    <span>Java</span>
  </label>
  <label>
    <input id="js" name="js" type="checkbox" checked />
    <span>JavaScript</span>
  </label>
</div>
```

- Vamos iniciar a estilização da nossa página web
- No arquivo style.css selecionamos a section que está dentro do nosso header e adicionamos a ele um display
- Vamos também selecionar o nosso elemento label e estilizar a fonte dele.

```
header > section {  
    display: flex;  
}  
  
label {  
    font-weight: 600;  
    font-size: 14px;  
}
```

- Criamos uma estilização para o elemento cuja div tenha o id name e a div cujo id seja nameSearch
- Estes elementos se referem ao nosso bloco de pesquisa por nome

```
#name {  
  flex: 2;  
  max-width: 70%;  
}  
  
#nameSearch {  
  border-radius: 2px;  
  width: 100%;  
  height: 5vh;  
  margin-top: 8px;  
}
```

- Criamos uma estilização para o elemento cuja div tenha o id language e a div cujo id seja condition
- Estes elementos se referem ao nosso bloco de pesquisa por linguagem

```
#language {  
  display: flex;  
  align-items: center;  
  flex-direction: column;  
}  
  
#condition {  
  display: flex;  
  height: 100%;  
  align-items: center;  
  margin-top: 8px;  
}
```

- Criamos uma estilização para o elemento cuja div tenha o id languageOptions
- Estes elementos se referem ao nosso bloco de pesquisa por nome da linguagem.
- E por último utilizaremos padding para espaçar entre uma div e outra

```
#languageOptions {  
  display: flex;  
  align-items: center;  
  max-width: 180px;  
  min-height: 100%;  
  flex-wrap: wrap;  
  margin: 1.2% 0px;  
}  
  
section > div:not(:first-child) {  
  padding: 0 20px;  
}
```

- Voltando ao html criaremos uma nova section abaixo do header.
- Sendo assim, já incluímos dentro dela uma div que exibirá os resultados.



```
<section class="result">
  <div class="countResults">
    <span id="countResult"></span>
    <span>Resultados</span>
  </div>
</section>
```

- Estilizamos a classe que define a nossa section colocando uma cor de fundo em um tom de cinza.
- Em seguida definimos um estilo para a nossa div que exibe o número de resultados.



```
.result {  
  background-color: #EDEDED;  
}  
  
.countResults {  
  padding: 20px 30px 10px;  
  font-weight: bold;  
  font-size: 1.3rem;  
}
```


- Retornamos ao html e inserimos uma div container dentro da mesma section. Nesta div criaremos todos os cards dos desenvolvedores.
- Logo criaremos uma div com a classe card e o id de cada desenvolvedor.




```
<div class="container">  
  <div class="card" id="36392ffd-052e-41cf-a3f5-e574bef212b2">  
  </div>  
</div>
```

- Adicione uma div que contenha a imagem de perfil do desenvolvedor.
- Esta imagem tem que estar dentro da pasta images criada anteriormente.



```
<div class="profilePicture">  
    
</div>
```

- Ainda dentro da div de card crie uma div para a descrição do perfil desse desenvolvedor conforme o próximo slide



```
<div class="profileDescription">
  <div class="info">
    <h3 class="card-title" name="username">Dulcilene
Farias</h3>
    <small name="age">53</small>
    <small name="mail">dulcilene.farias@example.com</small>
    <span name="phone" class="invisible">(99) 1116-
1841</span>
    <span name="github"
class="invisible">https://github.com/greenwolf425</span>
    <span name="description" class="invisible">Graduado em
Redes de Computadores pela COTEMIG</span>
    <span name="descriptionAbility"
class="invisible">Experiências: - Mobile - Análise de Sistemas -
DevOps - Programação Web - Adm. de Banco de Dados (DBA) -
Gerência de Projeto </span>
  </div>
</div>
```

- Dentro da div profileDescription adicione mais uma div.
- Esta div irá exibir os ícones das linguagens de cada programador

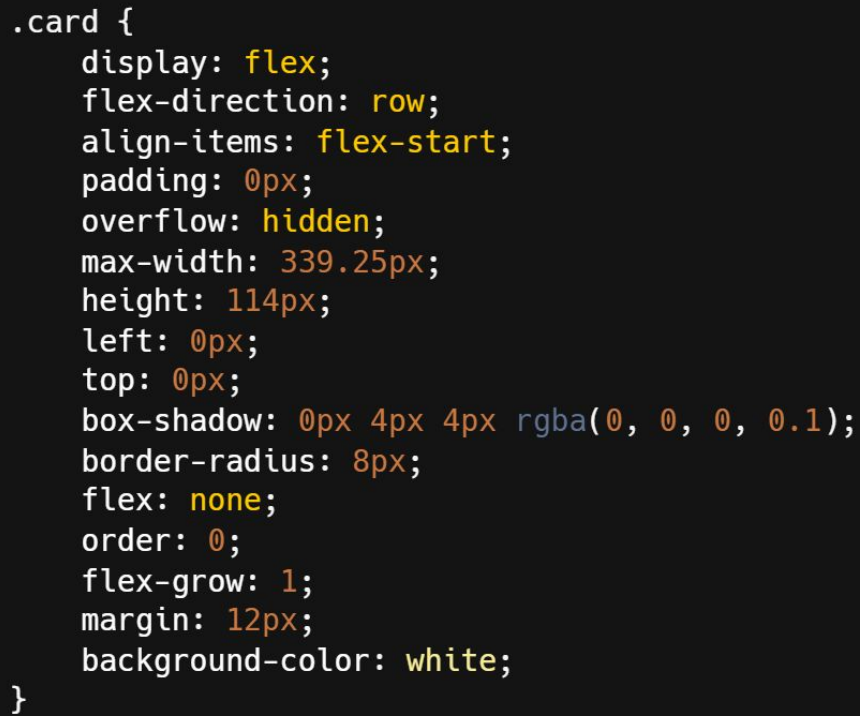
```
<div class="languages">
  
  
  
</div>
```

- Retorne ao css e adicione a estilização da classe container no que se refere ao container de cards



```
.container {  
  display: inline-flex;  
  flex-direction: row;  
  flex-wrap: wrap;  
}
```

- Adicione também a estilização para criação visual do próprio card conforme próximo slide



```
.card {  
  display: flex;  
  flex-direction: row;  
  align-items: flex-start;  
  padding: 0px;  
  overflow: hidden;  
  max-width: 339.25px;  
  height: 114px;  
  left: 0px;  
  top: 0px;  
  box-shadow: 0px 4px 4px rgba(0, 0, 0, 0.1);  
  border-radius: 8px;  
  flex: none;  
  order: 0;  
  flex-grow: 1;  
  margin: 12px;  
  background-color: white;  
}
```


- Adicione a classe invisible e oculte os itens que devem ser invisíveis na tela principal.
- Estilize também as informações do perfil do desenvolvedor.



```
.invisible {  
  display: none;  
}  
.profileDescription {  
  display: flex;  
  flex-direction: column;  
  height: 100%;  
  width: 100%;  
  justify-content: space-around;  
  padding-left: 15px;  
}
```

- Adicione uma estilização para a classe info.
- Retire as margens dos ícones e defina o tamanho padrão.
- Adicione uma estilização em h3 para o nome do desenvolvedor.


```
.info {  
  display: flex;  
  flex-direction: column;  
}  
.iconLanguage {  
  height: 24px;  
  margin: 0px;  
}  
h3 {  
  color: #333333;  
  flex: none;  
  margin: 0px;  
}
```

- Ao final do arquivo ainda dentro do body insira a chamada do seu script.
- Adicione um evento de onload no seu body. Ele será disparado assim qye o usuário entrar na tela chamando a função preRender().



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

```
<body onload="preRender( )">
```

- No app.js crie uma função preRender() que irá executar outras duas funções.
 - A função getCountVisibleCards() vai retornar a quantidade de cards visíveis.
 - A função updateResults() vai atualizar essa informação no textContent do span correspondente.
- 



```
function preRender() {  
  let countVisibleCards = getCountVisibleCards();  
  updateResults(countVisibleCards);  
}  
  
function getCountVisibleCards() {  
  return  
  Array.from(document.getElementsByClassName("card")).filter((card)  
=> !card.style.display || card.style.display !== "none").length;  
}  
  
function updateResults(count) {  
  document.getElementById("count-result").textContent = count;  
}
```

- No input de pesquisa por nome inclua o evento `oninput` chamando a função `filter()`, este evento irá disparar toda vez que o usuário digitar algo dentro do input.
- Nos demais elementos do tipo `radio` e `checkbox` inclua o evento de `onchange` também chamando a função `filter`, que por sua vez será acionada toda vez que o usuário fizer alguma alteração nestes campos



```
<input id="nameSearch" placeholder="Digite aqui..." type="text"
oninput="filter()" />
```

```
<input id="1" type="radio" name="option" onchange="filter()"
checked />
```

```
<input id="java" name="java" type="checkbox" onchange="filter()"
checked />
```

- Vamos iniciar a função que irá filtrar os dados.
- Cria a função no js e declare variáveis que serão retornadas da função `getFilterProperties()` que será criada a seguir.



```
function filter() {  
  let { search, operation, languages } = getFilterProperties();  
}
```

- Por ora declaramos uma variável que receberá o retorno de uma outra função a ser criada, a `getSearchValue()`..
- Na função `getSearchValue` declaramos uma variável que receberá os caracteres digitados pelo usuário e logo em seguida retornará para a variável declarada na função anterior.



```
function getFilterProperties() {  
    let search = getSearchValue();  
}  
  
function getSearchValue() {  
    let inputSearchEl = document.getElementById( "nameSearch" );  
    return inputSearchEl.value;  
}
```


- Vamos incrementar a nossa função `getFilterProperties` adicionando também a chamada da função `getSelectedRadio()` para verificar qual radio está selecionado.
- Na função `getSelectedRadio()` retornamos o elemento cujo radio está checado.
- Com o retorno desta função faremos uma validação para que, de acordo com o id saibamos se a operação será de AND ou OR.



```
function getFilterProperties() {  
    let search = getSearchValue();  
    let [RADIO] = getSelectedRadio();  
    let operation = RADIO.id == "1" ? "AND" : "OR";  
}  
  
function getSelectedRadio() {  
    return Array.from(document.querySelectorAll( 'header  
input[type="radio"]:checked' ));  
}
```

- Por último acionaremos a função `getSelectedLanguages()` para que ela retorne todas as linguagens que foram checadas.
- Logo retornaremos as 3 variáveis em sequência assim como declaramos na função inicial `filter()`.

```
function getFilterProperties() {  
  ...  
  let languages = Array.from(getSelectedLanguages()).map((lang)  
=> lang.name);  
  return {  
    search,  
    operation,  
    languages,  
  };  
}  
  
function getSelectedLanguages() {  
  return Array.from(document.querySelectorAll('header  
input[type="checkbox"]:checked'));  
}
```

- Na função filter criaremos um setInterval() para que todo o código que estiver dentro dela seja executado se, e somente se, o usuário não fizer nenhuma alteração no intervalo de 800 milissegundos



```
function filter() {  
  let { search, operation, languages } = getFilterProperties();  
  let interval = setInterval((_) => {  
  
    }, 800);  
}
```

- Dentro do `setInterval` vamos declarar uma variável guardando o o nosso elemento cuja classe tem o nome de `container`.
- Em outra variável vamos fazer uma comparação entre o que o usuário digitou no início da função na qual já tinha sido armazenado em variável e o que temos de `string` neste momento em que o código dentro do `setInterval` está sendo executado chamando novamente o `getSearchValue()`.
- Se não houve modificação limpamos o `interval`.
- Verificamos se o meu elemento de classe `container` existe e se ele possui filhos que seriam os nossos `cards` e se não houve alteração para então chamar uma função que será criada para `updatar` a visibilidade dos `cards`.





```
let interval = setInterval((_) => {  
  let [container_el] =  
document.getElementsByClassName( "container");  
  let changedText = search !== getSearchValue();  
  if (!changedText) clearInterval(interval);  
  if (container_el && container_el.children && !changedText) {  
    let visibleCards = updateVisibleCards(container_el, search,  
operation, languages);  
  }  
}, 8000);
```

- Nesta função a ser criada vamos iterar sob os nossos cards para fazer uma comparação entre os dados filtrados e os dados que cada card possui. Para isso vamos verificar os nossos 3 inputs;
- No decorrer criaremos uma função auxiliar para verificar se as informações filtradas e contidas nos cards deram match.



```
function updateVisibleCards(container_el, search, operation,
selectedLanguages) {
  let visibleCards = 0;
  Array.from(container_el.children).forEach((card_el) => {
    let [title_el] = card_el.getElementsByClassName("card-
title");
    let cardLanguages =
Array.from(card_el.getElementsByClassName("iconLanguage")).map((ima
ge) => image.name);
  });
};
```