15

Transcrição das aulas

A performance é bastante importante, pois, todos queremos um site rápido. Um site mais rápido deixa os usuários mais satisfeitos e usuários mais felizes compram mais da marca, acabam voltando mais ao site e falam melhor dele para as pessoas. Portanto, o que buscamos é um site mais rápido.

Nesse curso, trataremos sobre otimização de performance nas páginas "html" de nossos sites.

Duas das primeiras perguntas que as pessoas normalmente fazem a respeito desse curso são:

Que tipo de otimizações?

Por que este curso está na categoria de *front-end*?

Porque, normalmente, em um site temos duas frentes envolvidas. A primeira delas é o *back-end* que é onde desenvolvemos as coisas no servidor, é onde escrevemos o código *java*, *php* e etc. E a segunda dela que é o *Front-End*, que é onde fica o *html*, *css*, *java script* e etc.

Nesse curso, focaremos em performance de *Front-End*. O que isso quer dizer? Não falaremos de performance de banco de dados, não falaremos de como escrever o melhor código *php* ou coisas do tipo.

Falaremos de como melhorar a performance do ponto de vista do *html*, *css*, *java script* e etc. Isso é muito bacana pois de aplica a todo o tipo de cenário, aplicativo *web*, de site, não interessa se alguém utiliza *php*, *java* etc. A pessoa utilizará *java script*, *html*, *ccs*.

Esse curso não só apela para todos que trabalham com *web* ou *back-end*, pois, vários estudos mostram que a performance final conta muito para o usuário.

É a performance que conta para o usuário, quando ele abre algo no navegador, no celular, no 3G. A performance que o usuário sente é, geralmente, mais atribuída ao *front-end* do que do *back-end*.

O que queremos dizer com isso? A não ser que você esteja fazendo algo muito grave no *back-end*, uma busca no banco de dados e que esta seja extremamente demorada, geralmente, na prática, o maior gargalo de performance dos site reais acaba sendo em técnicas de *front-end*.

Como assim?

Bom, coisas como: como fazemos para carregar o *css*, como fazemos para carregar o *java script*, como criamos as imagens e disponibilizamos elas?

Tudo isso acaba afetando, a prática, o mundo real e muito as páginas *web*, independente do *back-end* que estamos utilizando.

Então, por isso, é interessante que discutamos esse tema. Esse é o foco do curso número 1 de performance e também do curso número 2 que continua depois dele.

Então, o que faremos nesse curso?

O nosso projeto é desenvolver um site, no caso, o site do *Alura*. Ele já está pronto e temos a seguinte versão atual:

O site vai ser mudado e talvez a versão que tenhamos hoje já esteja distinta a que aparece na imagem.

Os arquivos iniciais serão fornecidos e a ideia é que esse site do *Alura* que já foi desenvolvido e já está codificado, mas que não é uma versão otimizada, seja melhorado.

A ideia é que localizemos quais são os gargalos de performance que a página possui, analisar esses gargalos, perceber seus impactos e corrigir isso, melhorando essa página. Nosso site ficará mais rápido do que é atualmente. Interativamente, visualmente, ele permanecerá o mesmo, mas ele estará muito mais rápido do que antes.

Ao longo do curso veremos diversas técnicas diferentes que são capazes de melhorar a performance.

Uma coisa que mapeia diretamente a performance do site é o tamanho dele, isto é, dos arquivos que teremos que baixar. Uma primeira coisa que podemos fazer para otimizar o nosso site é diminuir o tamanho dos arquivos do nosso site.

Tem um jeito óbvio de fazer isso que é cortar funcionalidades, isto é, assim faremos menos coisas e, consequentemente teremos um código menor. Mas, a ideia é que continuemos com o nosso site completo e veremos aqui como otimizá-lo em relação ao tamanho.

Precisamos pensar que tipo de dados estamos enviando para o navegador e que tipo de dados são desnecessários para serem enviados. Se olharmos nosso código fonte e observarmos, por exemplo, o *css*, ele é normal e igual ao que estamos acostumados a escrever, inclusive, temos diversos comentários nesse *css*:

Pense o seguinte, é interessante para o navegador renderizar esses comentários? Não. Eles existem para o desenvolvedor.

Para a execução da página esses comentários são desnecessários, assim como os espaços, os "Enter", que organizam o código. Eles não precisam constar, aliás, podemos juntar várias coisas na mesma linha que continuará a ser um *css* válido. Imagine quantos espaços, "Tab", "Enter" e comentários que existem em nosso código e que são desnecessários no momento da execução. Podemos remover tudo isso e diminuir o tamanho de nossos arquivos.

Por que deixamos o código cheio de espaços? Porque nós humanos precisamos entender o que está escrito nele.

Removemos isso "na mão"? Existem ferramentas que fazem isso por nós, chamamos esse procedimento de minificar.

Vamos dar um *google* em *minify css js*. Entraremos no site [refresh-sf.com (http://refresh-sf.com)](http://refresh-sf.com) e mostraremos como funciona uma ferramenta de minificação.

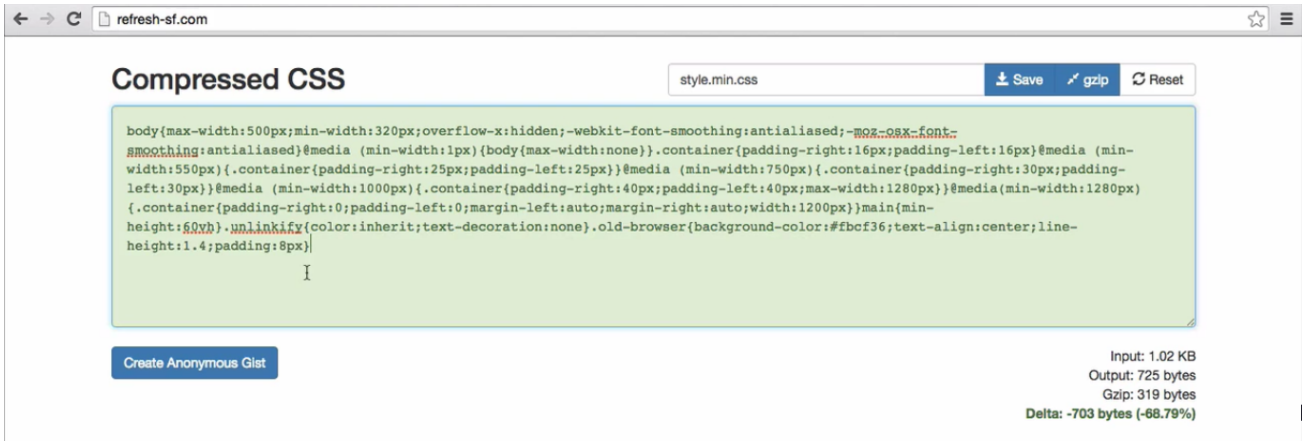
Selecionaremos todo o código do nosso *css*, copiaremos e colaremos no campo em branco desse site. Clicaremos o botão *css* e ele devolve o código minificado, como podemos observar na imagem acima. Ele informa quantos kbytes tínhamos e quanto temos agora. Observe no canto direito abaixo do campo, tínhamos cerca de "1002 KB" e agora temos "725 bytes", ou seja, economizamos cerca de 30% do tamanho desse arquivo.

Podemos fazer esse mesmo procedimento com o *java script*. A única coisa que faremos de diferente é apertar o botão *Javascript* ao em vez do botao *css*.

Vamos observar agora os programas que já baixamos.

Você já reparou nas ferramentas que você baixou?

Por exemplo, o *jquery*, ele possui um tamanho relativamente grande, "229KB". Você já reparou que no momento em que baixamos essas ferramentas elas possuem algumas opções, por exemplo, se formos no site do *jquery*, o jquery.com/download/ (http://jquery.com/download/) ele nos pergunta se desejamos baixar a versão comprimida ou descomprimida. Isso significa que a a versão descomprimida é o código gigante, cheio de comentários e espaços. A versão comprimida é um código com este estilo:



É um código em uma linha só que nós não compreenderemos, mas o navegador entenderá o código.

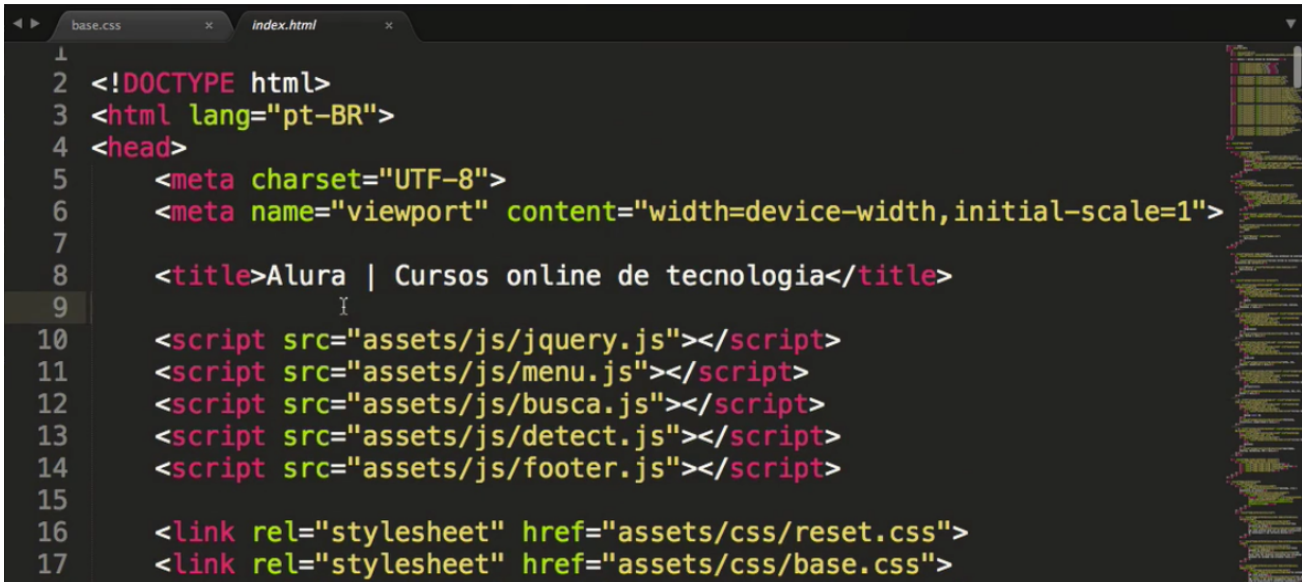
Você pode estar se perguntando se isso realmente traz alguma diferença, e a resposta é sim! Tínhamos um *jquery* de "259KB" e, agora, temos uma de versão comprimida de "86 KB". Repare:

Assim, apagamos as versões que não estavam minificadas e, portanto, são maiores e conservamos as que acabamos de baixar que estão minificadas e, assim, menores.

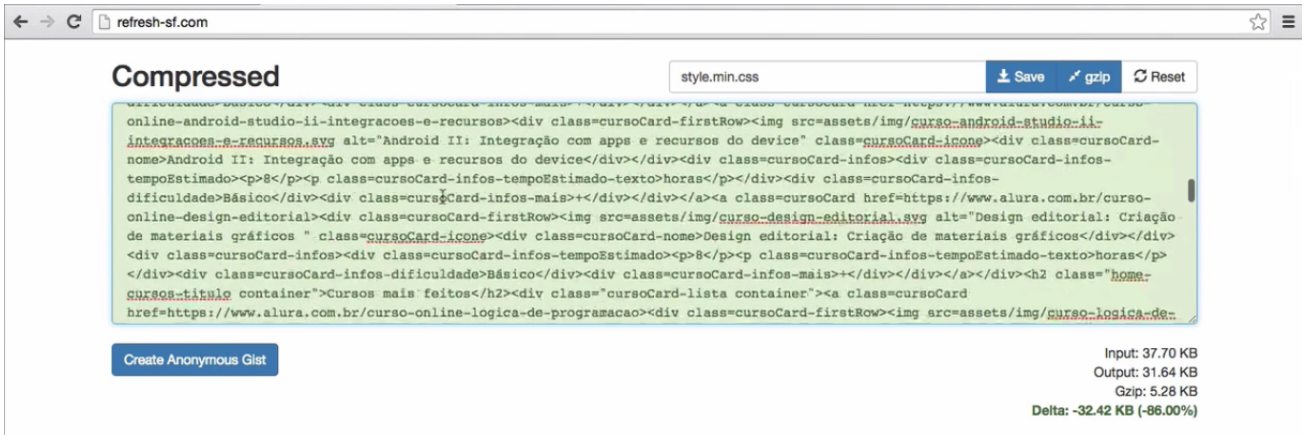
Bom, e o que faremos agora?

Podemos baixar os *frameworks* minificados e minificaremos os demais arquivos utilizando a ferramenta do *site*. Repare que o site ainda traz a opção de minificar em *HTML*.

Se abrirmos o *html* temos a mesma coisa, espaços e comentários que também poderiam ser removidos.



Então, vamos querer fazer uma minificação de *html*. Vamos fazer um teste! Nesse site temos que clicar na opção que está abaixo do campo, a *html Minifier* e configurar que tipo de remoções gostaríamos no *html*. Vamos selecionar as opções de remover espaços, comentários, aspas dos atributos quando desnecessárias e rodar com essas opções:



Também funciona! E economizamos de *37KB* para *31KB, 6KB*.

Mas, a ideia é que não precisemos ficar abrindo o site e sim que automatizemos isso. Podemos instalar uma ferramenta em nossa máquina, por exemplo, temos o *Uglify JS 2*, que é uma ferramenta do *Node*, e podemos instalá-la em nossa máquina. Instalaremos isso digitando o seguinte no terminal:

```
> npm install uglifyjs -g
```

O `-g` é para indicar que é global. Após digitar isso ele baixará o *Uglify* e teremos a permissão de usá-lo na linha de comando. Então, podemos passar um nome de algum arquivo para ele e é feito o processo de minificação. Por exemplo, vamos passar o arquivo `site/assets/js/busca.js` . Ao digitar isso ele nos passa o arquivo já minificado.

```
performance-web $ uglifyjs site/assets/js/busca.js
(node) util.print is deprecated. Use console.log instead.
window.addEventListener("load",function(){var botaoBusca=document.querySelector
(".header-busca");var body=document.body;var navegacaoForm=document.querySelect
or(".header-navegacao-form");if(botaoBusca&&"classList"in document.documentElem
ent){function toggleSearch(event){body.classList.toggle("header-barraBusca-visi
vel");navegacaoForm.classList.toggle("navegacao-form-active");event.preventDefa
ult()}function closeSearch(){body.classList.remove("header-barraBusca-visivel")
;navegacaoForm.classList.remove("navegacao-form-active")}function cancelPropaga
tion(event){event.stopPropagation()}function setupCloseSearch(){setTimeout(func
tion(){document.documentElement.addEventListener("click",closeSearch);document.
```

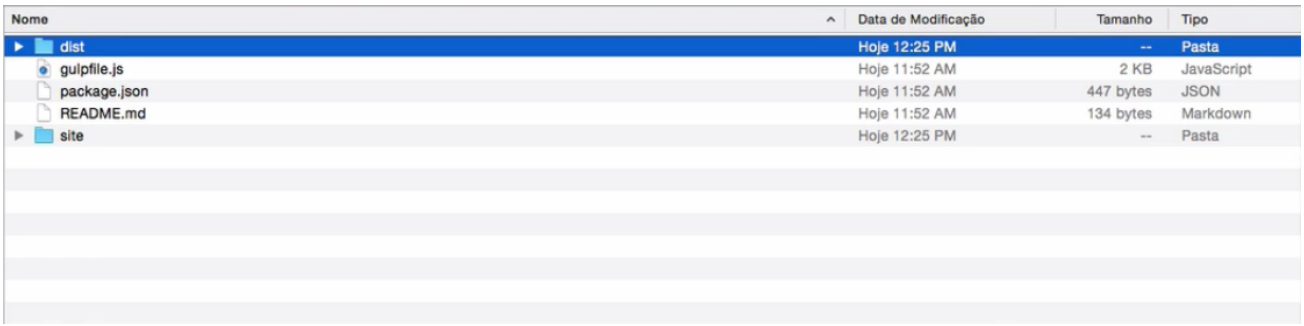
Ele cospe para nós, no terminal, o código sem espaços, sem comentários e etc. A ideia é que possamos rodar isso para vários arquivos.

E como fazemos para guardar isso?

Não colaremos esse código no nosso arquivo fonte, pois o código fonte é útil para que nós. Os desenvolvedores utilizam o código fonte para compreender o que está acontecendo e é preciso entender o que está escrito nele.

O que fazemos então?

Queremos minificar para otimizar o código, mas, não queremos estragar nosso código fonte. Na prática o que fazemos é não minificar o código fonte, criamos uma outra pasta, isto é, duplicamos o conteúdo da pasta do código fonte e nessa cópia é que realizamos a minificação. Por exemplo, temos uma pasta "site", copiaremos ela e renomearemos para "dist", nome muito comum e que se aplica àquilo que será distribuído.



Nessa pasta "dist" nós colocaremos o código minificado, sem estragar o arquivo fonte. Podemos fazer isso apenas copiando e colando o código da Internet. Podemos fazer isso de outra maneira, usando no Terminal algo que jogará o conteúdo para a pasta "dist".

Digitaremos no terminal o seguinte:

```
uglifyjs site/assets/js/busca.js -o dist/assets/js/busca.js
```

Fazemos isso, digitando o `-o dist/assets/js/busca.js`, o `-o` significa `output`, ou saída. O que estamos falando é para o *uglify* pegar o arquivo `site/assets/js/busca.js`, minificar ele e jogar no `dist/assets/js/busca.js`. Teremos:

Vemos que ele está minificado em uma única linha. Deu certo!

Precisaremos rodar isso para cada arquivo?

Não, a ideia é que tentemos automatizar isso, mas não focaremos nesse tema de automatização nesse capítulo, pois, temos cursos específicos que abordam esse assunto. Assim, já temos pronto um `gulpfile.js` . Apenas para compreender um pouco a cerca do `gulp` teremos o seguinte:

```
gulp.task('minify-js', function() {
    return gulp.scr('site/**/*.js')
        .pipe($.ugly())
        .pipe(gulp.dest('dist/))
});
```

Compreendendo: Temos uma `task` , tarefa, que buscará minificar todos os `js` , os *javascript*, do site. Rodará o `uglify` e salvaremos o que foi minificado na pasta `dist` .

A mesma coisa é feita com o *css* e o *html*.

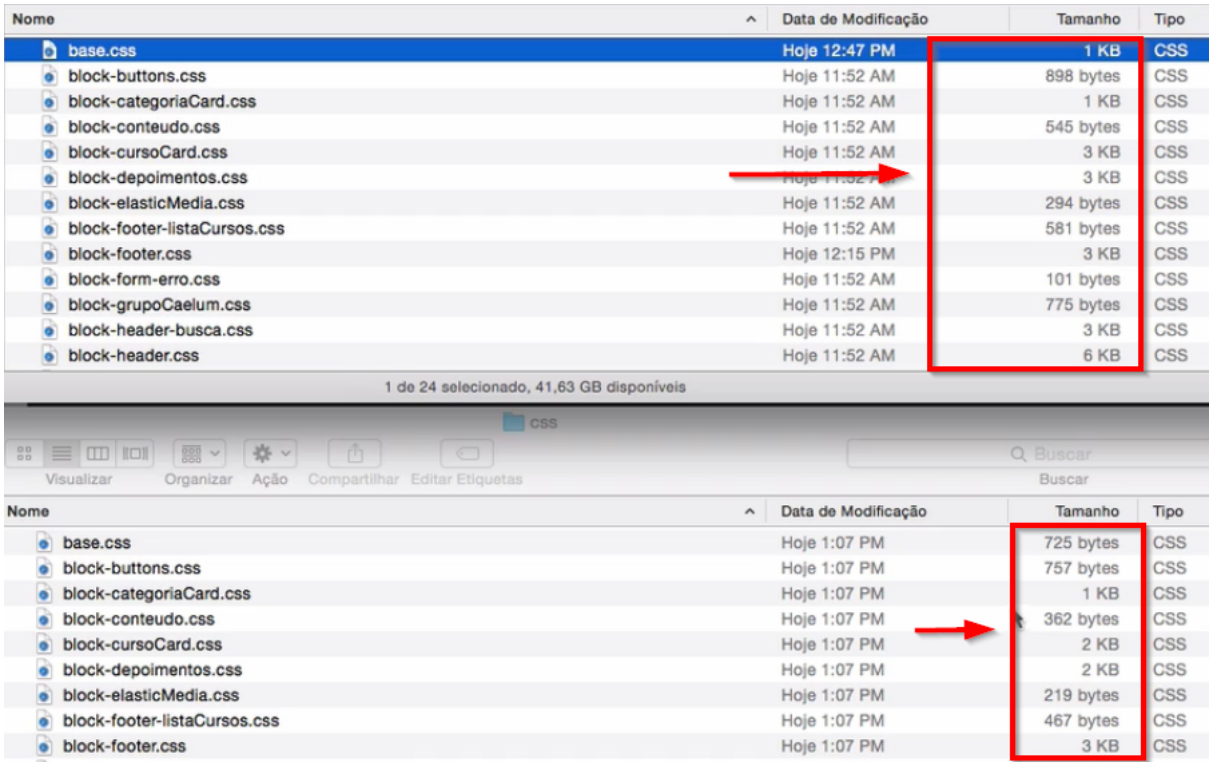
Não precisamos nos preocupar muito com esse código, pois ele não é o foco desse curso.

A primeira coisa que temos que fazer é instalar o *gulp* no nosso projeto e todos os *pluggings* que ele necessita. Para isso rodaremos na pasta do projeto os comandos `npm install` e o `npm install gulp-cli -g` para ter ele globalmente na máquina. Após a instalação podemos rodar o `gulp` na linha de comando e com as tarefas que já tinhamos preparado, por exemplo, veremos como funciona o `gulp` para minificar todos os *java script* da página. Para isso digitaremos `gulp minify-js` .

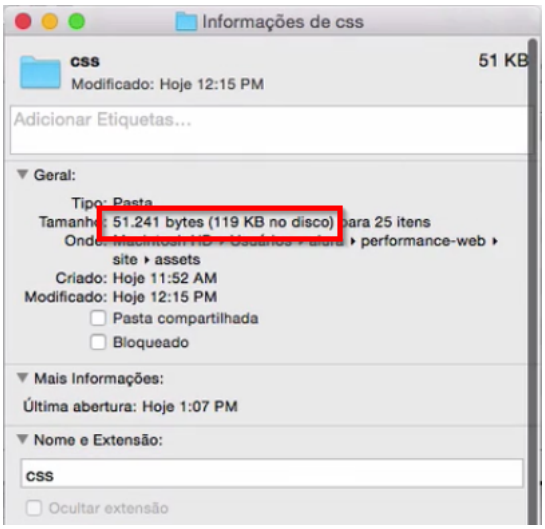
Vamos conferir se ele rodou? Se formos na pasta `dist` veremos que todos os arquivos `js` da pasta `dist` estarão minificados.

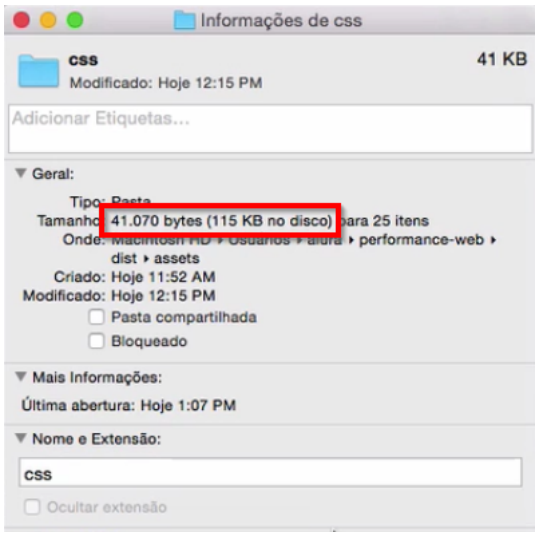
E podemos fazer o mesmo com o `css` digitando `gulp minify-css` e daremos um "Enter". E se formos na pasta "css" do "dist" teremos tudo minificado:

Podemos conferir o antes e o depois em relação ao tamanho dos arquivos.



Repare na diferença dos tamanhos. Um arquivo que antes tinha "1 KB" agora possui "725 KB" e assim por diante. Podemos comparar as pastas inteiras, primeiramente a pasta tinha "51 KB" e agora possui "41 KB":





Minificamos o *html* a mão, mas também poderíamos ter usado o `gulp` . Bastaria digitar `gulp minify-html` e ele também faria a minificação.

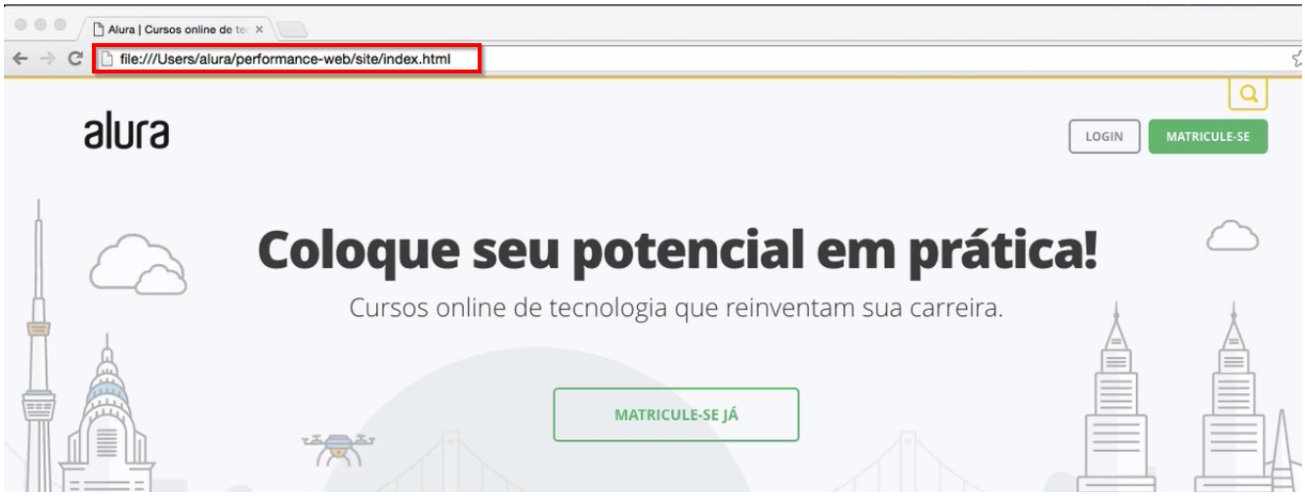
E se quisermos rodar as três coisas? Se digitarmos `gulp minify` no terminal ele vai minificar as três coisas e vai escrever tudo na pasta "dist".

Nós utilizamos o `gulp` pois é uma das ferramentas mais famosas de `front-end` , mas existem diversas ferramentas que são úteis para minificação. Por exemplo, você pode trabalhar com *php* e querer utilizar outra coisa que não seja o *gulp*. Basta jogar no *google* e encontrar algum projeto de alguém que tenha feito um minificador para *php*.

A ideia é que minificar é bacana e relativamente fácil, pois as ferramentas são configuradas uma única vez e pode ser feito com *php*, *node*, *java*, *gulp* e etc. Com as ferramentas que melhor se adequam para você. Além disso, fazer a minificação nos dá ganhos de cerca de 20 a 30% em termos de tamanho.

A otimização que fizemos usando a minificação foi algo que mexeu diretamente com os arquivos do projeto, entretanto, várias das otimizações que veremos ao longo do curso realizam alterações na parte do servidor.

Até esse momento não estamos utilizando um servidos *http*, estamos acessando através de "file:". Observe:



A primeira coisa que faremos antes de seguir adiante com a otimização é rodar um *http*.

Podemos escolher entre os vários servidores que existem no mercado, aqui, utilizaremos um que está bastante famoso no mercado, o *nginx*, além disso, esse servidor é fácil de ser utilizado.

Podemos configurar esse servidor facilmente e usamos, para tanto, arquivos de configuração. Primeiro, criaremos um arquivo de configuração que salvaremos dentro da pasta do *nginx*. No nosso caso a pasta está em "/usr/local/etc/nginx/servers", mas essa localização pode mudar em outros computadores.

Dentro dessa pasta criaremos um arquivo novo que denominaremos "curso alura" e nele estarão contidas as configurações *http* para que o possa se servir desse arquivo *nginx* do *Alura*.

É bastante simples, vamos criar um *directive server*, colocaremos a porta que quisermos que seja escutada, no caso, escolheremos a `3030`, mas pode ser qualquer uma. Como nosso site é estático, usando a diretriz `root` diremos onde estão os arquivos do site, no caso, na pasta `/Users/alura/performance-web/site/`. Só isso é o suficiente para criar o servidor *nginx*:

```
server {
    listen 3030;
    root /Users/alura/performance-web/site/
}
```

"Startamos" o *nginx* na linha de comando e apenas digitamos no terminal `nginx`.

E agora, no navegador, podemos acessar digitando: "localhost:3030". Agora, temos o *nginx* rodando em local host com *http*.

Por que fizemos isso?

Porque rodando com *http* conseguimos fazer alterações do lado do servidor. Uma delas é habilitar a compressão das coisas.

O que isso significa isso?

Toda vez que acessamos um servidor remoto baixamos o *html*, o *css*, o *javascript* e etc e que são arquivos de texto, minificados ou não. Um arquivo pode ter o tamanho de "50 KB" e quando o transferimos pela rede ele é transmitido do jeito que estiver, entretanto, podemos comprimir esses "50 KB" para baixá-lo mais rapidamente.

É a mesma coisa que fazemos quando queremos enviar um arquivo para algum amigo e zipamos esse arquivo pois ele está muito grande. Assim, enviamos um arquivo zipado que está comprimido, mas quando ele é aberto, do outro lado, ele está do tamanho original.

Conseguimos fazer esse mesmo mecanismo na *web*, podemos "zippar" e "deszippar" conteúdos através do navegador e do servidor. O servidor "zipa" as coisas, manda para o navegador que "deszipa" elas. Mas, por padrão não fazemos isso.

Para fazermos isso vamos usar o *gzip*. Vamos habilitar isso através de uma diretiva muito simples, o comando *gzip on*. O *gzip* serve para que o servidor "zipe" o conteúdo e mande para o "cliente" que "deszipa" e exibe no navegador. E tudo isso ocorre de maneira bastante transparente. Teremos o seguinte:

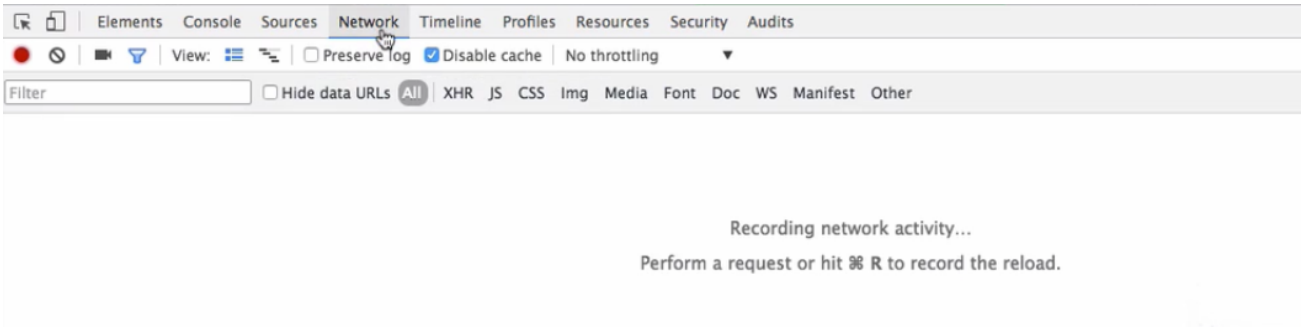
```
server {
    listen 3030;
    root /Users/alura/performance-web/site/

    gzip on;
}
```

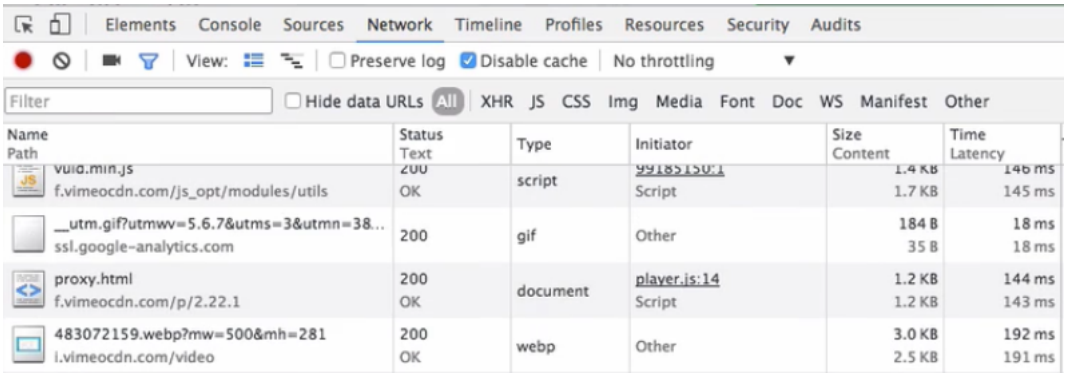
Agora, precisamos recarregar o *nginx* no terminal, para isso digitaremos `ngix -s reload` e acessamos novamente o "localhost:3030".

Como sabemos se funcionou?

Começaremos a aprender como analisar as métricas de performance no site. E iniciaremos com o *dev tools*, para isso usamos o "F12" que abrirá as ferramentas de desenvolvimento. Estamos utilizando o *google chrom* e temos as diversas ferramentas que ele mostra, a que mais nos interessa nesse instante é a aba *network*.

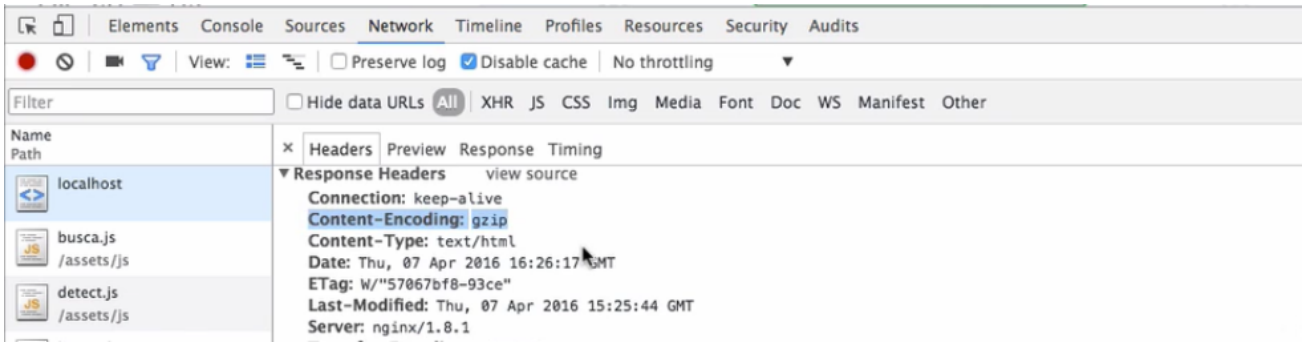


Nessa aba, se atualizamos o site, ele mostra tudo o que foi baixado no site, o *css*, o *html*, imagens e diversas outras coisas, além de sinalizar, também, o tamanho disso tudo.



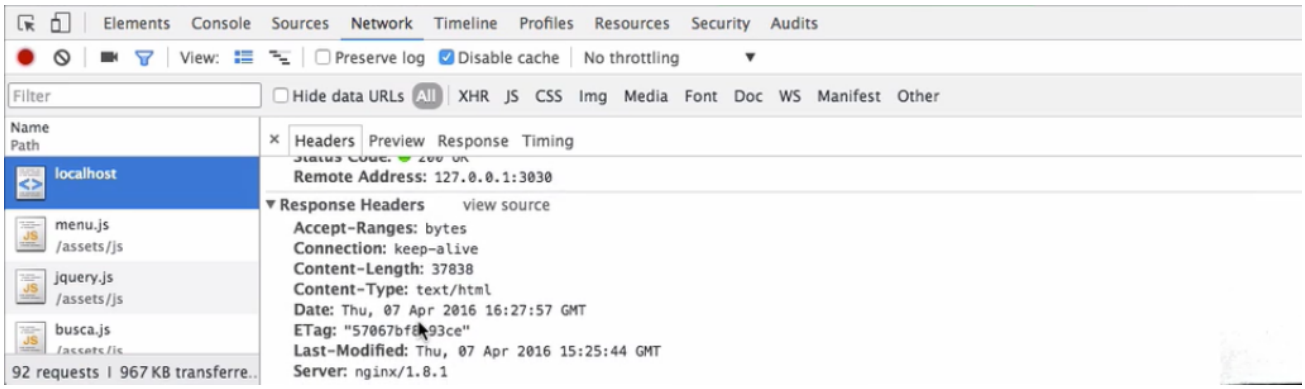
Repare que o *localhost* é o "html" original e se clicarmos nele podemos perceber que ele foi baixado e que possui duas versões, em preto está escrito a versão que foi baixada na rede e em cinza mostra o tamanho do conteúdo em si.

Se olharmos o arquivo *html* veremos que a pasta possui "38KB", mas, se olharmos no `dev tools` veremos que ele baixou apenas "6.9 KB", mas o ponto é que isso foi baixado por que ele foi *gzipado*. Para confirmar isso podemos olhar nos *headers*:



E se apagassemo o `gzip on` e no terminal e pedíssemos para recomeçar? Vamos apagar o `gzip on` e no terminal, para recomeçar, digitaremos `ngix -s reload`.

Vamos dar um *reload* na página para observar a diferença:



Temos "37 KB" e "37 KB" de tamanho e no *header* não tem mais *gzip*. Habilitar o *gzip* no servidor é tão simples, quanto colocar uma única linha de configuração e já ganhamos uma compreensão absurda. Pois, lembra-se, já tínhamos "37 KB" e tinham ficado "6.9 KB".

Por padrão o servidor *ngix* só comprime o *html*. Por exemplo, o *javascript* continua com "83 KB", ele não é comprimido. Mas, é muito fácil fazer ele comprimir outras coisas, nós digitamos abaixo de `gzip on`, `gzip_types`.

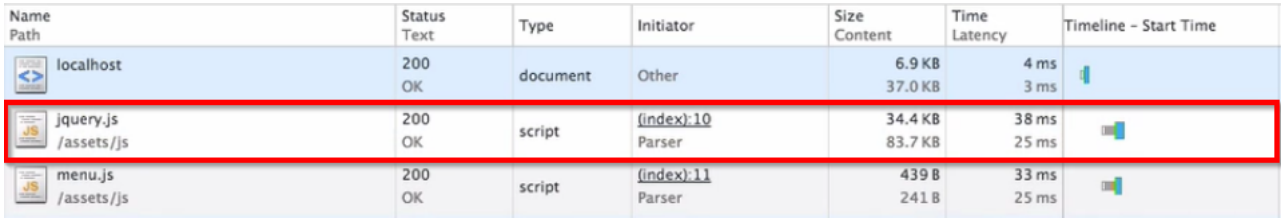
O `gzip_types` recebe uma lista de ouros conteúdos que queremos que sejam "gzipados", por exemplo, diremos para ele que queremos que todos os *css*, *javascript*, imagens *svj* sejam "gzipados". E podemos colocar ainda diversas outras coisas. Digitaremos o seguinte:

```
gzip on;
gzip_types text/css application/javascript image/svg+xml;
```

Algo comum a todos esses tipos é que são arquivos com formato texto. O *gzip* lida muito bem com arquivos em formato texto, ele é muito bom para isso. Portanto, se pegamos um *jpeg*, por exemplo, não há tanta diferença, por isso nem tentaremos comprimir esses arquivos aqui, além disso, é um arquivo muito pesado e que pode deixar, inclusive, o servidor mais sobrecarregado.

Vamos dar um *reload* e observar o que aconteceu? Digitaremos no terminal, de novo, o `nginx -s reload` .

Vamos observar o *jquery*, por exemplo, ele tem cerca de "83 KB", mas *gzipado* são cerca de "34 KB".



Name / Path	Status / Text	Type	Initiator	Size / Content	Time / Latency	Timeline – Start Time
localhost	200 OK	document	Other	6.9 KB 37.0 KB	4 ms 3 ms	
jquery.js /assets/js	200 OK	script	(index):10 Parser	34.4 KB 83.7 KB	38 ms 25 ms	
menu.js /assets/js	200 OK	script	(index):11 Parser	439 B 241 B	33 ms 25 ms	

Podemos observar outro exemplo, o *css* que tem "1.1 KB" e *gzipado* possui "563KB":

Name / Path	Status / Text	Type	Initiator	Size / Content	Time / Latency	Timeline – Start Time
/assets/js	OK	script	Parser	249 B	24 ms	
reset.css /assets/css	200 OK	stylesheet	(index):16 Parser	515 B 371 B	34 ms 24 ms	
colors.css /assets/css	200 OK	stylesheet	(index):18 Parser	563 B 1.1 KB	58 ms 42 ms	

As vezes ele aumenta um pouco pois são considerados, também, o tamanho dos *headers*. Podemos ignorar isso, em arquivos pequenos a diferença não é muito grande, mas em arquivos maiores ela é mais significante.

Podemos observar qual o tamanho final para transportar toda a página que é "772 KB". Se desabilitarmos o *gzip*, apagando `gzip on` podemos observar a diferença. A página inteira ficará com "967 KB".

Assim, foram economizadas centenas de *Kbytes*, simplesmente, habilitando o *gzip*.

E se você não utilizar o `ngix` ?

Cada servidor é de um jeito e cada um precisa de uma configuração, mas geralmente esta configuração é bem simples. Se você utiliza algum *hosting* externo ou contratado é bastante grande a chance de que ele já esteja com *gzip* habilitado, é quase que padrão. Se você utiliza algum servidor diferente, consulte a documentação. Por exemplo, no *Apache* não é a mesma configuração, mas é tão simples quanto.