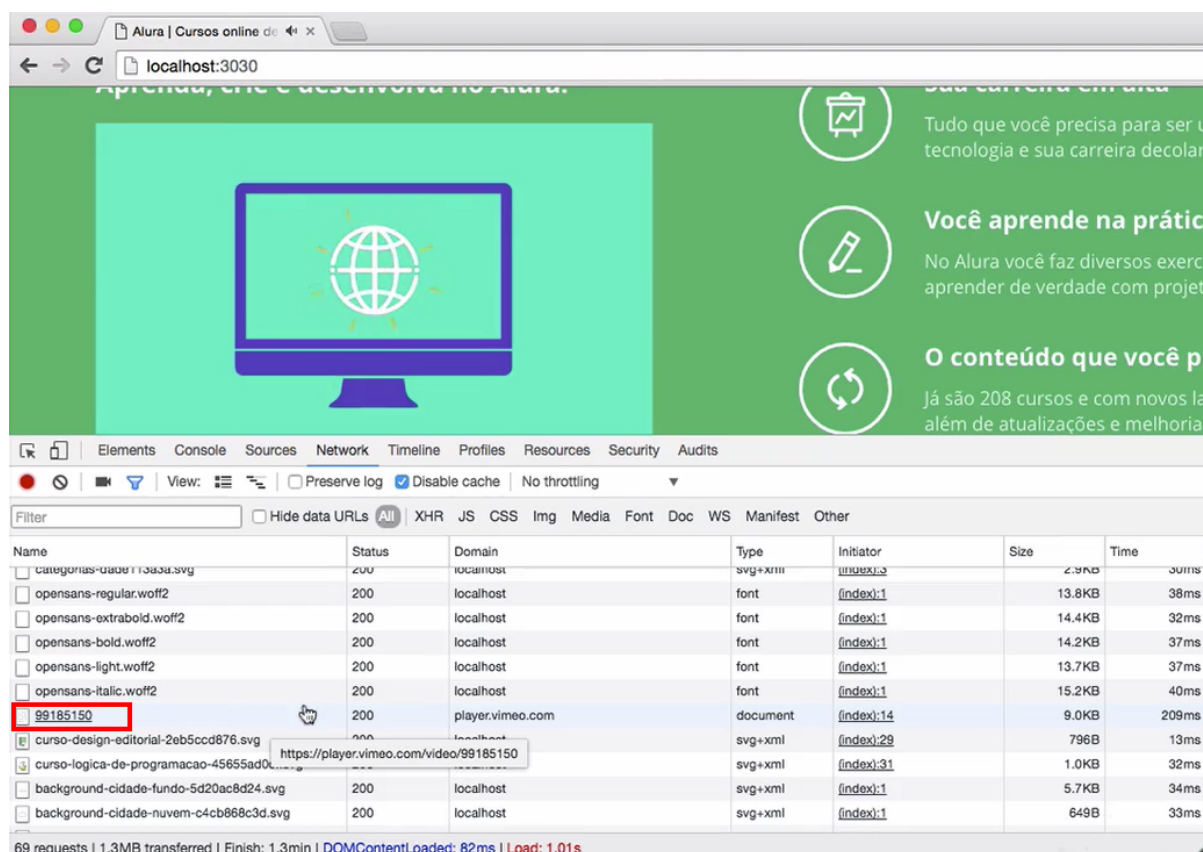


Performance Web II: Critical Path, HTTP/2 e Resource Hints: Aula 3

Vamos nos atentar no site! Se observarmos o *console* podemos verificar o `css`, bloqueante, 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:



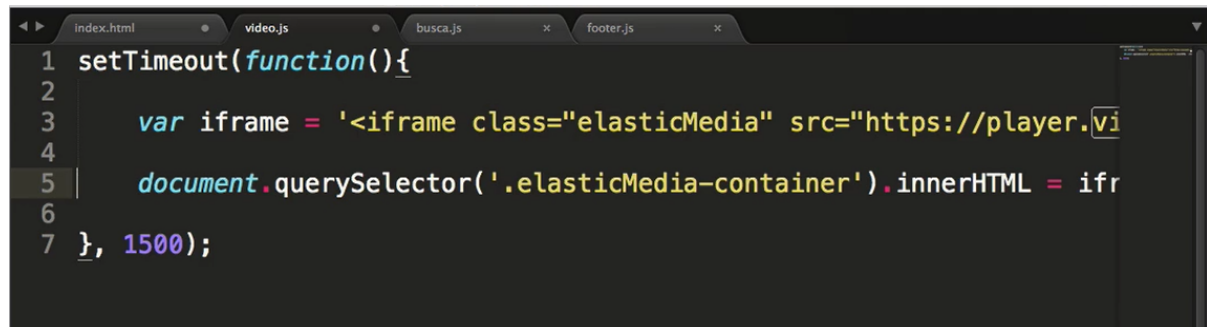
Esse vídeo, entretanto, não é tão importante para ser executado antes do *background*. É uma questão de prioridade. Podemos adiar um pouco a execução desse vídeo também. Vamos buscar no *Sublime* onde está o vídeo, podemos ver que ele na verdade é um "iframe". Vamos fazer com que esse "iframe" seja carregado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

A screenshot of a code editor with four tabs: 'index.html', 'video.js', 'busca.js', and 'footer.js'. The 'video.js' tab is active, showing a JavaScript function wrapped in a setTimeout. The code is as follows:

```
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 -->
```

Copiar código

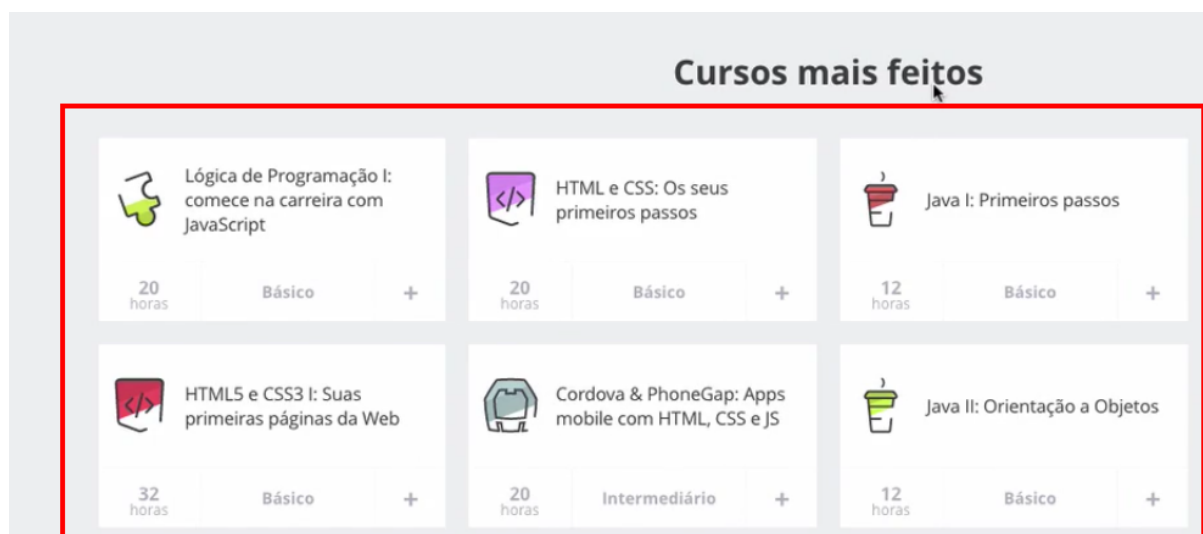
Dando um `gulp default` veremos que funciona. Testando no site veremos que estará tudo certo! O interessante de fazer isso é que o usuário primeiro se concentrará no início da página e depois ele irá dar um *scroll* e visualizar o vídeo, o que não acarreta em nenhum problema.

Estamos forçando o carregamento assíncrono porque ele seria bloqueante se fosse síncrono. E podemos adiar tanto um *script* quanto um *iframe* (mesmo que ele seja não bloqueante, 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

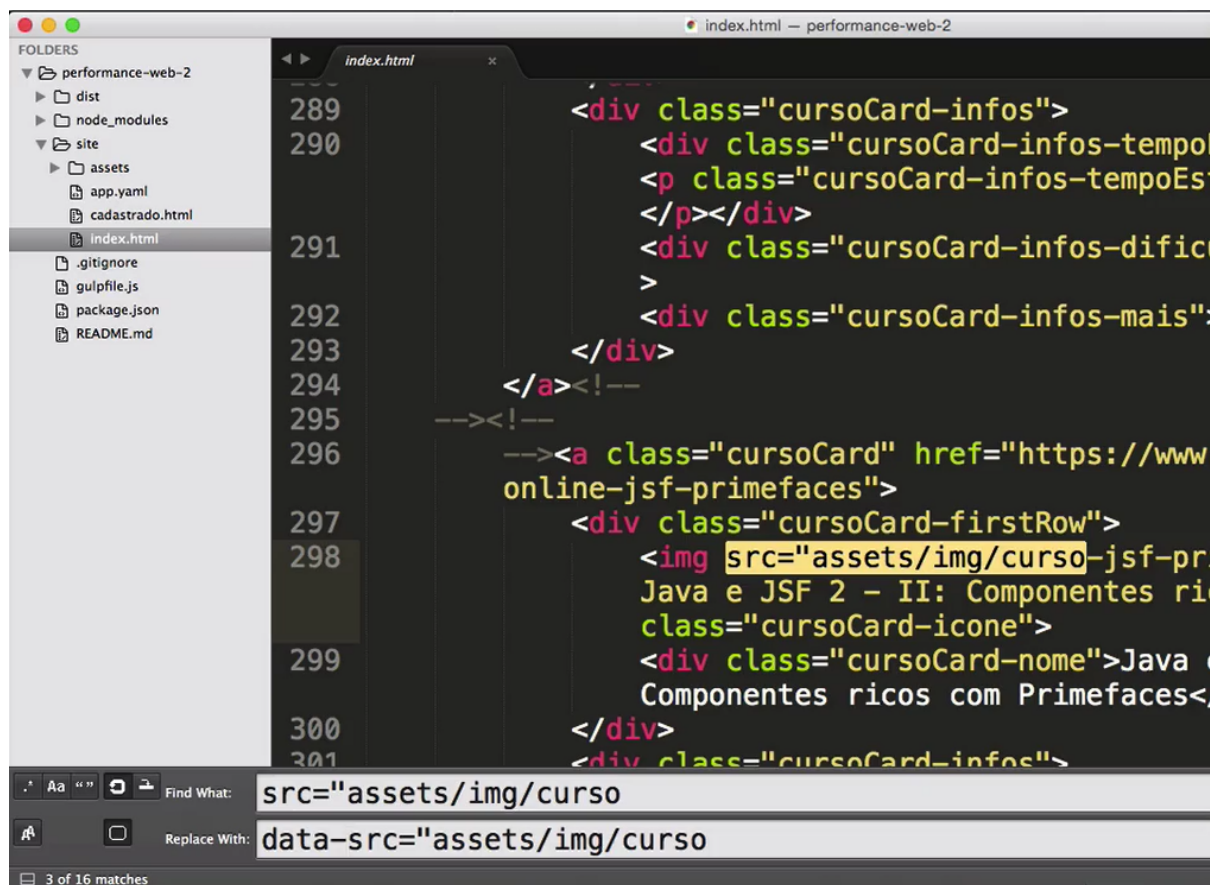
Temos diversos ícones que podem ser submetidos ao que chamamos de *lazy load*:



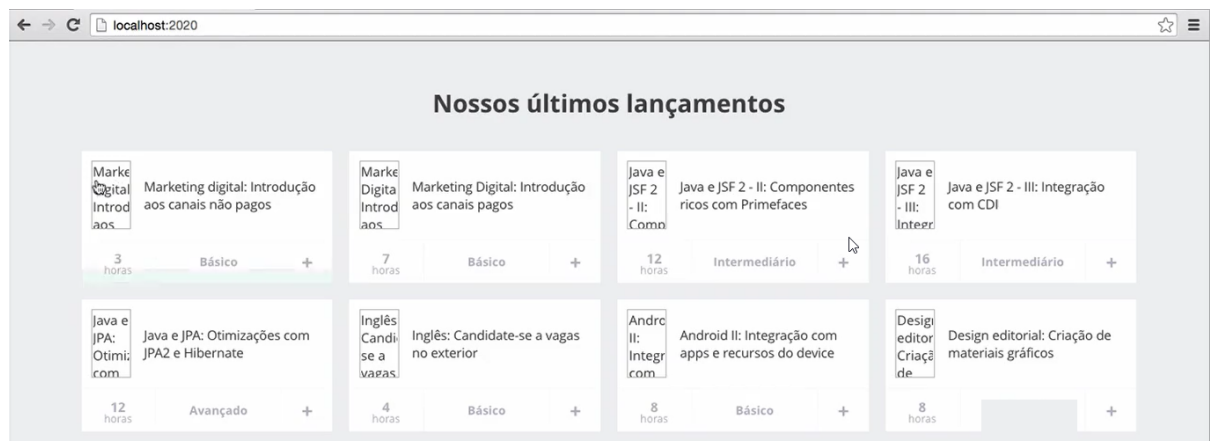
Isso significa que primeiro se prioriza o carregamento do topo da página e depois se carrega o que vêm na sequência, quando o usuário estiver *scrollando* a página. Existem vários *frameworks* no mercado que são focados em fazer o *lazy load* das imagens.

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.



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



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/lazyload.js"></script>  
<!-- endbuild -->
```

[Copiar código](#)

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);  
}
```

[Copiar código](#)

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 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 `getBound`. 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')  
        }  
    }  
};
```

[Copiar código](#)

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  
            imgs[i].src = imgs[i].getAttribute('data-src')  
  
        }  
    }  
};
```

Copiar código

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`

```

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)
            imgs[i].src = imgs[i].getAttribute('data-src')

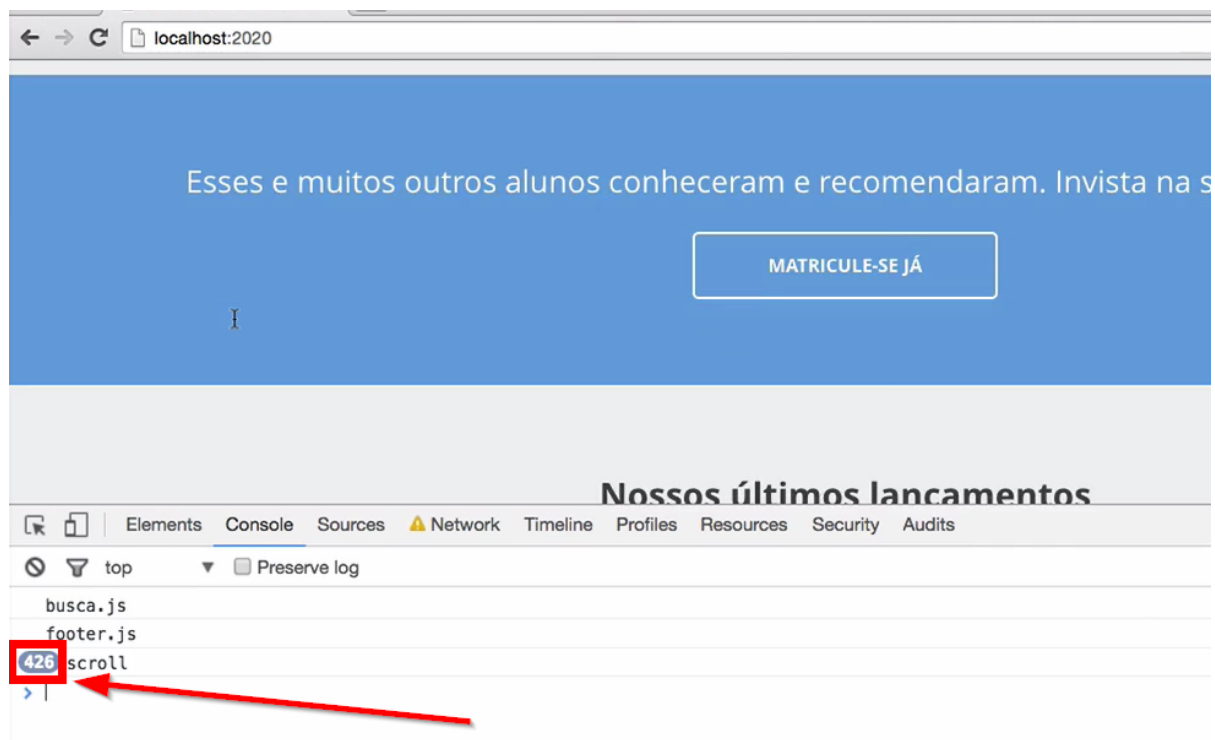
    }

};

```

Copiar código

Observe o Console:



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 milésimos 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 milissegundos, deve ser rodado de novo, assim, usaremos um `setTimeout` que rode em 100 milissegundos. 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);
}
```

[Copiar código](#)

Na prática o que fizemos foi dizer para rodar a cada 100 milissegundos. 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.