01

# Workflow & Viewport

## Transcrição

Neste vídeo vamos falar um pouco sobre *viewport* e fluxo de desenvolvimento quando trabalhamos com sites responsivos, principalmente para testes no mobile. Estávamos fazendo nosso site de forma fluida, com *media queries*, bem interessante, testando na versão desktop, redimensionando a janela, tudo funcionando bem. Não sei se você parou para testá-lo em seu celular, caso tenha feito isso, talvez o resultado não tenha sido tão interessante.


Acima, vemos nossa página aberta no emulador do iPhone, e percebemos que todo aquele trabalho para deixar tudo responsivo foi "por água abaixo", pois o aparelho está abrindo a versão desktop, com o *zoom out*, letras bem pequenas, nenhuma *media query* sendo aplicada, nada do tipo. Isto está acontecendo porque não acrescentamos o tal do *viewport*.

Para entendermos sua necessidade, há muito tempo, quando os iPhones surgiram, a maioria dos sites ainda não eram responsivos, e sim desktop. O iPhone talvez foi o primeiro celular a implementar um navegador completo, capaz de abrir não apenas sites em versão mobile, como desktop também. Para que eles fossem compatíveis com todos estes dispositivos, não era desejado que todos os sites feitos para desktop abrissem "quebrados", então, por padrão, em seu surgimento, ele adotou a estratégia de "fingir" que a página fosse aberta em uma versão desktop.

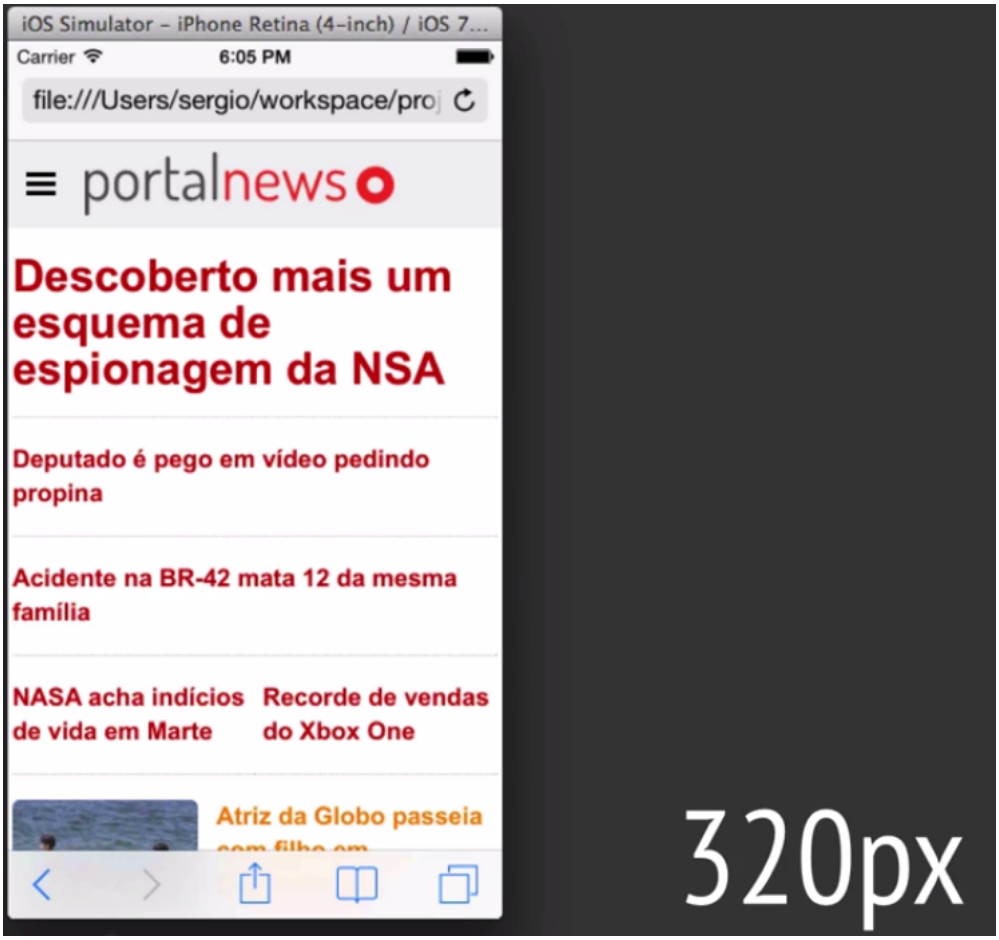
Isto quer dizer que o dispositivo encara a resolução do iPhone como se fosse de `980px`. Outros fabricantes acabaram copiando esta ideia utilizando valores ligeiramente distintos, mas a ideia é a mesma: por padrão, se o site não informa nada, assume-se que ele está na versão desktop, por questões de compatibilidade, abrindo-se a tela pequena como se fosse grande.

O iPhone tem `320px` de largura, então por que ele abre em uma versão de `980px`? Para se obter esta compatibilidade. Sites desenvolvidos responsivamente continuarão funcionando bem em telas menores, no entanto é preciso avisar o dispositivo

sobre isto, através de uma meta tag chamada *viewport*:

```
<meta name="*viewport*" content="width=320">
```

Sua sintaxe é bem simples, com `width` definindo a largura de tela que queremos usar, em vez de "fingir" estarmos utilizando um desktop. O resultado fica assim:




A página está adaptada, as *media queries* são executadas; repare que a necessidade do *viewport* está em comunicarmos ao dispositivo que nossa página está adaptada para a versão mobile, sendo assim uma página responsiva.

No entanto, `320px` é o tamanho de tela de um iPhone. E quando queremos adaptar a página a outros dispositivos? Ou mesmo quando viramos a tela do próprio iPhone, alterando sua largura de visualização, deixando-a em modo paisagem? Não podemos utilizar este valor como sendo algo fixo, pois assim "travamos" em uma única resolução em detrimento de várias outras possíveis (no caso de iPads, Androids, Windows Phone, e assim por diante).

Na prática, não costumamos utilizar este código, uma vez que existe outro, o qual utilizaremos em todas as páginas, sendo bem parecido, com a diferença de que definimos a `width` como sendo a largura específica do dispositivo:

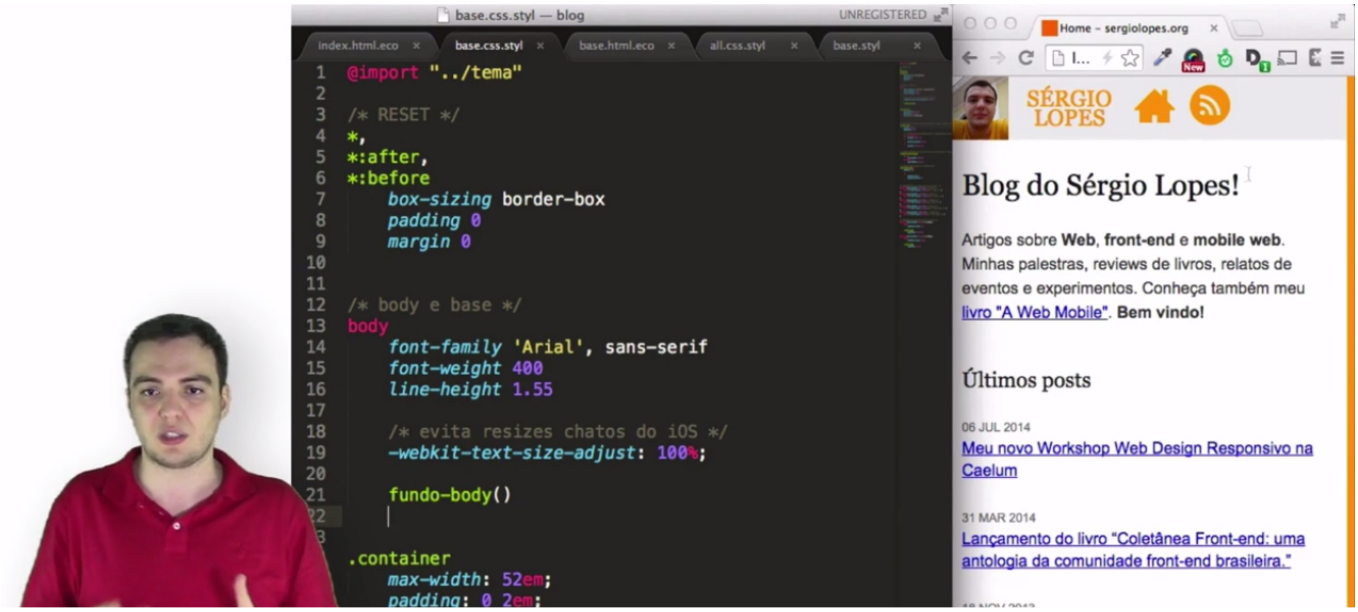
```
<meta name="*viewport*" content="width=device-width">
```

Quer dizer que estamos instruindo o dispositivo para que ele utilize sua largura nativa. Ou seja, se for um iPhone, `320px` . Se for um Android desses mais modernos, `360px` , um iPad, `768px` , entre outros. Agora sim, você pode copiar e colar este código em todos os seus projetos responsivos. Feito isto, podemos abrir nossas páginas nos diversos dispositivos e tudo ocorrerá como esperado.

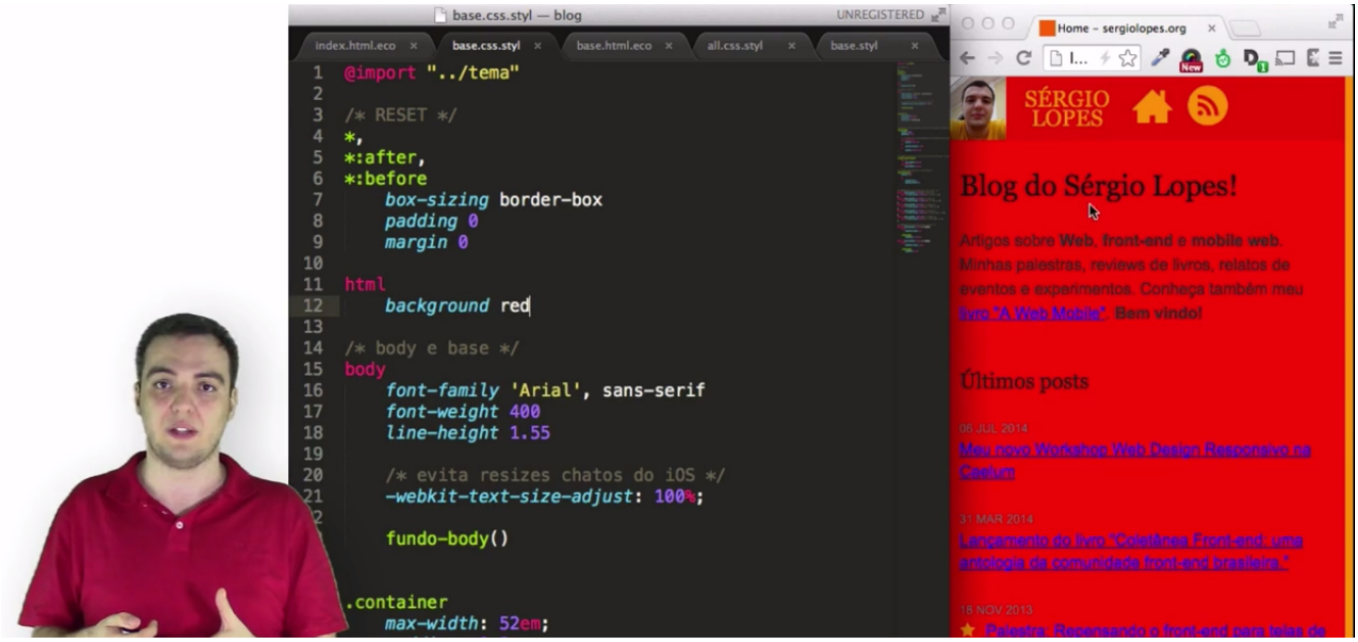
Para começarmos a testar nossa página em variados dispositivos, gostaria de discutir sobre estratégias de testes, pensando-se em dispositivos móveis, propondo-se três itens no momento do desenvolvimento: browser do desktop, em conjunto com alguns emuladores de dispositivos, e dispositivos reais.

Veremos os usos de cada um deles, assim como a melhor maneira de se adaptarem ao fluxo de trabalho, ou *workflow*. Começando pelo browser no desktop, como é que costumamos fazer? Abriremos o editor de texto e a página no desktop; como exemplo, estou utilizando meu blog e um editor qualquer.

Expandindo-se o navegador, vemos que a página vai sendo adaptada. Mantendo o editor de texto aberto ao lado, facilita-se a visualização de quaisquer mudanças feitas no navegador, de forma automática, e este é um dos motivos pelos quais utilizamos o desktop browser para desenvolvimento.



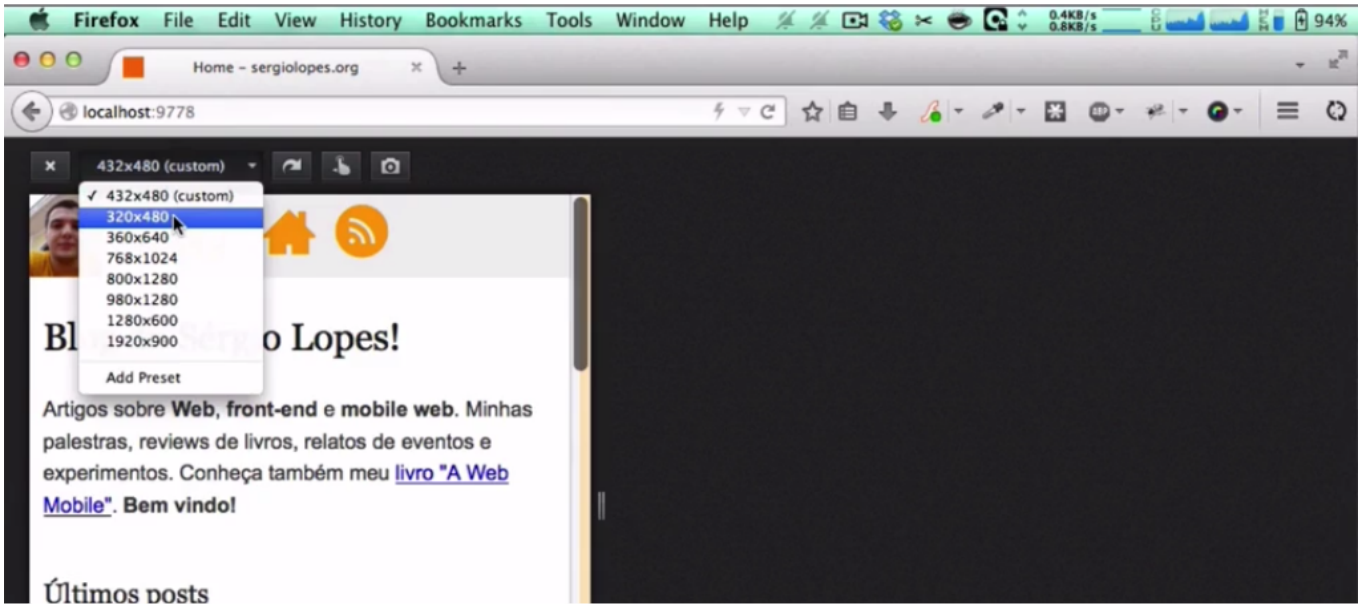

O interessante, também, é realizar alterações não apenas no HTML, e sim no CSS. Para fins de teste, vamos colocar um fundo vermelho na página:




Podemos fazer outros testes, de resolução de tela, acrescentando *media queries* por exemplo, um *background* de cor amarela que só será disparado em uma tela com `width` mínimo de `500px`. A ideia, portanto, é de utilizarmos o navegador desktop para facilitar o desenvolvimento, alterando-se o código e verificando-o em tempo real.

Utilizando o redimensionamento e a ferramenta de inspecionar a página, conseguimos realizar alterações diretamente no desktop, algo muito bom, prático e mais fácil de se fazer do que com um celular no bolso, por exemplo.

Neste exemplo, utilizamos o Google Chrome, mas poderíamos usar qualquer outro navegador. No Mozilla Firefox, existe um modo denominado "Responsivo", localizado no menu de ferramentas ("Tools > Web Developer > Responsive Design View"). Ele também possibilita redimensionamentos, porém dentro da própria janela do navegador, o que acaba facilitando um pouco mais. Também é possível trocar a orientação, criar outras, inspecionar elementos, tudo para facilitar o processo de redimensionamento de janela, simulando dispositivos diferentes no próprio desktop.




Uma das últimas funcionalidades incluídas no Chrome é bastante interessante, especialmente para desenvolvedores de sites mobile responsivos. Quando abrimos o inspetor em uma página qualquer, há um ícone de celular no canto esquerdo superior, com opções de emulação. Vamos clicar nele e ver o que ele faz. Incluindo emulação de tamanho de tela, com recursos novos que permitem que escolhamos um dentre vários modelos comuns de celulares disponíveis no mercado, redimensionando a janela para aquele determinado padrão.

No entando, ele também simula várias características daquele aparelho, como a bolinha que simula a ideia de *touch screen*, permitindo eventos de *touch*, e também o *device pixel ratio* da tela de retina, por exemplo. Além disto, é possível simular um *User agent* diferente para um determinado celular ou dispositivo. São várias ferramentas que atuam imitando o dispositivo com o qual estou trabalhando.

Outra opção que não tem a ver com o dispositivo, mas com a rede: podemos simular o uso de rede 3g, por exemplo, para verificar a performance da página. Às vezes é interessante testarmos localmente, mas isto não nos traz uma noção muito boa da real demora de uma página. Conseguimos simular esta rede através do menu *Network*.

Além de simulação de rede em diferentes dispositivos, pode-se utilizar um editor de *media queries* do Chrome, disponível no menu do lado esquerdo da página, ao qual se abrem todas as *media queries* sendo aplicadas naquele momento na página. Podemos clicar em cada uma delas, resultando em um redimensionamento específico; é bastante útil quando existem muitos códigos e *media queries*, para saber o que cada uma está fazendo, quais são os *breakpoints*. Não se trata de um dispositivo Android nem um Chrome do Android, e sim de desktop, com algumas opções para facilitar seu uso no dia a dia, em seu desenvolvimento, com testes de redimensionamento, e assim por diante.

Além de usar navegadores desktop para estas simulações, vamos precisar também de dispositivos reais mas, antes, podemos realizar tudo isto através de emuladores. Neste caso estou utilizando um do iOS que roda apenas no Mac. Se este for seu caso, pode instalar o Xcode, clicando posteriormente em "Xcode > Open Developer Tool > iOS Simulator".

Ele abre um simulador de iOS dentro da sua máquina, fiel ao simulador de iOS do seu dispositivo. Vamos utilizar o simulador e o Safari no desktop para debugação de páginas dentro do simulador. Abrindo-se o simulador, clicamos no ícone do Safari, em que carregamos uma página qualquer. Podemos debugar esta página pelo Safari do desktop. Clicaremos em "Safari > Preferences > Advanced" para habilitar as opções de desenvolvedor, marcando-se o *checkbox* de "*Show Develop menu in menu bar*".

Na barra de ferramentas do Safari, selecionaremos "Develop > iPhone Simulator > localhost". Então, conseguirmos verificar todas as abas abertas no simulador que, no caso, é apenas uma. Abrimos esta aba, vinculada ao iPhone *Simulator*. Repare que quando alteramos um elemento no Inspetor, altera-se no simulador também. Portanto, conseguimos editar CSS, HTML, fazer tudo aquilo que costumamos fazer com os Inspetores.

Voltando ao simulador e selecionando "Hardware" na barra de ferramentas, iremos à "Device", que nos mostra opções de dispositivos e modelos com os quais podemos trabalhar. Além do iPhone, há a versão Retina, iPads, várias outras.

Escolheremos o iPad, por exemplo, e o simulador abrirá todas as suas funcionalidades naquela versão específica, podendo-se instalar várias outras, o que possibilita o *debug* no iOS do Safari desktop tranquilamente. Podemos selecionar a escala `1:1` clicando em "Window > Scale > 100%", para usarmos o Safari pelo iPad.

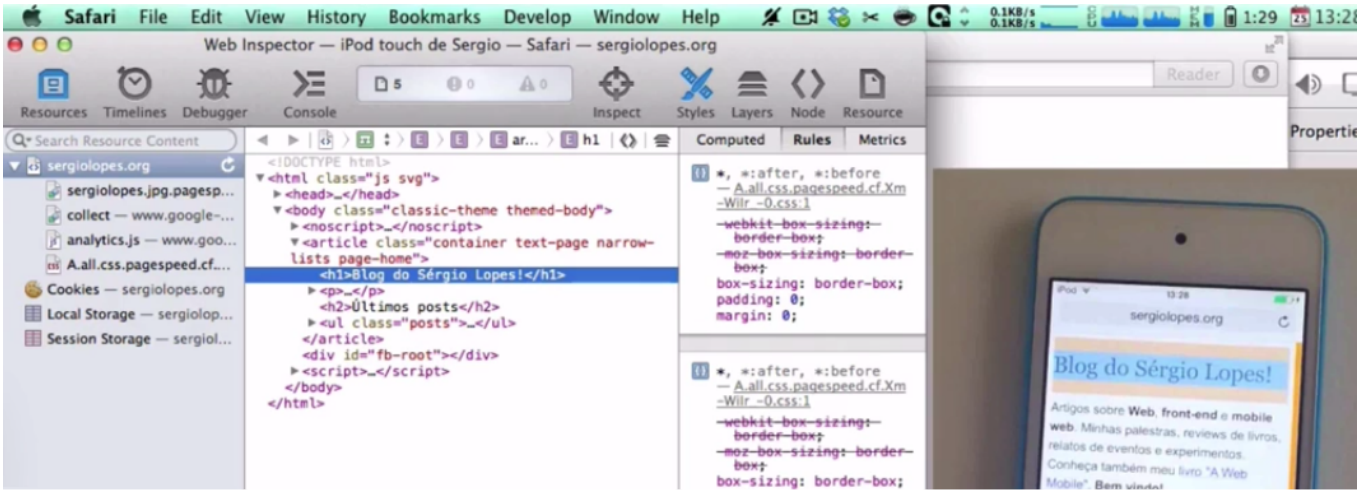
A vantagem do simulador de iOS é que ele é extremamente fiel ao dispositivo, portanto conseguimos simular todos os *bugs* de diferentes dispositivos através dele, sem necessariamente termos que adquirir o aparelho especificamente para este tipo de teste. Além disto, pode-se obter o mesmo comportamento para Android, instalando-se um simulador dele em sua máquina; ele é bem pesado, mas não é impossível de ser instalado.

O mesmo ocorre com o Windows Phone; caso você utilize a versão Windows `8`, pode-se instalar o simulador e realizar estes mesmos testes. É possível utilizar, portanto, simuladores e emuladores para trabalhar com o dispositivo sem ter que comprá-los. Na prática, você acabará precisando de vários dispositivos para testes em relação às práticas e experiências de uso, sensação do *touch screen*, performance, entre outros.

Para fazermos o *debug* trabalhando de forma remota com o dispositivo real, estamos utilizando o dispositivo iOS mais uma vez, um iPod *touch*, plugado via cabo USB ao Mac (infelizmente precisamos de um Mac para debugarmos um iOS, por restrições da Apple), abrindo o navegador Safari tanto no desktop quanto no mobile.

O primeiro passo a ser feito é configurar o iOS para se habilitar a inspeção remota. Na parte de ajustes do iPhone, iremos às configurações (ajustes), selecionando "Safari > Avançado" e habilitando a opção "Inspetor Web". Voltando ao Safari pelo celular, verificamos que tudo funciona conforme esperado.

Indo ao desktop e selecionando, na barra de ferramentas, "Develop > iPod touch de Sergio [nome configurado ao seu dispositivo]", vê-se que o mesmo menu utilizado pelo emulador está vinculado ao dispositivo real. Abrimos o dispositivo e conseguimos mexer no HTML, CSS, podemos editar, alterar algo, remotamente.




É muito simples fazer este *debug* com o dispositivo real, apesar do Mac ser necessário quando se trata de um iOS. Se estiver trabalhando com um Android, em especial com o Chrome no Android, este navegador permite *debug* remoto em qualquer sistema operacional.

Utilizaremos como exemplo um Android no Nexus 4, com o Google Chrome instalado e plugado via cabo USB em meu desktop. No celular, estamos com a página aberta, então vamos ao desktop e digitamos `chrome://inspect`, o que mostra todos os dispositivos conectados - neste momento, não há nenhum. No dispositivo mobile, vamos em "Opções do desenvolvedor" do Android, habilitando-se a opção "Debugação USB".

Quando clicamos nesta opção, vimos que a ação se reflete no Chrome do desktop, passando a mostrar o dispositivo, listando todas as abas existentes. Podemos clicar em "Inspect", abrindo um inspetor vinculado àquela aba no mobile.

Um ponto importante: este é um *debug* remoto do Google Chrome no Android, não do navegador Android padrão, que muitos dispositivos têm. O navegador padrão do Android dificulta um pouco o desenvolvimento... Aliás, são vários os navegadores que não suportam este tipo de *debug*, como o navegador padrão do Windows Phone e do Android, em versões mais antigas.

Para se trabalhar com dispositivos que possuem navegadores nativos que não realizam *debug* remoto e saber quais erros estão ocorrendo, que alterações precisam ser feitas, usaremos como exemplo o Windows Phone, que só precisa estar conectado à mesma rede Wi-Fi da máquina. Utilizaremos uma ferramenta gratuita chamada "Weinre", que disparamos à linha de comando, o qual, por sua vez, dispara um servidor, que linka o navegador desktop em uso ao mobile, permitindo-se a inspeção remota. Não é uma inspeção tão avançada quanto o Chrome ou o Safari fazem, porém é interessante nestes cenários com navegadores sem este suporte.

No terminal do desktop, digitaremos `weinre` na linha de comando, e seu servidor será disparado. Usarei a porta de onde foi disparada esta `weinre`, abrindo um navegador qualquer e acessando meu IP ( `192.168.1.110:8099` ), naquela porta em que estivermos trabalhando. Mais abaixo, na mesma tela que foi aberta, haverá um código em JAVASCRIPT que precisa ser colado em `index.html` da minha página, que abriremos no mobile.

Clicaremos na opção de *debug* (cuja url é `192.168.1.110:8099/client/#anonymous`), aguardamos um pouco, vamos ao celular, abrimos (no caso o Internet Exprorer), digitando a mesma url que está na rede, sendo portanto do IP da máquina em que o serviço está sendo rodado. No mesmo instante, o `weinre` do desktop encontra a página em questão, nos possibilitando um *debug* remoto. É bem parecido com o que tínhamos feito antes: podemos alterar elementos, editar o CSS, entre outros.

Através do `weinre`, podemos linkar, realizar *debug* remoto em browsers e dispositivos com sistemas operacionais que não possuem suporte para isto, como o Windows Phone ou um Android mais antigo, ou mesmo para um iPhone, se você não tiver um Mac para debugação.

Por fim, gostaria de mostrar que estamos utilizando uma estratégia em nosso desenvolvimento: desktop *browser*, emuladores, e *debug* remoto em dispositivos reais.

Em geral, utilizamos o desktop *browser* pois ele te traz mais produtividade e ferramentas para facilitar o dia a dia do desenvolvimento. Faz-se as alterações no código, que podem ser vistas simultaneamente no desktop, redimensionando-se a tela, como vimos nos primeiros vídeos. Lembrando que em algum momento será preciso fazer testes em um dispositivo de verdade.

Para tal, existem emuladores, que costumam ser mais baratos, bastando tê-los instalados em sua máquina, desktop. Com emuladores, pode-se lidar com diferenças de interfaces, bugs, compatibilidades, versões. Há grande variedade de emuladores, muito mais do que a quantidade de dispositivos diferentes a serem comprados.

> Para a etapa de testes em dispositivos reais, uma dica é comprar um iPod Touch, mais barato que um iPhone (se você já não for usuário deste).

O importante é encontrar dispositivos e navegadores que sejam significativos e adequados aos seus usuários finais. Isto pode ser definido analisando-se as estatísticas de acesso. Não adianta ter vários dispositivos caros ou "da moda" à mão sendo que a maior parte de seus usuários utiliza outro tipo. Por exemplo, sabemos que Androids mais simples e baratos são mais comuns no mercado, então é preciso testar nestes.

> É recomendável, também, montar-se um laboratório de dispositivos (*device lab*), através do qual será possível executar vários destes testes.

Relembrando o que vimos neste vídeo, o fluxo de desenvolvimento envolve o *viewport*, bastante importante para a correta adaptação das páginas nas variadas telas, e também browsers no desktop para auxílio no dia a dia do desenvolvimento, assim como emuladores oficiais, *debug* remoto, principalmente pensando-se em um laboratório de dispositivos reais, físicos.

Faremos alguns exercícios práticos a seguir, e continuamos em outro vídeo!