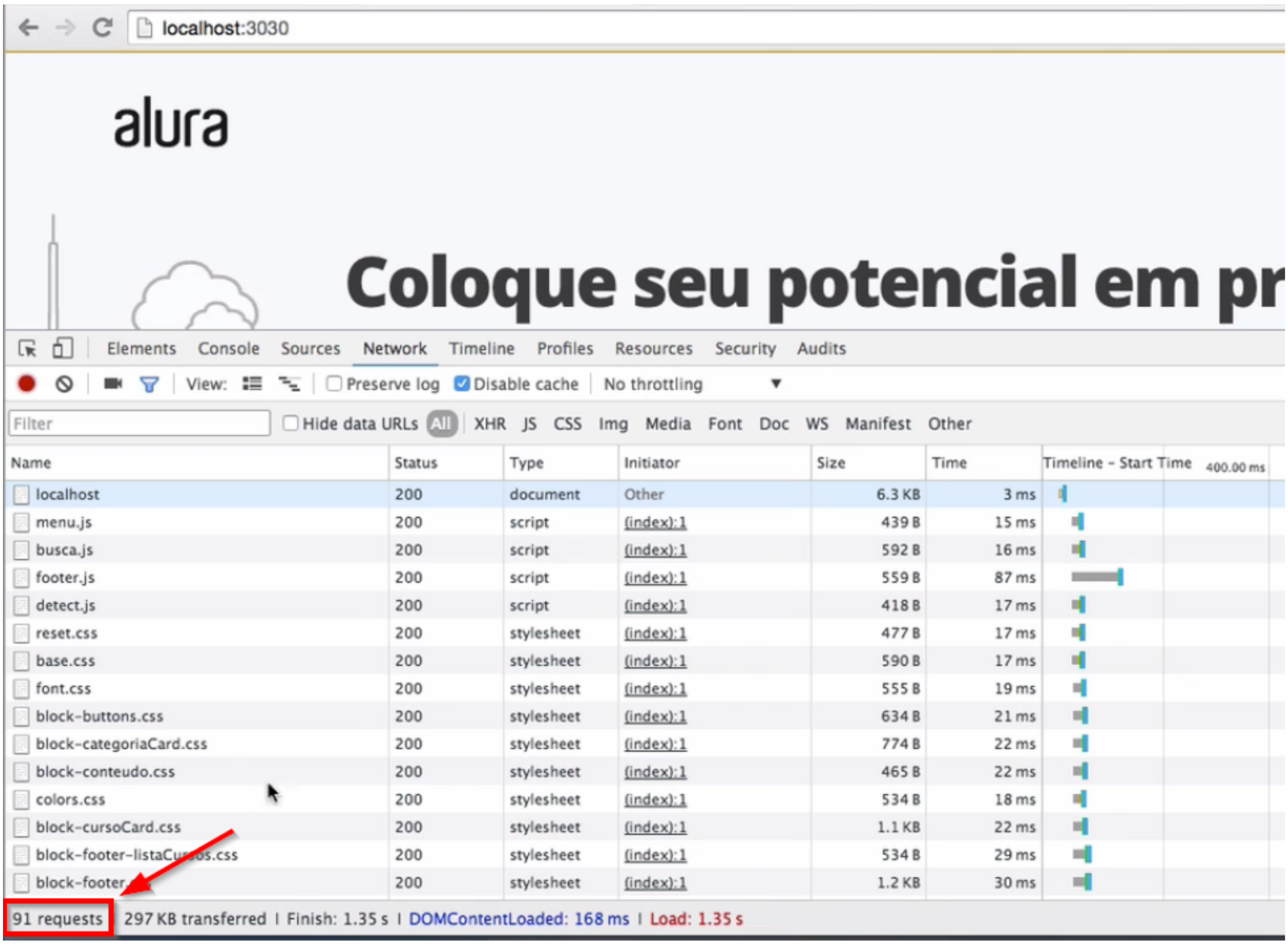
06

Transcrição das aulas

Aprendemos a ver e ler na aba do navegador os gráficos em forma de cascata. Aprendemos que são baixados de seis em seis *requests*. O ponto é que isso pode incorrer em um gargalo ruim de performance. Vamos tentar diminuir a quantidade de *requests* para que eles não fiquem "infinitamente" enfileirados esperando os outros terminarem de baixar para começarem, como atualmente ocorre.

Temos cerca de 91 *requests* e isso é muito:



Por exemplo, se pegarmos os *requests* de *css*, temos cerca de 25 arquivos e isso é muita coisa.:

```html
<link rel="stylesheet" href="assets/css/reset.css">
<link rel="stylesheet" href="assets/css/base.css">
<link rel="stylesheet" href="assets/css/colors.css">
<link rel="stylesheet" href="assets/css/font.css">

<link rel="stylesheet" href="assets/css/block-buttons.css">
<link rel="stylesheet" href="assets/css/block-categoriaCard.css">
<link rel="stylesheet" href="assets/css/block-conteudo.css">
<link rel="stylesheet" href="assets/css/block-cursoCard.css">
<link rel="stylesheet" href="assets/css/block-depoimentos.css">
<link rel="stylesheet" href="assets/css/block-elasticMedia.css">
<link rel="stylesheet" href="assets/css/block-footer-listaCursos.
css">
<link rel="stylesheet" href="assets/css/block-footer.css">
<link rel="stylesheet" href="assets/css/block-form-erro.css">
```

Quem desenvolveu esse site pensou em separar todos os componentes do site em arquivos, portanto, o site inteiro é modularizado.Isso é bom por um lado, pois ele é fácil de ler e acaba sendo mais simples realizar uma manutenção. Entretanto, quando jogamos em produção, temos um gargalo de performance, pois, imagine, temos 25 *requests* e isso apenas de *css*. Se podemos baixar apenas seis simultaneamente, teremos um bom tempo perdido aqui até que se possa baixar o restante dos arquivos, imagens e etc.

Temos que evitar ter tudo isso de *css*. Você pode estar pensando que basta copiar todos os *css* em um único arquivo, um a um, apagar os antigos.... Mas, será que é uma boa ideia? Iríamos apagar os arquivos originais e ficaríamos apenas com um arquivo gigante... não me parece uma boa ideia.

Qual o grande problema?

Queremos manter os arquivos separados por uma questão de manutenção e legibilidade, mas não queremos ao mesmo tempo que isso afete nossa performance de produção.

No fundo, o que estamos querendo fazer é ter um arquivo todo junto em produção, mas em desenvolvimento, ele deverá ser separado para facilitar a leitura e contínuas modificações. Entretanto, isso não é uma tarefa fácil. O que teremos que fazer é automatizar a junção dos arquivos, isto é, automatizar o que gruda essas coisas. Se fizermos isso a mão é possível que sujemos o código e nos equivoquemos no caminho. Podemos fazer isso no próprio terminal digitando um comando bem simples, o *cat*, utilizado para ler algum arquivo.

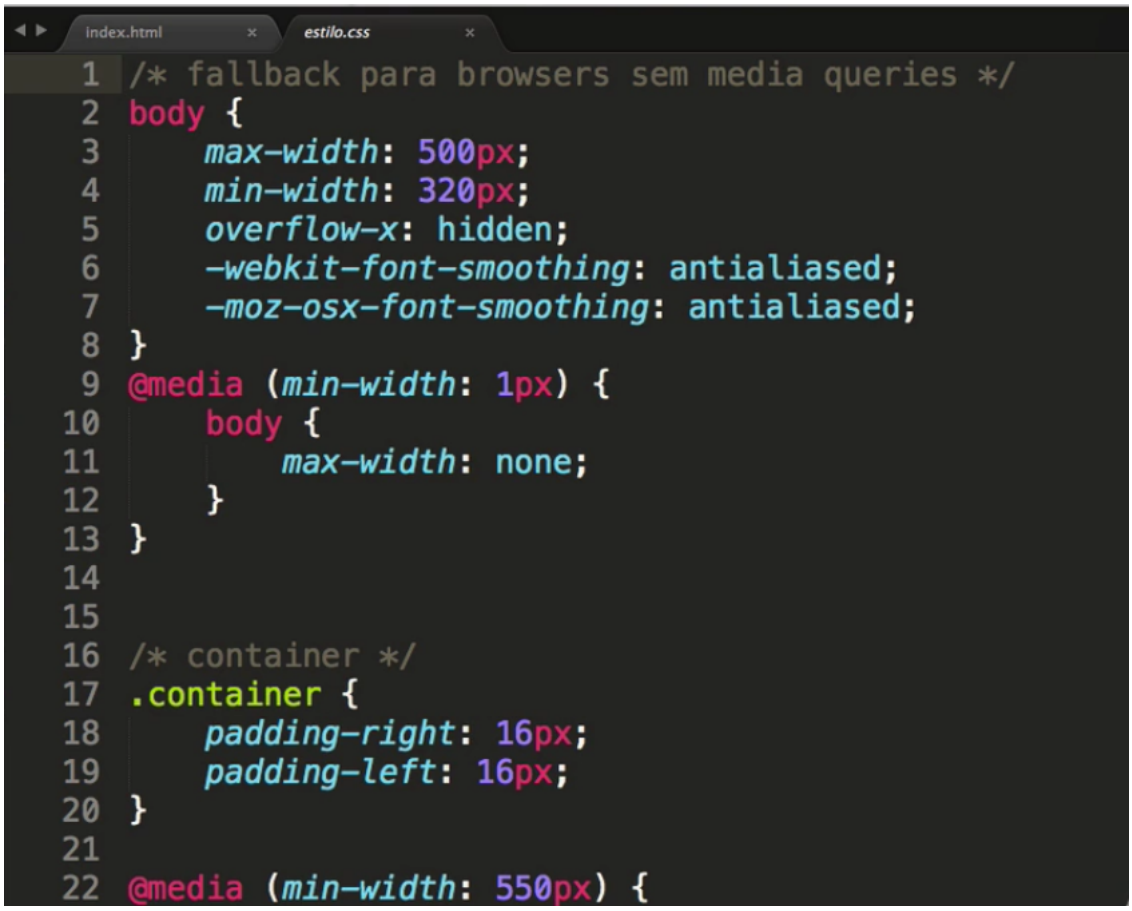
Imagine, então, que queremos ler o arquivo que está em uma determinada pasta, digitaremos o nome do comando seguido da localização da pasta.

```
cat site/assets/css/resete.css
```

Se dermos um *Enter* ele jogará o conteúdo desse arquivo para nós. Podemos pedir para que ele passe todos os arquivos, digitando `cat site/assets/css/*.css` . O que precisamos fazer é jogar todos esses arquivos em algum lugar específico. Para isso digitamos o comando *cat* e também o local onde queremos que esses arquivos estejam:

```
cat site/assets/css*.css > dist/assets/css/estilo.css
```

Utilizando apenas um comando conseguimos resolver esse problema. Nossa pasta "site" continuará dividida, mas a pasta *dist*, o arquivo "estilo.css", estará minificado e com cerca de 2018 linhas.



Falta um detalhe, teremos que alterar o *html* e não faremos isso à mão. Tentaremos automatizar isso utilizando uma ferramenta. Usaremos o *gulp*, que já está pronto. O *gulp* é capaz de concatenar sozinho através de *pluggins* e tarefas específicas e faz esse tipo de trabalho.

Como fazemos isso?

Nós escolhemos um *pluggin* chamado *gulp useref* que faz isso para nós e é bem simples de ser utilizado. A ideia é que só precisamos dizer para ele quais *css* devem ser grudados juntos. Vamos observar a sua documentação no link (https://github.com/digisfera/useref)https://github.com/digisfera/useref (https://github.com/digisfera/useref).

Ele funciona da seguinte maneira:

A ideia é que no código fonte coloquemos várias *tags* e *links* de *css* separados e embrulhemos essas *tags* com um comentário especial, o *build css*. Assim, ele recebe como argumento o nome do arquivo final que deve ser gerado e quando rodarmos o *gulp* ele vai ler esses arquivos, grudar isso e além de gerar o arquivo, vai editar o *index html* para ter a referência atualizada.

Vamos usar esse comando do *pluggin*. Vamos copiar esse comando e colar no nosso arquivo "index.html". Colaremos o seguinte: `<!-- build: css assets/css/estilo.css :`

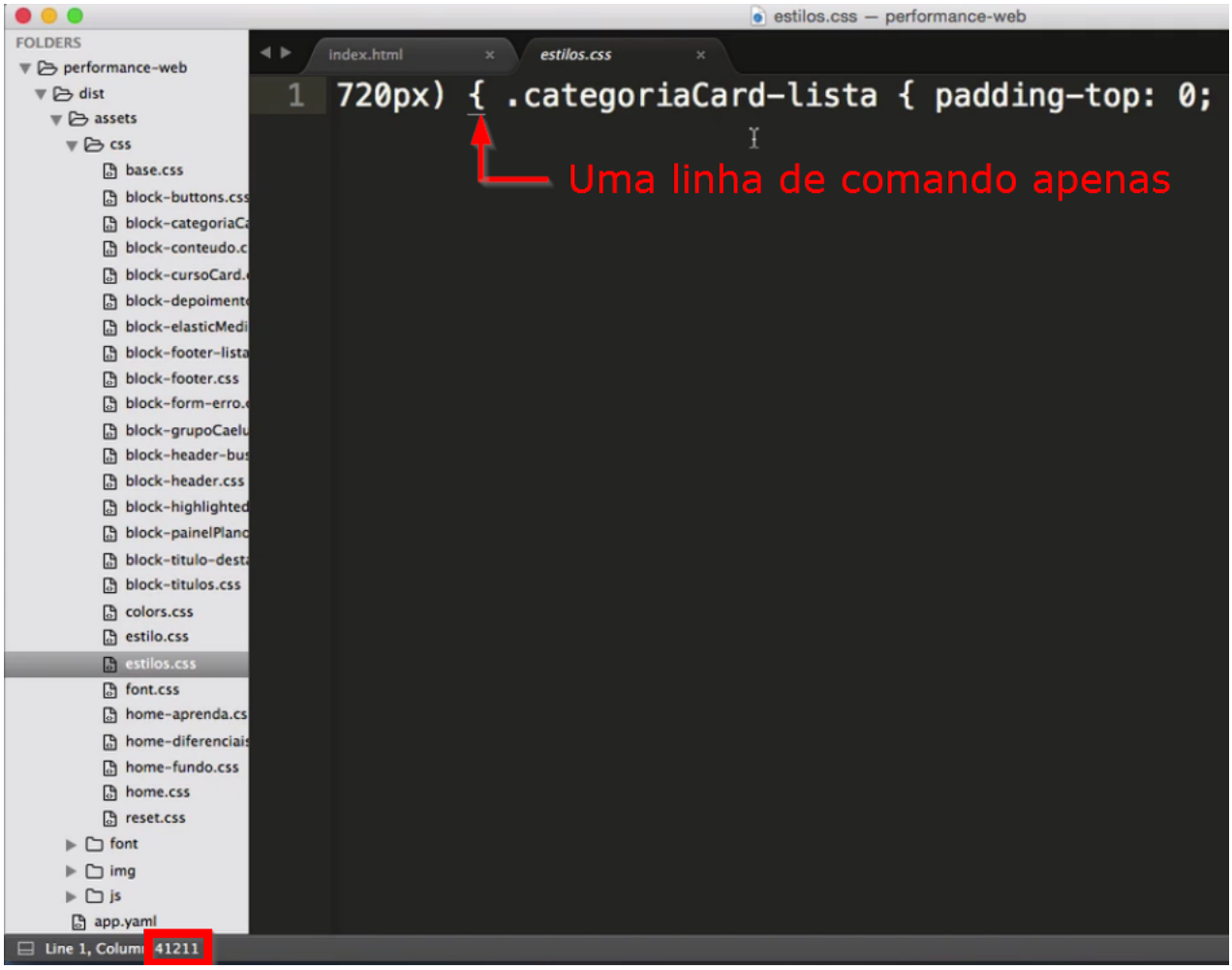
Vamos alterar apenas, onde queremos que isso seja cuspido, nesse caso, queremos que vá para a pasta "assets/css/estilos.css" e copiamos o fim da *tag build* que é `<!--endbuild-->` e colaremos no final de todos arquivos *css* que tivermos nesse arquivo.

Pronto, temos isso embrulhado em um comentário!

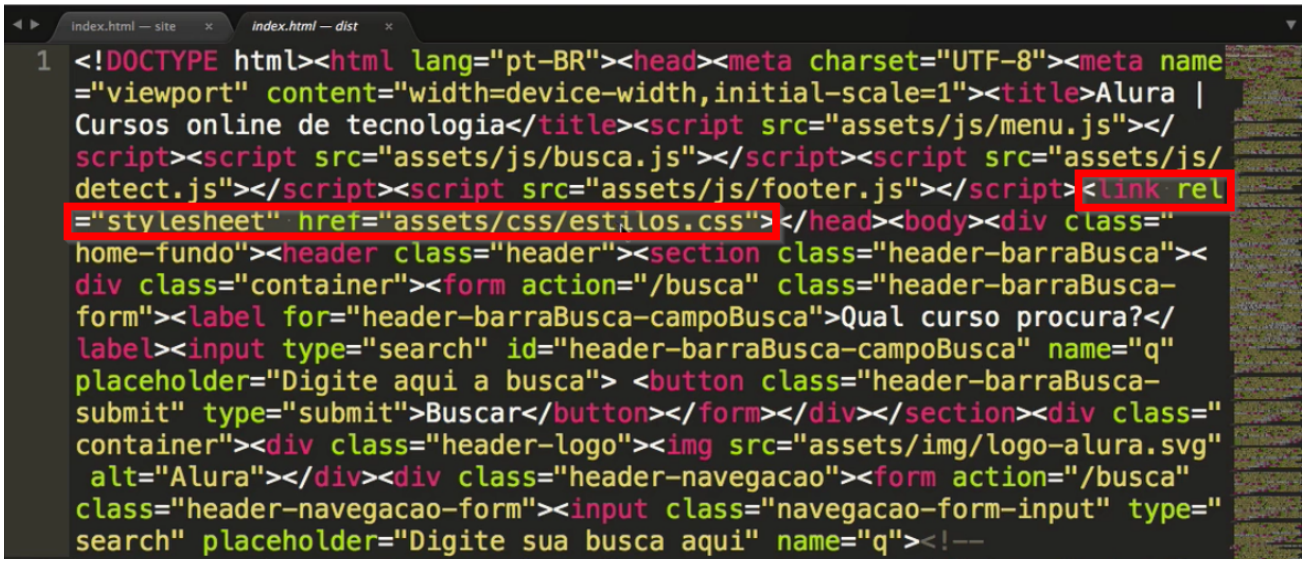
Se rodarmos essa página em nosso navegador, ele continuará baixando os "25 requests", porque no fundo ele ignora o comentário, que é importante apenas para o *gulp*. Vamos fazer o seguinte, vamos rodar na linha de comando o *gulp* e o *pluggin useref*:

```
gulp useref
```

Dando um "Enter" ele começará a realizar isso. Ele vai ler o *index*, procurar os comentários e gerar uma versão otimizada. Vamos ver se isso foi feito? O código fonte permanecerá intacto. A diferença será na pasta "dist" que o "gulp" gerou. Como está no *gulp*, tudo foi minificado. Temos cerca de 41.000 mil caracteres e uma linha de comando apenas.



Tem mais um detalhe, queríamos referenciar um único *css* ao em vez de 25 *css* distintos. Vamos observar o arquivo *index.html* que está minificado. Podemos identificar que temos apenas um *css*:



Se rodarmos o navegador veremos no *dev tools* que temos apenas um *estilos.css*. Se dermos uma olhada no *requests* veremos que já não temos mais a mesma quantidade de *css*.

Será que não podemos melhorar também os *javascript* também? Sim, podemos!

Para isso vamos no "html" original e criamos a mesma *tag* ao redor do "java". Escreveremos no início do *java* `<!--- build:js assets/js/scripts.css-->` e no final `<!--endbuild-->`

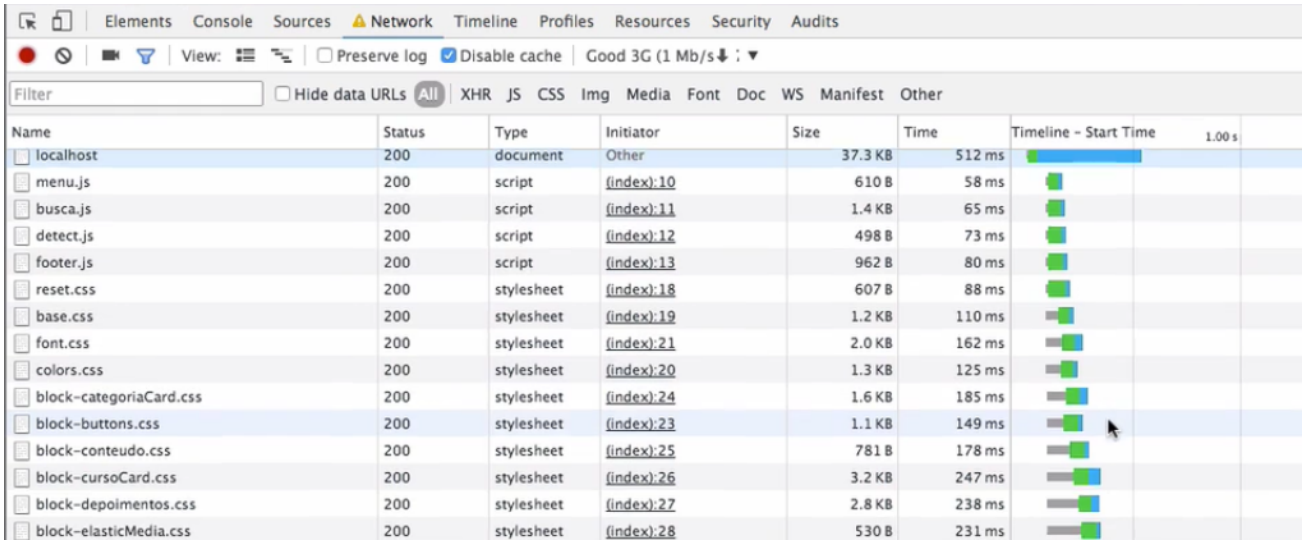
```
9
10    <!-- build:js assets/js/scripts.css -->
11    <script src="assets/js/menu.js"></script>
12    <script src="assets/js/busca.js"></script>
13    <script src="assets/js/detect.js"></script>
14    <script src="assets/js/footer.js"></script>
15    <!-- endbuild -->
16
```

E rodamos no terminal o `gulp useref` e damos um "Enter". Vamos voltar ao "index.html" e podemos ver o *script* único do *java*. O que estamos fazendo é usar o processo de *build* para criar arquivos únicos de *javascript* e *css* sem influenciar no código fonte. Assim, podemos continuar mantendo o desenvolvimento de maneira organizada.

Vamos rodar no navegador para ver como ficou? Olhando o *dev tools* vemos que temos um arquivo "estilos.css" e um único arquivo "script.js".

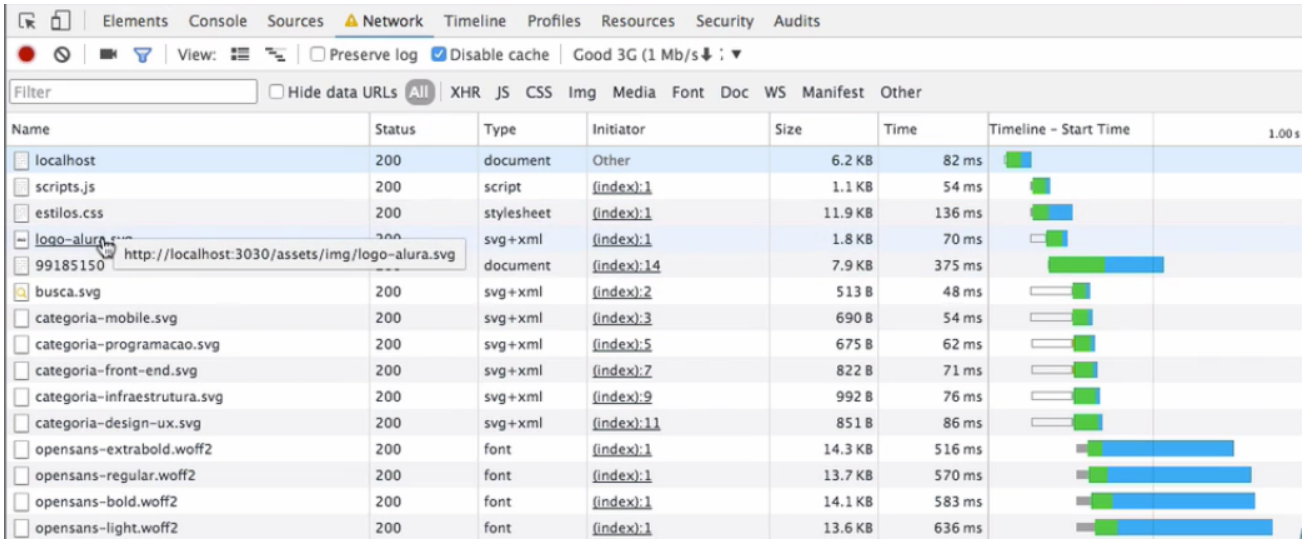
Vamos reparar nas versões otimizadas e não otimizadas. Primeiro, entraremos no "*localhost:2020". Podemos perceber que temos diversos* css *e* javascript *e na versão otimizada temos apenas um* css *e um* javascript*.

Lembra-se que ao utilizar o *localhost* essas diferenças acabam sendo difíceis de serem percebidas? Vamos alterar nossa rede, escolheremos uma "3G" para fazer uma comparação. Vamos rodar a versão não otimizada:



Podemos ver que ele lança o primeiro *request* e este vai encavalando os demais *requests* de forma que ele aguarda muito tempo para baixar os próximos.

Vamos analisar a versão otimizada:



Como temos seis conexões simultâneas ele baixas próximas coisas logo em seguida e vemos que isso causa menos tempo cinza, isto é, menos tempo de espera do que se compararmos com a versão não otimizada.

Então, podemos fazer isso que nos traz bastante vantagens, mas gostaria de saber o que perdemos fazendo isso?

Temos que ter em mente alguns problemas que podemos enfrentar, primeiro, quando colocamos tudo em um arquivo só, do ponto de vista da produção, estamos perdendo a capacidade de cachear os arquivos individualmente, se mexermos uma linha de um arquivo, terá que baixar tudo de novo. Se mexêssemos uma linha de um arquivo em outro cenário ele iria baixar apenas aquele arquivo que foi alterado. Então, se você tiver um site muito grande, com muito *kbytes* e *megas* de *javascript* e *css* será disso que você estará abrindo mão ao fazer essas alterações que causam novos downloads de vários megas.

Falamos de uma desvantagem, vamos falar também de uma vantagem que é o *gzip*, quando concatenamos tudo em um arquivo só o algoritimo do *gzip* roda de uma vez só e o resultado é evidente.

Um outro ponto que temos que pensar bem é o fato de termos juntados os 25 arquivos de *css* , o site foi originalmente feito de maneira a ser todo modular, onde cada componente da página está em um *css* diferente e é um arquivo separado e fácil de ler e etc... Mas o ponto é, será que nessa *home* usamos todos os componentes do site inteiro? Como fazemos quando temos um site com cinco páginas e cada página usando componentes diferentes. O que podemos fazer? Concatenar tudo em um arquivo apenas? E assim, toda vez que ele abre a *home* ele baixará os componentes de páginas pelas quais o usuário ainda nem navegou?

É complicado. Escolher onde a quebra será feito pode ser algo não tão trivial. As vezes não conseguiremos juntar tudo isso em um arquivo só, teremos que quebrar em dois ou três arquivos. Esse tipo de decisão, na prática, não é tão simples e veremos sobre concatenações e *http2* mais adiante, pois envolve alguns detalhes a mais.

Por enquanto, queríamos mostrar o impacto de se diminuir os *requests* fazendo isso através da concatenação de *javascript* e *css* que é na verdade uma boa prática.