

04
Lazy Load

Vamos nos atentar no site! Se observarmos o *console* podemos verificar o `css`, blocante, logo abaixo do "html". Ou seja, é algo que está priorizado. Podemos verificar, dando um *scroll*, que temos o download de um tal de "99185150" cujo domínio é "player.vimeo.com". Esse *request* corresponde ao seguinte vídeo:

Name	Status	Domain	Type	Initiator	Size	Time
categorias-prateleira333.svg	200	localhost	svg+xml	[index]:3	2.9KB	30ms
opensans-regular.woff2	200	localhost	font	[index]:1	13.8KB	38ms
opensans-extrabold.woff2	200	localhost	font	[index]:1	14.4KB	32ms
opensans-bold.woff2	200	localhost	font	[index]:1	14.2KB	37ms
opensans-light.woff2	200	localhost	font	[index]:1	13.7KB	37ms
opensans-italic.woff2	200	localhost	font	[index]:1	15.2KB	40ms
99185150	200	player.vimeo.com	document	[index]:14	9.0KB	209ms
curso-design-editorial-2eb5cccd876.svg	200	localhost	svg+xml	[index]:29	796B	13ms
curso-logica-de-programacao-45655ad0...	200	https://player.vimeo.com/video/99185150	svg+xml	[index]:31	1.0KB	32ms
background-cidade-fundo-5d20ac8d24.svg	200	localhost	svg+xml	[index]:1	5.7KB	34ms
background-cidade-nuvem-c4cb868c3d.svg	200	localhost	svg+xml	[index]:1	649B	33ms

69 requests | 1.3MB transferred | Finish: 1.3min | DOMContentLoaded: 82ms | Load: 1.01s

Esse vídeo, entretanto, não é tão importante para ser executado antes do *background*. É uma questão de prioridade. Podemos adiar um pouco a execução desse vídeo também. Vamos buscar no *Sublime* onde está o vídeo, podemos ver que ele na verdade é um "iframe". Vamos fazer com que esse "iframe" seja carregado depois de 3 segundos e, assim, priorizaremos a renderização de outras coisas.

Vamos dar um "Ctrl + X" para copiar esse código e vamos criar um arquivo nome que se chamará "video.js" e colocaremos um `setTimeout` e com o código que copiamos dentro de um `var frame`. Queremos colocar o `iframe` dentro do `div class="elasticMedia-container"`, para isso digitaremos:

```
document.querySelector('.elasticMedia-container').innerHTML = iframe
```

Teremos o seguinte:

```

1 setTimeout(function(){
2
3     var iframe = '<iframe class="elasticMedia" src="https://player.vi'
4
5     document.querySelector('.elasticMedia-container').innerHTML = ifr
6
7 }, 1500);

```

Falta carregar o arquivo no "index.html" digitando

```
<script async src="assets/js/video.js"></script> :
```

Teremos:

```
<!--build:js assets/js/scripts.js async -->
<script async src="assets/js/menu.js"></script>
<script async src="assets/js/busca.js"></script>
<script async src="assets/js/detect.js"></script>
<script async src="assets/js/footer.js"></script>
<script async src="assets/js/video.js"></script>
<!-- endbuild -->
```

Dando um `gulp default` veremos que funciona. Testando no site veremos que estará tudo certo! O interessante de fazer isso é que o usuário primeiro se concentrará no início da página e depois ele irá dar um *scroll* e visualizar o vídeo, o que não acarreta em nenhum problema.

Estamos forçando o carregamento assíncrono porque ele seria blocante se fosse síncrono. E podemos adiar tanto um *script* quanto um *iframe* (mesmo que ele seja não blocante, como foi o caso aqui). Se esses itens competem por recursos de execução eles não são prioridade, eles podem, portanto, serem atrasados.

A maior lição, aqui, é perceber que as vezes adiar as coisas é melhor, mesmo que o tempo total de carregamento da página seja maior, a sensação do usuário de rapidez é aumentada e isso é que importa. Poderíamos, ainda, escolher uma estratégia híbrida, ao em vez de carregar direto um vídeo, podíamos carregar uma imagem *fake* para dar a ilusão de que o vídeo está carregado e quando apertássemos o *play* é que ele carregaria efetivamente.

Lazy load

Temos diversos ícones que podem ser submetidos ao que chamamos de *lazy load*:

Cursos mais feitos				
 Lógica de Programação I: comece na carreira com JavaScript	20 horas	Básico	+	
 HTML e CSS: Os seus primeiros passos	20 horas	Básico	+	
 Java I: Primeiros passos	12 horas	Básico	+	
 HTML5 e CSS3 I: Suas primeiras páginas da Web	32 horas	Básico	+	
 Cordova & PhoneGap: Apps mobile com HTML, CSS e JS	20 horas	Intermediário	+	
 Java II: Orientação a Objetos	12 horas	Básico	+	

Isso significa que primeiro se prioriza o carregamento do topo da página e depois se carrega o que vêm na sequência, quando o usuário estiver *scrollando* a página. Existem vários *frameworks* no mercado que são focados em fazer o *lazy load* das imagens.

Bom, queremos fazer o *lazy load* com os ícones que possuem "curso-", mas se não queremos que o *browser* faça o download inicial, não podemos colocar ele no "src" da imagem.

Uma ideia é fazer um *replace* para substituir o "src" por algum outro atributo, nós usaremos o "data-src". Daremos um *replace all* e ele vai substituir, no caso, as 16 imagens.

```
index.html — performance-web-2
index.html
289 <div class="cursoCard-infos">
290   <div class="cursoCard-infos-tempoEstimado">
291     <p class="cursoCard-infos-tempoEstimado">1 hora</p>
292   </div>
293   <div class="cursoCard-infos-dificuldade">
294     <p class="cursoCard-infos-dificuldade">Fácil</p>
295   </div>
296   <a class="cursoCard" href="https://www.onlinesf.com.br/primefaces">
297     <div class="cursoCard-firstRow">
298       
299       <div class="cursoCard-icon">
300         <div class="cursoCard-nome">Java e JSF 2 – II: Componentes ricos com Primefaces</div>
301       </div>
    </div>
  </div>

```

Se olharmos o navegador perceberemos que retiramos todos os nossos ícones da página:

The screenshot shows a grid of course cards. Each card includes a thumbnail, the course title, duration, level, and a '+' button.

- Marketing digital: Introdução aos canais não pagos**: 3 horas, Básico
- Marketing Digital: Introdução aos canais pagos**: 7 horas, Básico
- Java e JSF 2 - II: Componentes ricos com Primefaces**: 12 horas, Intermediário
- Java e JSF 2 - III: Integração com CDI**: 16 horas, Intermediário
- Java e JPA: Otimizações com JPA2 e Hibernate**: 12 horas, Avançado
- Inglês: Candidate-se a vagas no exterior**: 4 horas, Básico
- Android II: Integração com apps e recursos do device**: 8 horas, Básico
- Design editorial: Criação de materiais gráficos**: 8 horas, Básico

Como a ideia é que as imagens só sejam carregadas quando o usuário fizer o *scroll* vamos trabalhar com esse aspecto do site. Vamos no *Sublime* e criaremos uma nova pasta chamada "lazyload.js" e vamos no "index.html" e colocaremos mais uma *lazyload.js*:

```
<!--build:js assets/js/scripts.js async -->
<script async src="assets/js/menu.js"></script>
<script async src="assets/js/busca.js"></script>
<script async src="assets/js/detect.js"></script>
<script async src="assets/js/footer.js"></script>
<script async src="assets/js/video.js"></script>
<script async src="assets/js/lazyload.js"></script>
<!-- endbuild -->
```

Vamos escrever um comando, no arquivo "lazyload", que serve para nos dizer nossa posição de *scroll* na página:

```
window.onscroll = function() {

    console.log('Scroll', scrollY);
}
```

Precisamos saber nossa posição para calcularmos direitinho o que escreveremos no comando. Voltando no *Sublime* podemos apagar o *console.log* e pensar o que precisamos. Precisamos carregar as imagens que que possuem o atributo *data-src* e para cada uma dessas imagens precisamos verificar se a imagem está visível ou não e, por fim, para sabermos que a imagem está visível utilizaremos o método *getBoundingClientRect*. Com isso poderemos verificar no console a posição das imagens, em pixels, em relação ao topo da janela, conforme vamos baixando na página, a imagem vai se aproximando a "0". O que queremos fazer, também, é carregar a imagem logo antes do momento em que ela fica visível. O tamanho da janela conseguimos descobrir usando o *window.innerHeight*. Para que isso aconteça vamos dizer que queremos trocar o *data-src* pelo *src*. Vamos ter o seguinte no "lazyload.js":

```
window.onscroll = function() {

    var imgs = document.querySelectorAll('img[data-src]');

    for (var i = 0; i < imgs.length; i++) {

        if (imgs[i].getBoundingClientRect().top < window.innerHeight) {
            imgs[i].src = imgs[i].getAttribute('data-src')
```

```

        }
    }
};
```

Se atualizarmos nossa página veremos que tudo estará certo. O usuário nem se dará conta do carregamento das imagens.

As vezes, conforme estamos fazendo o *download* e damos um *scroll* rápido na página, podemos verificar a imagem piscando o que dá uma sensação estranha, assim, podemos alterar o carregamento para que ele aconteça quando estivermos próximo a imagem e não, exatamente, quando bater nela. Para isso basta alterar o valor, quando o *scroll* estiver a 200 pixels de distância da imagem, vamos escrever o seguinte:

```
< window.innerHeight + 200
```

E, ao todo:

```
window.onscroll = function() {

    var imgs = document.querySelectorAll('img[data-src]');

    for (var i = 0; i < imgs.length; i++) {

        if (imgs[i].getBoundingClientRect().top < window.innerHeight + 200) {
            imgs[i].src = imgs[i].getAttribute('data-src')

        }
    }
};
```

Se você quiser, pode verificar no *Console* o carregamento das imagens. Essa é a técnica de *lazyload*. Esse código não é o mais otimizado que existe, ele pode, inclusive, ser melhorado. E se preferir, você pode utilizar *frames* já prontos para fazer isso. O princípio a ser apresentado aqui é economizar *requests* iniciais e deixar a rede inicial para baixar outros *request* mais importantes.

Throttle

Vamos observar um famoso gargalo que ocorre quando temos o *scroll*. Um dos grandes problemas é fazer coisas pesadas no *onscroll* pois ele dispara em uma frequência muito alta. Vamos usar o `console.log ('scroll');` para verificar a quantidade de vezes que ele é chamado:

```
window.onscroll = function() {

    console.log ('scroll');

    var imgs = document.querySelectorAll('img[data-src]');

    for (var i = 0; i < imgs.length; i++) {

        if (imgs[i].getBoundingClientRect().top < window.innerHeight + 200) {
            imgs[i].src = imgs[i].getAttribute('data-src')

        }
    }
};
```

```

    }
};
```

Observe o Console:

localhost:2020

Esse e muitos outros alunos conheceram e recomendaram. Invista na s

MATRICULE-SE JÁ

Console

426 scroll

Portanto, se temos dentro do `onscroll` um código lento podemos ter um grande gargalo de performance. A solução é um truque. O que não podemos fazer é executar o código a todo momento que o `onscroll` disparar. O que precisamos fazer é ignorar algumas das chamadas do `scroll` e delimitar o número de execução do código, por exemplo, a cada 100 miles de segundo.

No "lazyload.js" vamos acrescentar uma variável nova que vai controlar se já executamos um determinado código ou não, ela é a `jarodei = false`. Se já tivermos rodado, não queremos que isso se repita, e por isso diremos `if(jarodei) return` e abaixo disso teremos o `jarodei= true`. Assim, ele começará com o `false` e a primeira vez que ele rodar o código ficará `true`. Se deixarmos dessa maneira parece que ele rodará apenas uma vez, o que precisamos fazer, o que precisamos dizer é que depois de alguns miles de segundo, deve ser rodado de novo, assim, usaremos um `setTimeout` que rode em 100 miles segundo. Essa técnica chamamos no `javascript` de `throttle` e isso significa que estamos limitando a execução. Teremos o seguinte:

```

var jarodei = false;

window.onscroll = function() {

    if (jarodei) return;
    jarodei = true;
    setTimeout(function(){
        jarodei = false;
    }, 100);
}
```

Na prática o que fizemos foi dizer para rodar a cada 100 miles segundos. Se testarmos no navegador estará funcionando conforme queremos. Essa foi mais uma dica usando uma tática avançada de *javascript*. Se você utilizar um *frame* de *lazyload* provavelmente ele já irá trazer esse tipo de estratégia.