

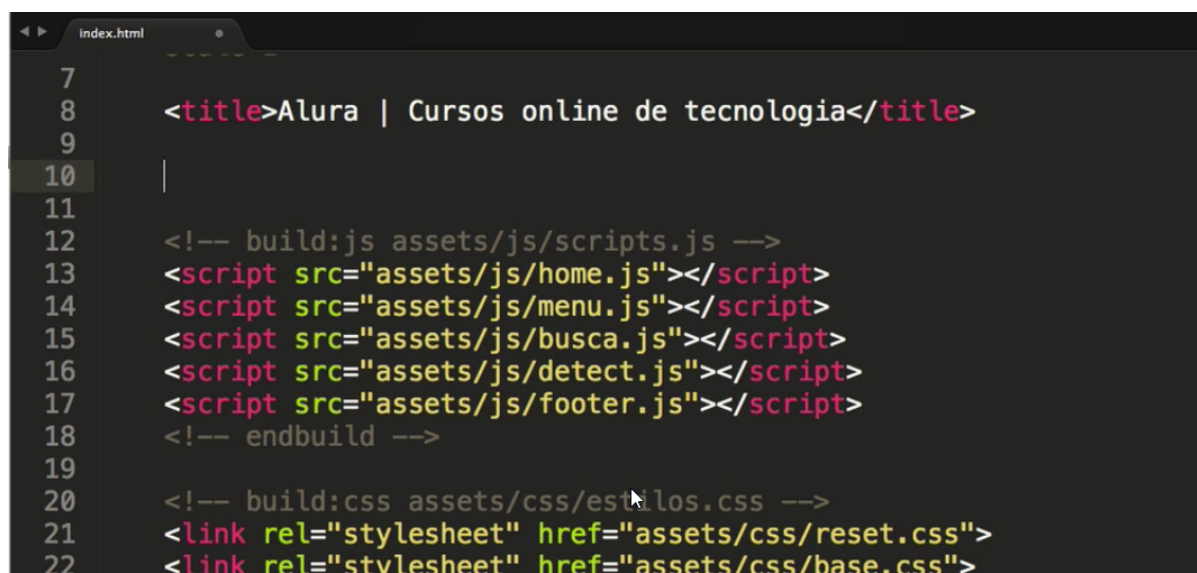
Performance Web I: otimizando o front-end: Aula 6 - Atividade 6 Transcrição das aulas | Alura

7-9 minutes

Fizemos várias otimizações e concatenamos diversas coisas e ficamos com cerca de 54 *requests*. A pergunta é: Será que podemos ir além?

Sim!

Quando falamos de um *html*, aquele inicial da página, ele também permite colocarmos *css* e *scripts* embutidos na página. Vamos tentar explicar melhor isso. Se abrirmos o nosso *html* veremos que temos uma série de *scripts* e *css* apontados em arquivos externos:



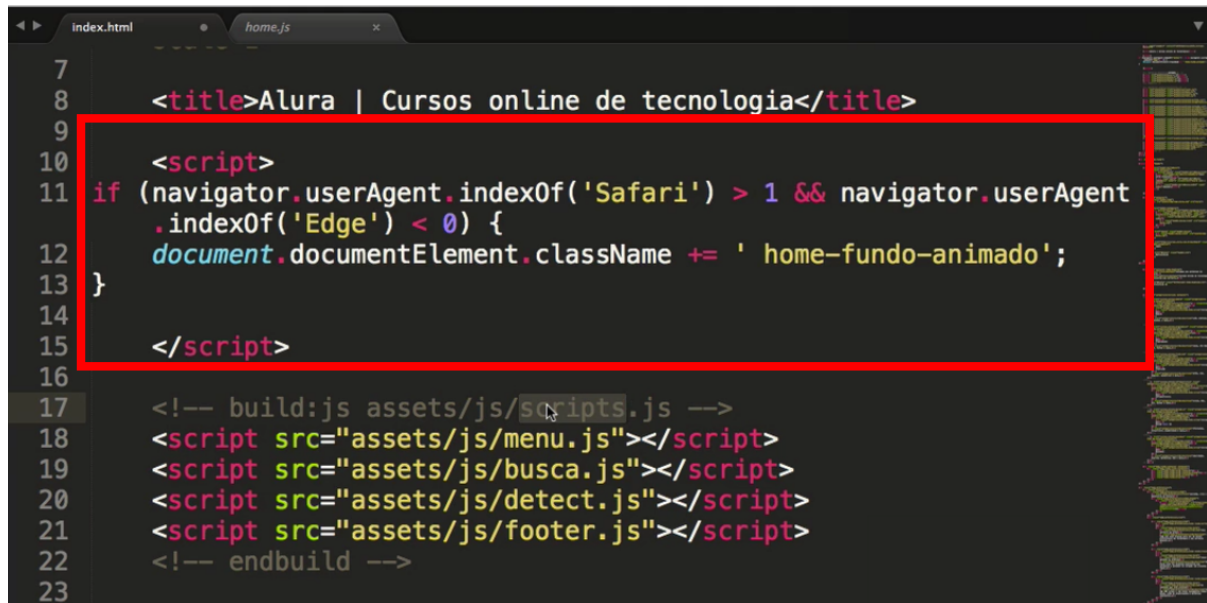
```
7
8 <title>Alura | Cursos online de tecnologia</title>
9
10 |
11
12 <!-- build:js assets/js/scripts.js -->
13 <script src="assets/js/home.js"></script>
14 <script src="assets/js/menu.js"></script>
15 <script src="assets/js/busca.js"></script>
16 <script src="assets/js/detect.js"></script>
17 <script src="assets/js/footer.js"></script>
18 <!-- endbuild -->
19
20 <!-- build:css assets/css/estilos.css -->
21 <link rel="stylesheet" href="assets/css/reset.css">
22 <link rel="stylesheet" href="assets/css/base.css">
```

Não precisaria, necessariamente ser assim, poderíamos colocar um *script* local, ou, como costumamos falar *in line*. A vantagem de fazer isso ao em vez de baixar um *script* com a *tag script* é que não teremos um *request* adicional, afinal, ele já vêm embutido no *html*.

Esse é um truque que podemos utilizar quando pretendemos diminuir a quantidade de *requests*. Claro, não é o caso de colocar o *jquery*, por exemplo, que tornaria o arquivo muito grande, mas podemos colocar alguns *scripts* que são arquivos pequenos que não valem o custo de um *request* novo. Lembre-se que quanto maior a quantidade de *requests* corremos o risco de ter um encavalamento e além disso, o custo de fazer um *request*, as vezes, sai mais caro do que os próprios dados nele contidos. Por essa razão, economizar *requests* seja, talvez, algo válido de ser feito.

Podemos, portanto, analisar os *javascript* que temos e observar qual é curto o suficiente e que valha a pena realizar esse processo. Vamos pegar um exemplo, o arquivo "home.js"

que possui três linhas. Vamos tentar colocá-lo junto ao "index.html"! Basta, para tanto, copiar e colar seu código no "html", entre as *tags* de *script*. Observemos:

A screenshot of a code editor with a dark theme. The editor shows two files: 'index.html' and 'home.js'. The 'index.html' file is open, and a red rectangular box highlights a section of code between lines 10 and 15. This section contains a `<script>` tag followed by a JavaScript code block that checks the user agent for 'Safari' and 'Edge' and adds a class 'home-fundo-animado' to the document element. Below the highlighted section, there are comments and other `<script>` tags for 'menu.js', 'busca.js', 'detect.js', and 'footer.js'. The 'home.js' file is also visible in the background.

```
7
8 <title>Alura | Cursos online de tecnologia</title>
9
10 <script>
11 if (navigator.userAgent.indexOf('Safari') > 1 && navigator.userAgent
12   .indexOf('Edge') < 0) {
13   document.documentElement.className += ' home-fundo-animado';
14 }
15 </script>
16
17 <!-- build:js assets/js/scripts.js -->
18 <script src="assets/js/menu.js"></script>
19 <script src="assets/js/busca.js"></script>
20 <script src="assets/js/detect.js"></script>
21 <script src="assets/js/footer.js"></script>
22 <!-- endbuild -->
23
```

Acabamos de economizar esse "home.js", que passa a vir junto com o *html* . Não só diminuímos o *request* como também aceleramos a execução do *script*. Podemos utilizar esse meio tanto para uma economia de *requests* como também para priorizar certas funcionalidades.

E você pode estar se perguntando: teremos que escrever todos os *javascript* a mão? Podemos fazer isso de uma maneira mais automática?

Podemos! Imagina que queremos manter o código *javascript* em um arquivo separado só que em produção gostaríamos que ele estivesse embutido na página. Para fazer utilizaremos algumas ferramentas, por exemplo, usando o *gulp file*. Esse arquivo faz diversas coisas e um dos pluggins que podemos utilizar se chama *inline source* que é capaz de colocar um arquivo externo embutido *in line*. A primeira coisa que precisamos fazer é tirar o "home.js" do *bundle*. Queremos que ele seja independente. Ao em vez de colocar ele como um request a parte vamos escrever ele *inline*:

O atributo do *gulp* ao ler a tag *inline* pegará o conteúdo e irá embuti-lo dentro do *html*. Vamos ver se isso funciona na prática? Vamos escrever no terminal `gulp useref` e dar um "Enter". Após rodar isso vamos observar como ficou em nosso navegador. Se formos observar o *page source* conseguiremos observar o *script* embutido.

```
view-source:localhost:3030
1 <!DOCTYPE html><html lang="pt-BR"><head><meta charset="UTF-8"><meta name="viewport" content="width=device-width,initial-scale=1">
  <title>Alura | Cursos online de tecnologia</title>
  <script>navigator.userAgent.indexOf("Safari")>1&&navigator.userAgent.indexOf("Edge")<0&&(document.documentElement.className+=" home-
  fundo-animado");</script><script src="assets/js/scripts.js"></script><link rel="stylesheet" href="assets/css/estilos.css"></head>
  <body><div class="nome-rumo"><header class="header"><section class="header-barraBusca"><div class="container"><form action="/busca
  class="header-barraBusca-form"><label for="header-barraBusca-campoBusca">Qual curso procura?</label><input type="search" id="header-
  barraBusca-campoBusca" name="q" placeholder="Digite aqui a busca"> <button class="header-barraBusca-submit"
  type="submit">Buscar</button></form></div></section><div class="container"><div class="header-logo"></div><div class="header-navegacao"><form action="/busca" class="header-navegacao-form"><input
  class="navegacao-form-input" type="search" placeholder="Digite sua busca aqui" name="q"><!--
  --><button class="navegacao-form-submit" type="submit"></button></form><a href="/busca" class="header-busca"> </a><a href="https://cursos.alura.com.br/dashboard" class="header-areaAluno">Login </a><a href="#planos"
  class="header-cta">Matricule-se</a></div></div></div><div class="container home-headline"><h1 class="titulo-destaque">Coloque seu
  potencial em prática!</h1><h2 class="subtitulo-destaque">Cursos online de tecnologia que reinventam sua carreira.</h2><a
  href="#planos" class="buttonLight home-headline-cta">Matricule-se já</a></div><div class="categoriaCard-lista container"><!--
  --><a href="/cursos-online-mobile" class="categoriaCard-item bg-categoria-mobile"><svg class="categoriaCard-item-
  icone"><use xlink:href="assets/img/categorias.svg#mobile"/></svg><h3 class="categoriaCard-item-nome"><div class="categoriaCard-item-nome-cursos">Cursos de</div>Mobile</h3><p class="categoriaCard-item-descricao">iOS, Android, PhoneGap, e mais...</p></a><!--
  --><!--
  --><a href="/cursos-online-programacao" class="categoriaCard-item bg-categoria-programacao"><svg
  class="categoriaCard-item-icone"><use xlink:href="assets/img/categorias.svg#programacao"/></svg><h3 class="categoriaCard-item-nome">
  <div class="categoriaCard-item-nome-cursos">Cursos de</div>Programação</h3><p class="categoriaCard-item-descricao">Java, C#, Ruby,
  PHP, Python e mais...</p></a><!--
  --><!--
  --><a href="/cursos-online-front-end" class="categoriaCard-item bg-categoria-front-end"><svg class="categoriaCard-
  item-icone"><use xlink:href="assets/img/categorias.svg#front-end"/></svg><h3 class="categoriaCard-item-nome"><div
  class="categoriaCard-item-nome-cursos">Cursos de</div>Front-end</h3><p class="categoriaCard-item-descricao">HTML, CSS, Angular,
  JavaScript e mais...</p></a><!--
  --><!--
```

A *tag* que tínhamos escrito virá parte do *html* e, inclusive, já está minificada.

Podemos utilizar esse recurso para priorizar os *requests*. Continuamos no nosso navegador e vamos na aba "Network > Img" e vamos reparar no logo **Alura**. Ele está em um *request* separado e acaba demorando um pouco para ser baixado:



Imagine que gostaríamos de alterar essa priorização, que esse "logo-alura.svg" fosse acelerado.

Como fazemos isso?

É muito fácil o *svg* é um *xml*, poderíamos abrir o arquivo "logo alura.svg" no *Sublime* e podemos copiar o seu código direto onde estamos usando a imagem do logo, no "index.html" e colamos em baixo de

```

```

Mas será mesmo que copiar e colar esse código é necessário? Podemos utilizar, também, o *plugin*, para isso basta escrever inline:

```
<img inline src=""assets/img/logo-alura.svg" alt="Alura">
```

E executar no terminal o `gulp userref`. Vamos observar como ficou no navegador? Podemos observar que nas requisições não temos mais nada relacionado ao logo. Pois, se clicamos no *html* podemos observar que teremos nele o *svg* do logo embutido:

Além de economizarmos os *requests* estamos acelerando a chegada da imagem.

Mas isso não é ruim? O *html* não acabará ficando muito grande com o que acabamos de embutir nele? Podemos fazer um teste, vamos retirar a imagem que acabamos de colocar nele e vermos a diferença em termos de *kbytes*. Com a imagem temos 7 KB e sem ela temos 6.2 KB.

É ruim, então, termos diversas coisas no *html*?

Na verdade, o que pode ser ruim é escrever tudo isso a mão, o que pode atrapalhar a manutenção. Por exemplo, se quisermos editar a imagem no *Illustrator* isso pode ser complicado. Mas, temos ferramentas que podem nos auxiliar na ida dessa questão.

Talvez a pergunta seja: é ruim do ponto da performance? Ou, qual o limite para fazer esse tipo de *inline*?

Para recursos pequenos é bastante válido e pequeno consideraremos cerca de 1, 2 ou 3 KB. Nessa caso o custo de fazer um novo *request* é muito maior.

Uma coisa que temos que pensar é que quando embutimos algo no *html* isso significa que todas as páginas terão que baixar esse recurso, assim, acabamos ao mesmo tempo com o *cache*. Isso pode ser ruim no caso de você ter muitos dados, entretanto, nesse cenário nos parece "ok" fazer isso. Não é problema baixar, por exemplo 5 KB a mais se isso acelerar a renderização da página.

E você pode, ainda, estar se perguntando: Qual é um tamanho "razoável" para um *html*?

Existem várias métricas para medirmos isso. O tamanho ideal para o *html* é que ele seja menor do que 14 KB, isto é um *html* gzipado e com tudo que deve ser embutido.

E de onde saiu esse número?

Isso está relacionado ao *tcp*, o protocolo de rede básico onde transferimos pacotes entre servidor e cliente. É interessante entendermos um pacote *tcp*. Vamos dar um *google* em *tcp segment size*.

Typical MTU size in TCP for a home computer Internet connection is either 576 or 1500 bytes. Headers are 40 bytes long; the MSS is equal to the difference, either 536 or 1460 bytes. In some instances the MTU size is less than 576 bytes, and the data segments must therefore be smaller than 536 bytes.

What is maximum segment size (MSS)? - Definition from ...
searchnetworking.techtarget.com/definition/maximum-segment-size

Ele nos informa que na maior parte dos cenários, um segmento *tcp*, a quantidade de dados que é passado do servidor para o cliente e vice-versa é de 1460 bytes, que equivale a mais ou menos 1.4 K.

A grande pergunta é: quantos segmentos *tcps* chegam logo de cara para nós?

Novamente, vamos dar um *Google* em *tcp initial size*. E ele nos informa que o tamanho inicial é cerca de quatro vezes o tamanho do segmento que vimos. Isso quer dizer que o tamanho inicial da resposta *tcp* que vêm do servidor é de aproximadamente 5.8 KB. Mas não eram 14 KB?

Abrindo o link <https://tools.ietf.org/html/rfc3390> podemos ver uma outra proposta, de que a janela inicial seja aumentada. Nesse texto é afirmado que devem ser 10 segmentos ao

em vez dos 4 clássicos e com isso teremos um cálculo aproximado de 14 kbytes ($1460 \times 10 = 14.600$).

O que estamos tentando dizer é que podemos transferir 14 kbytes se estivermos utilizando a rfc 3390.

E as pessoas utilizam isso?

Grande parte dos servidores modernos utilizam a janela inicial do tcp dessa maneira. Na prática, se você tiver um *html* de 10, 12 ou 14 Kbytes a diferença é quase negligenciável.

Pois, como funciona um *tcp*? O *browser* manda a requisição para o servidor que responde com uma leva de pacotes iniciais que chegam mais ou menos juntos. Se precisássemos de mais de 14 kbytes ele precisaria pedir mais 14 kbytes que irão demorar, mas eles virão juntos e se forem só 10 eles virão juntos e assim por diante. Assim, não faz tanta diferença prática, a não ser que tenhamos um *html* tenha muito mais do que 14 Kbytes. Por isso tentamos deixar abaixo de 14 Kbytes.

Vamos voltar, rapidamente, ao que tínhamos na prática. Ter de 6 KB para 7 KB na prática não faz muito diferença pois eles chegaram todos juntos.

Usar esse recurso de embutir coisas no *html* é interessante para diminuir a quantidade de requisições e desde que o *html* não estore os 14 kbytes o custo não é tão grande assim.

A dica é usar esse recurso com parcimônia e analisar bem o impacto. Pois, as vezes o impacto também é negativo.