O QUE É PROGRESSIVE ENHANCEMENT?

Quando desenvolvemos para web, sempre sofremos com a variação de suporte às tecnologias existentes no mercado. Nosso querido amigo Internet Explorer, por exemplo, durante muitos anos não seguia por completo as definições e padronizações existentes na web, mas será que o mesmo é o único que possui limitações? A resposta é simples: não. Existem vários navegadores e ferramentas que possuem deficiência para acessar um conteúdo na web, e que demandam que páginas sejam escritas de forma semântica e acessível. Entre eles, posso mencionar ferramentas de leitura para cegos, ferramentas de indexação, navegadores de linha de comando (sim, eles existem! E quem usa linux com frequência, já deve ter usado um deles).

Por questões como essas, surgiram algumas técnicas para tentar garantir a navegação mínima de sistemas e páginas web em diferentes ferramentas e navegadores, porém as mais conhecidas são o "progressive enhancement" e o "beautiful degradation". Nesse artigo, vou me focar no primeiro.

A DEFINIÇÃO

Progressive enhancement é uma estratégia de desenvolvimento de páginas web com ênfase na acessibilidade, utilização de tags HTML semânticas, estilos CSS externos e JavaScript não obstrusivo. Nessa estratégia utilizam-se as tecnologias web em forma de camadas, permitindo que todos acessem o conteúdo básico de uma página independente de navegador e link de internet, porém, quanto maior o suporte às tecnologias visuais ou recursos mais recentes de navegação, mais rica é a experiência do usuário.

PRINCÍPIOS

Os pilares do progressive enchancement são:

1. O CONTEÚDO BÁSICO DEVE SER ACESSÍVEL EM TODOS OS NAVEGADORES

O conteúdo que deseja transmitir para o usuário deve ser feito da forma mais básica possível com título, parágrafos, citações, links, etc.

Imagens, músicas e animações não devem ser consideradas como informação básica na maior parte dos casos, no entanto, quando o entendimento das mesmas se fizer necessário, é imprescindível que as mesmas estejam devidamente descritas com o uso dos atributos alt e title.

2. FUNCIONALIDADES BÁSICAS DEVEM SER ACESSÍVEIS EM TODOS OS NAVEGADORES

Botões e links de navegação devem funcionar corretamente em todos os navegadores quando essenciais para a navegação na página (e sim, isso inclui o IE). Caso a funcionalidade não seja imprescindível e não seja possível aplicá-la usando recursos

simples de navegação, a mesma deve ser removida dos navegadores que não possuírem suporte ao recurso.

3. MARCAÇÃO SEMÂNTICA EM TODO O CONTEÚDO

O uso do HTML deve seguir corretamente uma estrutura semântica, sendo assim, div's aninhadas para criar efeitos visuais devem ser evitadas.

4. LAYOUT É FORNECIDO VIA CSS LINKADO EXTERNAMENTE

Não utilize style inline nem utilize a tag style. O CSS deve ser chamado através de uma tag link, pois assim, quando o navegador não oferecer suporte à CSS, esses dados nem serão carregados.

5. COMPORTAMENTO É FORNECIDO VIA JAVASCRIPT NÃO OBSTRUSIVO, LINKADO EXTERNAMENTE

Não manipule eventos por código no HTML e não insira a tag script no meio do conteúdo. A página deve funcionar sem que scripts sejam carregados e permitir o carregamento posterior quando o suporte ao mesmo estiver disponível.

6. PREFERÊNCIAS DO NAVEGADOR DO USUÁRIO DEVEM SER RESPEITADAS

Se o usuário alterar as definições do tamanho de fontes ou desabilitar o JavaScript, não o force a mudar suas configurações. Um exemplo é usar medidas em "em" para fontes ao invés de usar medidas em pixels, já que isso permite manter as medidas relativas às configurações que o usuário fez em seu navegador.

O PROCESSO DE DESENVOLVIMENTO

Você deve estar pensando: "Certo, então para meu site ser acessível usando essa técnica, eu devo deixá-lo o mais feio possível, e não usar os recursos interessantes como efeitos visuais e animações, nem páginas usando os recursos de SPA (Single Page Application) ??". Na verdade, você pode usar tudo o que você quiser =] . O processo de desenvolvimento continuará sendo o mesmo e você não deve ter problemas. O que ocorre é que se o navegador não tiver suporte a "algumas extravagâncias", o conteúdo será exibido de forma simplificada, porém 100% funcional.

Por exemplo: podemos fazer requisições utilizando AJAX sem nenhum problema! No entanto o carregamento completo da página (por page reload) deve estar disponível caso os recursos para o uso de AJAX não estejam disponíveis. De forma simples, para alcançar os objetivos do progressive enhancement, você poderia:

1- Primeiramente desenvolver a página usando apenas HTML, de forma clara, legível, semântica e minimamente apresentável;

2- Após a produção do HTML, você poderia aplicar uma primeira "camada" de CSS, para aprimorar posicionamentos, cores, fundos, fontes e afins com recursos básicos e amplamente disponíveis nos navegadores (esse seria o primeiro "enhancement" ou melhoria na navegação básica);

3- Em seguida você poderia desenvolver uma nova camada de CSS com recursos mais bonitos como sombras, transparências, transformações e afins (que seriam a segunda melhoria. Quem não tiver suporte a essa camada, continuará vendo as melhorias aplicadas na camada anterior);

4- Por último, você poderia aplicar uma camada de JavaScript, alterando as navegações antes feitas por "page reload" para links utilizando técnicas de AJAX, aplicar transições e outras estripulias, já que usuários sem suporte a JavaScript, já estão "protegidos" pelas camadas anteriores (obviamente, desde que os scripts aqui criados sejam não obstrusivos).

Como é possível ver, foram feitas "melhorias progressivas", e N camadas podem ser aplicadas, até se atingir o resultado esperado em navegadores modernos, porém, ainda fornecendo uma navegação mínima a quem não terá suporte a essas novas camadas.

UTILIZE A TOLERÂNCIA A FALHAS DO NAVEGADOR A SEU FAVOR

Podemos tirar proveito da tolerância a falhas dos browsers no HTML e no CSS: os navegadores, por definição, ignoram tags HTML e propriedades CSS das quais eles não conhecem! Pois é! E isso ocorre desde sempre, já que garante de certa forma que o que esta sendo desenvolvido hoje, funcione de alguma forma daqui a alguns anos.

1. TOLERÂNCIA A FALHAS NO HTML

```
<nav>
    <ul>
        <li><a href=”#”>Design</a></li>
        <li><a href=”#”>Soluções em tecnologia</a></li>
        <li><a href=”#”>Contato</a></li>
        <li><a href=”#”>Blog</a></li>
    </ul>
</nav>
```

O exemplo acima mostra um exemplo de tolerâncias à falhas, pois a tag nav só foi incluída no HTML5. Mesmo assim, navegadores que não conhecem essa tag, exibirão a lista interna normalmente. Um efeito colateral nesse caso, é que navegadores que não conhecem essa tag, não irão renderizá-la, e por consequência, estilos CSS aplicados a mesma não serão visíveis. Uma alternativa para que os estilos sejam carregados mesmo que o navegador não conheça a tag aplicada, é usar o html5shiv.

2. TOLERÂNCIA A FALHAS NO CSS

```
h2 {
    background-color: rgb(255, 0, 0);
    background-color: rgba(255, 0, 0, .1);
}
```

O exemplo acima tira proveito da tolerância a falhas do navegador utilizando sobrescrita de propriedades. Se o navegador reconhecer a propriedade rgba ele irá sobrescrever a propriedade anterior, caso contrário a propriedade anterior (rgb) será mantida.

USANDO UM COMPORTAMENTO CONDICIONAL

A tolerância a falhas não resolve todos os problemas, pois existem situações onde a solução não é simplesmente sobrescrever uma propriedade CSS, e sim utilizar propriedades diferentes para um efeito parecido. Por exemplo: simular uma sombra com a propriedade border e criar uma sombra com box-shadow.

Existem também casos em que a função JavaScript utilizada, não estará disponível no navegador em uso pelo usuário. Nesses casos podemos contornar o problema utilizando técnicas de detecção e ajustes condicionais. Lembrando que o JavaScript deve ser não obstrusivo, ou seja, não deve impedir a navegação!

1. POLYFILL

Polyfill é usado para a detecção da existência de uma funcionalidade e, quando o mesmo não está disponível, é criado um recurso "tapa buraco" para que o recurso funcione. Isso permite que mesmo em alguns navegadores que não possuam uma determinada função, o aprimoramento em questão esteja disponível para o seu usuário.

2. FEATURE DETECTION

```
if(window.Notification){
    Notification.requestPermission();
    new Notification("teste");
}
```

Uma forma de não gerar erros quando se utiliza um recurso não suportado é verificar se o mesmo existe. Uma forma simples de se atingir essa verificação é fazer um if! No exemplo acima, a condicional no "if" verifica se existe suporte a Notification.

3. FEATURE DETECTION COM MODERNIZR

```
Modernizr.load({
    test: Modernizr.notification,
    nope: ['script/notification-polyfill.js']
});
```

Caso não queira ou não saiba fazer a detecção de algum recurso, **MODERNIZR** pode ser uma opção! No exemplo acima, quando é detectado que o browser não oferece suporte ao recurso "notification" é carregado um script (polyfill) para simular o comportamento desejado.

4. MANIPULAÇÃO DE DOM

```
<p id=”printMessage”>Imprima essa página</p>


(function(){
    if(document.getElementById){
        var printMessage = document.getElementById('printMessage');
        if(printMessage && typeof window.print === 'function'){
            var btPrint = document.createElement('input');
            btPrint.setAttribute('type','button');
            btPrint.setAttribute('value','Imprimir');
            btPrint.onclick = function(){
                window.print();
            };
            printMessage.appendChild(btPrint);
        }
    }
})();
```

Existem situações em que uma funcionalidade só deve existir se o JavaScript estiver habilitado. Nesse caso não devemos exibir para o usuário um link ou botão que não funcione. O ideal é que essa funcionalidade seja inserida via JavaScript!

Um exemplo clássico é o botão de imprimir: sabemos que só com HTML não é possível chamar a função de imprimir, no exemplo acima fica claro uma forma de não frustar o usuário com um botão sem efeito - exibimos uma mensagem informando que o usuário deve imprimir a página em exibição, e caso o JavaScript esteja habilitado, fazemos a inserção do botão manipulando o DOM.

5. COMPORTAMENTO CONDICIONADO NO CSS

```
div {
    border-style: solid;
    border-width: 1px 1 px 4px 4px;
    box-shadow: -3px 3px 2px;
 }
```

Perceba que, no exemplo acima, a tolerância a falhas não nos ajuda. Se o browser não der suporte a box-shadow, a simulação de sombra funcionaria, porém, a criação de uma sombra de fato não seria aplicada. Em contrapartida, se o suporte existir, tanto a simulação quanto a sombra seriam aplicadas, causando um efeito no mínimo estranho =/ . Mais uma vez, é possível usar o Modernizr para contornar o problema:

```
.boxshadow div {
    box-shadow: -3px 3px 2px;
}
.no-boxshadow div, .no-js div {
    border-width: 1px 1 px 4px 4px;
}
```

O Modernizr adiciona na tag html de sua página uma classe para cada recurso que ele detecta que seu navegador tenha ou não. Caso o recurso não seja detectado, um "no-" é adicionado na frente da classe do recurso. No exemplo acima, foi disponibilizado a classe no-boxshadow.

Um problema no exemplo acima seria caso o JavaScript estivesse desabilitado. Para isso o Modernizr possui um mecanismo de fallback que funciona da seguinte maneira: o programador deve inserir na tag html a classe "no-js" e, quando o Modernizr for executado, ele se encarregará de remover a mesma.

Infelizmente nem tudo são flores: caso o navegador dê suporte ao recurso em questão, mas o JavaScript esteja desabilitado no navegador, a classe do recurso não seria disponibilizada e o recurso de CSS em questão não seria aplicado.

E COMO EU POSSO TESTAR O 'MEU' PROGRESSIVE ENHANCEMENT?

Testar sua aplicação é simples! Utilizando os navegadores você pode desabilitar CSS e JavaScript para ver o resultado.

Também é possível utilizar leitores de tela, já que isso permite que você "veja" sua página como alguém com deficiência visual a veria. Segue **LISTA DE LEITORES DE TELA.**

Você também pode checar o resultado em navegadores de texto, como o **LYNX**.

CONCLUSÃO

Podemos ver que o uso de "progressive enhancement" não é um trabalho fácil e demanda um certo esforço para que suas regras não sejam quebradas. No entanto, é importante desmistificar que utilizá-lo o obriga a criar páginas "pobres" visualmente ou com poucos recursos de tecnologia. Todos os recursos de navegação, animação e afins podem ser aplicados, desde que sejam utilizados como aprimoramentos, de forma a gerar naturalmente, fallbacks para navegadores mais simplistas.

Fonte: https://www.miati.com.br/blog/o-que-e-progressive-enhancement