

Centro de Tecnología y Artes Visuales

Diseño y Desarrollo Web

Tema:

**Graceful Degradation
and
Progressive Enhancement**

Integrantes:

Gloriana Angulo Alvarado.

Mariam Mora Robles

Alisson Zamora Sánchez

Tercer Avance - Julio, 2018.

Introducción

Se explica el significado, el propósito, el funcionamiento, la aplicación o implementación de progressive enhancement y graceful degradation a la hora de crear nuestro sitio web.

También se quiere asegurar que el contenido, los mensajes, y la funcionalidad de una página web llegue a cada persona o a cada uno de nuestros usuarios y no solo a algunas personas con browsers más modernos que soportan las últimas tecnologías web.

Objetivo General

Investigar sobre los conceptos, ejemplos de código, como usarlo o implementarlo, tanto de Progressive Enhancement como de Graceful Degradation para poder utilizarlos de la mejor manera en la creación de páginas web.

Objetivos Específicos

- Definir cada uno de los temas, explicar la diferencia que existe entre Progressive enhancement y Graceful Degradation.
- Definir las ventajas y desventajas de cada uno de los temas, explicar cómo implementar todas las buenas prácticas y recomendaciones.
- Explicar el porqué es importante implementar lo que estos temas nos ofrece en nuestras páginas web.

Conceptos Básicos :

- **Progressive enhancement o Mejora progresiva (PE):** Consiste en comenzar ofreciendo una web que funcione en cualquier dispositivo y navegador, e ir agregando características según aumenten las capacidades del navegador, la pantalla del dispositivo, la velocidad de la red, etc.
- **Graceful Degradation o Degradación gradual (GD):** Un sitio web diseñado y pensado gradualmente para que se vea primero y de manera correcta en los navegadores modernos y en los navegadores más antiguos, y menos ricos en características, de esta manera este tipo de páginas se degradaran en su forma de funcionar, estética, etc... La página funcionará pero con menos características.
- **User-agent:** Es un software que actúa en nombre de un usuario . Un uso común del término se refiere a un navegador web que le dice a un sitio web información sobre el navegador y el sistema operativo. Esto permite que el sitio web personalice el contenido para las capacidades de un dispositivo en particular.
- **Polyfill:** Un polyfill es una pieza de código (generalmente JavaScript en la web) que se utiliza para proporcionar funcionalidad moderna en navegadores antiguos que no lo admiten de forma nativa.
- **X-ray perspective:** Es una metodología usada para evaluar un diseño de sitio complejo, dividiéndolo en sus partes modulares más básicas, y reconstruyendolo de tal manera que una página codificada individualmente funcione en browsers modernos con capacidades funcionales completas tanto como otros browsers y dispositivos que pueden entender sólo HTML básico y nada más.

- **Cortar la mostaza:** Es un término acuñado por el equipo de la BBC para agrupar navegadores en, "modernos" navegadores y "viejos" navegadores.
- **Proxys:** Ordenador intermedio, que no permite la conexión directa entre internet y yo.
- **AJAX (*Asynchronous JavaScript And XML*):** es una técnica de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.
- **JSON (*JavaScript Object Notation*):** es un formato liviano de intercambio de datos.
- **Foundation markup:** Es la planeación del HTML que va a servir como la base para las experiencias básicas y mejoradas.
- **Semantic markup:** Tim Berners-Lee definió el markup semántico como una manera de “brindar estructura al contenido significativo de las páginas web”.
- **Pixel perfect:** Concepto utilizado mayoritariamente por diseñadores y maquetadores digitales, que se basa en: diseñar logrando la perfección, para que cada parte de la página o aplicación trabajen conjuntamente en ofrecer la mejor y la misma experiencia para los usuarios en cualquier tipo de navegador y dispositivo.
- **Modernizr:** es una librería JavaScript que nos permite conocer la compatibilidad del navegador con tecnologías HTML5 y CSS3, lo que nos permitirá desarrollar sitios web que se adapten a las capacidades cada navegador.

Investigación

La industria del diseño web siempre está en constante cambio, en parte porque los navegadores web y los dispositivos cambian constantemente. Dado que el trabajo que hacemos como diseñadores y desarrolladores web se ve a través de un navegador web de algún tipo, nuestro trabajo siempre tendrá una relación simbiótica con el software.

Uno de los desafíos que los diseñadores y desarrolladores de sitios web a sido tener que enfrentar no solo los cambios de los navegadores web, sino también la variedad de navegadores web que se utilizarán para acceder a los sitios web. Sería genial si todos los visitantes de un sitio estuvieran usando el último y mejor software, pero ese nunca ha sido el caso (y probablemente nunca lo será). Algunos de los visitantes de sitios verán las páginas web con navegadores que son muy antiguos y con las características que faltan de los navegadores más modernos. Por ejemplo, versiones anteriores del navegador Internet Explorer de Microsoft durante mucho tiempo han sido un dolor de cabeza para muchos profesionales de la web.

La realidad es que las personas que usan estos navegadores web anticuados a menudo ni siquiera saben que tienen un software obsoleto o que su experiencia de navegación web puede verse comprometida debido a su elección de software.

Para ellos, el navegador desactualizado que utilizan, es simplemente lo que han usado durante mucho tiempo para acceder a los sitios web. Desde la perspectiva de los desarrolladores web, queremos asegurarnos de que aún podamos ofrecer una experiencia útil a estos clientes, al tiempo que creamos sitios web que funcionan maravillosamente tanto en los navegadores y dispositivos más modernos, como en los más desactualizados y viejos; y con muchas funciones disponibles en la actualidad.

Graceful Degradation

La “**Degradación Elegante**” es una estrategia y un principio importante en el manejo del diseño de páginas web. Esto significa que cuando se implementan funciones diseñadas para aprovechar las mejores y más recientes características de los navegadores más actuales, hay que hacerlo de manera que los navegadores antiguos, además de aquéllos que permiten a los usuarios desactivar funciones particulares, puedan “retroceder” a un método que no impida el acceso al contenido básico del sitio, aunque quizás sin una apariencia tan elegante. Es un sistema tolerante a fallos, pero no significa que le está diciendo al usuario que use los últimos dispositivos o descargue un navegador actualizado. En el enfoque de degradación elegante, desarrolla o diseña un sitio web con las mejores funciones para los últimos navegadores o dispositivos, y agrega manejadores o funcionalidad para degradar su código según sea necesario para navegadores o dispositivos menos capaces.

Casi todas las características nuevas que se añaden a la web han sido diseñadas de una manera que permitan la degradación gradual. Por eso esta técnica crea páginas para los navegadores más modernos primero y luego los convierte para trabajar con navegadores menos funcionales.

La degradación agraciada es el proceso de decidir cómo un sitio grande debe ajustarse a pantallas más pequeñas o navegadores menos ricos en funciones.

El diseño web receptivo no siempre es único para todos. Al intentar apuntar a una audiencia en una red cada vez más diversa, la degradación elegante comienza a parecer que un tamaño no encaja lo suficiente.

La degradación agraciada tiene dos reglas principales:

- Su contenido debe ser visible y legible en todos los navegadores

- Los usuarios pueden navegar por el sitio web en todos los navegadores

Un sitio web diseñado para que degrade gradualmente está pensado para que se vea primero correctamente en los navegadores modernos. Para que se puedan acceder en los navegadores más antiguos, y menos ricos en características deben degradar, de forma a funcionar, pero con menos características.

Normalmente el sitio web está ya desarrollado con todas las funcionalidades de CSS3 y Javascript que necesitamos para hacer de nuestro sitio web una plataforma espectacular e innovadora.

La página empieza a degradar cuando, a través de detección user-agent, o utilizando la herramienta Modernizr como biblioteca de detección de características para HTML5/CSS3 le vamos sustituyendo o añadiendo soluciones al conflicto.

El “graceful degradation” me hace pensar en algunos proyectos en que el “Pixel perfect” es prioridad. El Pixel Perfect dice que todos los sitios webs tienen que tener exactamente la misma apariencia en todos los navegadores y todas las versiones. Normalmente, por diferentes cuestiones, entre ellas los deadlines, los desarrolladores se centran primero en los navegadores modernos y posteriormente hacen, o intentan hacer las modificaciones necesarias para que el sitio web se vea correctamente en los navegadores antiguos.

En una forma más fácil de decir es que crear una aplicación con una línea base de funcionalidad completa disponible en los navegadores modernos y luego quitar las capas para garantizar que funcione con navegadores más antiguos. Lo que hace es como lo dice su nombre, degrada la versión mejorada.

La degradación elegante consiste en la construcción de la página web, de tal forma que los usuario con los navegadores más modernos tengan un cierto nivel de experiencia, pero también permite su uso a los usuarios con navegadores antiguos ofreciendo un nivel más bajo en su experiencia de usuario. Este nivel más bajo no es tan agradable de usar para los visitantes del sitio, pero aun así la página ofrece la funcionalidad mínima y suficiente para utilizar el sitio web. La estructura no se rompe, las cosas están accesibles y además también funcionan para ellos.

El propósito de la Graceful Degradation es evitar fallas catastróficas. Lo ideal es que incluso la pérdida simultánea de múltiples componentes no provoque un tiempo de inactividad en un sistema con esta característica. En una Graceful Degradation, la eficiencia o velocidad de operación disminuye gradualmente a medida que un número creciente de componentes falla.

En el diseño del sitio web, el término Graceful Degradation se refiere a la implementación de características nuevas o sofisticadas para garantizar que la mayoría de los usuarios de Internet puedan interactuar eficazmente con las páginas del sitio. Importantes hitos en el diseño del sitio web y el uso de Internet a lo largo de los años han incluido la introducción de imágenes, marcos, juegos en línea, Java, JavaScript, controles ActiveX, navegación con pestañas, Voz sobre Protocolo de Internet (VoIP) y tecnología de videoconferencia. Cuando se lanza una versión actualizada de un navegador o sistema operativo, a menudo se incluyen nuevas funciones para mantener el ritmo de las últimas mejoras en Internet. Por diversas razones, muchos usuarios de Internet prefieren continuar usando sus navegadores existentes en lugar de actualizarse inmediatamente a la última versión cada vez que se vuelve popular una nueva tecnología de sitio web. Cuando se diseña un sitio con una Graceful Degradation, estos usuarios no se ven obligados repentinamente a actualizar sus navegadores a menos que utilicen los "antiguos"!

Graceful Degradation



Ejemplos de Graceful Degradation

HTML

Ex. 1: la etiqueta "noscript" es un mejor ejemplo de degradación elegante si ha proporcionado funciones de JavaScript en su sitio web y su navegador no admite JavaScript, o JavaScript está deshabilitado en el lado del cliente. Aparecerá el mensaje dentro de la etiqueta "noscript".

Por ejemplo, tiene una función JavaScript en un archivo JS que está incluido en la página desde el exterior, ha escrito código para menús desplegables, pero en el caso de que JavaScript no admita el navegador o esté deshabilitado, el código dentro del "noscript" "La etiqueta se representará y alcanzará la funcionalidad básica.

```
<script type = "text / javascript" src = "top-navigation.js"> </ script>
<noscript>
<ul id = "topNavigation" class = "topNavigation">
<li> <a href = "index.html "> Inicio </a> </ li>
<li> <a href="aboutus.html"> Sobre nosotros </a> </ li>
<li> <a href="products.html"> Productos </ a>
<ul class = "subMenus">
<li >> <a href="product1.html"> Producto 1 </a> </ li>
<li> <a href="product2.html"> Producto 2 < / a> </ li>
<li> <a href="product3.html"> Producto 3 </a> </ li>
</ ul>
</ li>
<li><a href="contactus.html"> Contáctenos </a> </ li>
</ ul>
</ noscript>
```

Ex. 2: el mejor ejemplo de mejora progresiva es el **diseño receptivo**, donde configura el HTML según los dispositivos y su navegador, que utiliza la consulta de medios. La técnica de diseño receptivo se basa en la estructura semántica de HTML, consulta de medios de CSS y dispositivos visibles. El código está escrito de una manera compatible tanto con los navegadores web como con los dispositivos móviles. Aquí, el diseño del sitio web se representa en función del ancho de la pantalla. En una consulta de medios de CSS, otorga las condiciones para ajustar el diseño o los elementos cosméticos en la página web.

```
@media solo pantalla y (max-device-width: 480px) {

    cuerpo {
color: # 000;
}

}
```

El código o las clases de CSS anteriores se aplicarán solo a aquellos dispositivos o navegadores que tengan un ancho máximo de pantalla de 480 px, y el siguiente CSS solo funcionará para aquellos que tengan un ancho mínimo de 481 px y un máximo de 1024 px.

```
@media solo pantalla y (min-width: 481px) y (max-width: 1024px) {

    cuerpo {
color: #ccc;
}

}
```

Está definiendo las condiciones para que sus navegadores o dispositivos puedan responder al mismo elemento con diferentes comportamientos.

Progressive Enhancement

Se centra en proporcionar una experiencia central "independiente del dispositivo" para todos los usuarios. Al comenzar con una base sólida que funciona en todas partes, HTML, nos aseguramos de que nuestra aplicación sea utilizable para la mayor audiencia posible. Esto nos anima a reducir nuestras aplicaciones a sus características *esenciales*, ya sea leyendo las noticias o enviando y recibiendo mensajes. Una vez que estamos seguros de que esta experiencia central es funcional para todos, podemos comenzar a aplicar el estilo y la interacción para proporcionar una mejor experiencia a los dispositivos que la admiten.

Algunas de las funciones más impresionantes de la web imitan las aplicaciones nativas, como el chat en tiempo real y los editores 'WYSIWYG'. Desafortunadamente, a menudo estos archivos pueden ser muy grandes y complejos, generalmente JavaScript, que resultan en sitios más lentos y descargas más grandes.

Tomar un enfoque amigable para PE significa asegurar que las funciones centrales detrás de estas acciones (por ejemplo, escribir el contenido, enviar un formulario) no dependan de archivos externos o JavaScript. Luego, en segundo plano, podemos 'progresivamente' cargar las versiones mejoradas de estas características y cambiarlas cuando estén listas. Mejor aún, podemos cargar estos activos sólo cuando sea relevante y no (por ejemplo) si no hay un editor WYSIWYG en la página.

Esto significa que los usuarios no necesitan esperar a la funcionalidad principal dentro de su aplicación. Si están en un dispositivo lento o en una conexión de red, aún pueden usar una versión simple de una función sin esperar.

Muchos navegadores antiguos tienen poca compatibilidad con características modernas, a menudo requieren la adición de grandes bibliotecas auxiliares (conocidas como 'polyfills') para llenar los vacíos. Esto generalmente significa servir

a los visitantes archivos más grandes de lo que necesitan, sólo para atender el mínimo común denominador.

Consiste en comenzar ofreciendo una web que funcione en cualquier dispositivo y navegador, e ir agregando características según aumenten las capacidades del navegador, la pantalla del dispositivo, la velocidad de la red, etc.

A veces nos olvidamos de que no todo el mundo accede en las mismas condiciones a Internet. Hay gente que accede desde dispositivos viejos, desde conexiones lentas, mediante proxys que bloquean ciertos contenidos... Por eso, es recomendable hacer que las webs sean accesibles desde cualquier dispositivo, de forma que, independientemente del navegador del usuario o de su velocidad de Internet, pueda acceder al contenido o usar las funciones básicas de la aplicación. Según los medios de acceso del usuario mejoren, se mejorará la experiencia agregando nuevas características a la web.

Pongamos un ejemplo para que quede más claro. Supongamos que tenemos un blog en el que usamos *flexbox* y animaciones. Un usuario que accede desde IE6 no va a ver las animaciones, y posiblemente vea algún elemento descolocado. Sin embargo, podrá acceder al contenido del blog. Si ese usuario actualiza su navegador, entonces verá los elementos posicionados de la forma correcta y funcionarán las animaciones. Desde ambos navegadores se podrá ver el contenido, solo que en navegadores más modernos se podrá ofrecer una mejor experiencia. Una web no tiene por qué verse igual en todos los navegadores.

No hay por qué esperar a que ciertas características estén soportadas por todos los navegadores para poder usarlas. El usar `position: sticky`; solo es soportado por Firefox y Safari. Sin embargo, si en estos dos navegadores funciona, ¿por qué no seguir usandolo? La gente que acceda desde otros navegadores simplemente no verán el efecto, pero podrán seguir usando la web sin problemas. Con el tiempo, los demás navegadores soportarán esta característica.

El *responsive design* es un claro ejemplo de que una web no tiene porque verse igual en cualquier navegador. Técnicas como *mobile first* obligan a pensar cuáles son las características mínimas que debe tener una web en pantallas

pequeñas. En pantallas más grandes se pueden mostrar los elementos de distinta forma, o agregar nuevos. En cualquier caso, la versión móvil debería poder ofrecer las características mínimas.

Por supuesto, hay muchos otros beneficios de una aplicación progresivamente mejorada. Su aplicación será mucho más tolerante a errores: si algo sale mal con una función nueva y elegante, existe una buena posibilidad de que los usuarios puedan completar su tarea con la versión 'básica'.

La mejora progresiva también ayuda a ahorrar tiempo cuando se trata de pruebas en varios dispositivos, ya que puede estar seguro sabiendo que su aplicación solo incluirá capas en las características cuando son compatibles. Prácticamente todo el mundo tiene garantizada una experiencia funcional, independientemente del navegador o dispositivo.

Aunque todos aparentan un mejor rendimiento y resistencia, la mejora progresiva sigue siendo un tema polémico. Algunos desarrolladores creen que el enfoque limita su capacidad para proporcionar experiencias dinámicas, especialmente aquellas que imitan las aplicaciones nativas. Muchas herramientas y marcos modernos, como React y Angular, están escritos completamente en JavaScript. Si bien proporcionan una experiencia rápida, similar a la de una aplicación, tampoco funcionarán (generalmente) para clientes que tienen JavaScript desactivado, o si hay un problema al analizar el script. Otros consideran que el futuro de la mejora progresiva radica en la resiliencia de la red, asegurando que las páginas funcionen 'fuera de línea' gracias a la nueva tecnología, como los trabajadores de servicios.

Al igual que todas las técnicas y metodologías, la mejora progresiva no necesita ser seguida dogmáticamente y es más una guía que una regla. Sin embargo, es una mentalidad valiosa que puede ayudar a mejorar su aplicación para todos los usuarios, no solo para aquellos en dispositivos más lentos o menos capaces. Ambos persiguen el mismo objetivo que es hacer que las páginas web sean accesibles para una variedad de navegadores y sus versiones.

La mejora progresiva es un enfoque del desarrollo web que tiene como objetivo ofrecer la mejor experiencia posible al público más amplio posible, y simplifica también la codificación y las pruebas. Ya sea que los usuarios vean tus sitios en un iPhone, el último y mejor sistema de alta gama, o incluso los escuchen en un lector de pantalla, su experiencia debe ser fácil de entender y usar, y lo más completa y funcional posible.

La mejor opción para un diseñador es elegir el enfoque de mejora progresiva porque es lógico. Cuando comienzas con una experiencia de visualización básica y creas tu sitio web progresivamente para agregar más funciones, no solo estás brindando una base sólida, sino que también estás permitiendo que las nuevas tecnologías avancen a medida que surgen. Esto también significa un menor mantenimiento y puede ahorrar en estos costos. Tendrá una nota positiva con los espectadores y podrás estar en una mejor posición para atraer más tráfico a tus páginas. La degradación elegante comienza en una complejidad final y trata de arreglar las cosas para las experiencias de los usuarios más bajas frente a la mejora progresiva que comienza con un mínimo que funciona de forma muy básica, pero que permite extenderla de forma constante en futuros entornos.

Progressive Enhancement



Cutting the mustard

Es una técnica que la BBC acuno para ejecutar una prueba simple cuando carga nuestro sitio web que verifica la existencia de características 'modernas' y luego carga el archivo apropiado en función del resultado. Los navegadores modernos obtienen el código moderno, los navegadores antiguos obtienen el código más grande compatible con versiones anteriores, y los navegadores *muy* antiguos no obtienen ninguna de las mejoras.

¿La única línea de JavaScript que decide si el navegador es HTML 4 o 5?

```
if('querySelector' in document
    && 'localStorage' in window
    && 'addEventListener' in window) {
    // Modern browser. Let's add JavaScript functionality
}
```

Al verificar desde el principio, podemos asegurarnos de que los usuarios solo descarguen y ejecuten el código apropiado. Aún mejor, dado que la funcionalidad básica se puede utilizar *sin* las mejoras, los usuarios pueden seguir interactuando con la página mientras esto sucede en segundo plano.

Tipos de navegadores y cómo se catalogan.

Navegadores HTML4: -

- IE8-
- Blackberry OS5-
- Nokia S60 v6-
- Nokia S40 (todas las versiones)
- Todas las demás variantes de Symbian
- Teléfono con Windows 7 (pre-Mango)
- ... y muchos más que son demasiado numerosos para mencionar

Navegadores HTML5: -

- IE9 +
- Firefox 3.5+
- Opera 9+ (y probablemente más atrás)
- Safari 4+
- Chrome 1+ (creo)
- iPhone y iPad iOS1 +
- Teléfono Android y tabletas 2.1+
- Blackberry OS6 +
- Windows 7.5+ (nueva versión de Mango)
- Mobile Firefox (todas las versiones probadas)
- Opera Mobile (todas las versiones probadas)

Entonces, ¿puedo o debo usar también este control de CTM? A menudo, bueno, probablemente la mayor parte del tiempo, ¡depende! Depende de su audiencia, depende de la funcionalidad utilizada en su sitio web o aplicación, depende de muchas cosas. Sin embargo, usar Cut the Mustard es un ejemplo perfecto de mejora progresiva. No es necesario que los sitios web se vean exactamente iguales en todos los navegadores y los navegadores antiguos no necesitan toda la funcionalidad que ofrecen los navegadores modernos, siempre que el mismo contenido esté disponible para todos. No sature los navegadores antiguos con toneladas de polyfills para obtener funciones modernas de JavaScript que funcionen allí y haga que su vida como desarrollador web sea innecesaria al tratar de aprender a volar un viejo automóvil.

Ventajas y Desventajas

Progressive enhancement

Ventajas:

Debe tener en cuenta que la implementación pragmática de la separación del marcado, el estilo y las capas de comportamiento le brindará muchos beneficios en los proyectos de su sitio. Estas son las principales razones para abrazar la mejora progresiva:

- **Accesibilidad:** el contenido está al alcance de todos los visitantes.
- **Portabilidad:** Soporte entre navegadores y dispositivos cruzados.
- **Modularidad:** Tener componentes desacoplados con límites inteligentes hace que las construcciones de sitios sean más fáciles y más tolerantes a fallas.
- **Rendimiento del sitio:** las mejoras en términos de tiempos de carga de páginas (percibidas que afectan la usabilidad) se implementan más fácilmente.
- Puede adivinar con una línea de código en cuál de los navegadores se está abriendo el sitio.
- Comienza con un nivel básico de características que funcionan en todos los navegadores.
- Es bueno para el usuario.
- Es bueno para el desarrollador.
- Su código es más fácil de mantener y depurar.

Desventajas:

- Mayor costo y tiempo de desarrollo

Graceful Degradation

Ventajas:

- Brinda una caja de herramientas clara.
- **Utiliza clases de visibilidad.** Organizar las características y los elementos en "debe tener" y "agradable de tener".
- **Separa los sitios móviles.** Redirigir a los usuarios móviles a un segundo sitio. Mantener dos sitios puede duplicar el trabajo, pero resuelve el problema inmediato.
- **Caída a 640px de ancho.** La mayoría de los dispositivos de mano pueden mostrar razonablemente bien sitios web de 640 píxeles de ancho.
- Permitir el acceso al contenido básico del sitio mientras se garantiza una experiencia de usuario óptima en dispositivos menos capaces.
- Asegura que todos los usuarios puedan ver su sitio web, independientemente del navegador que usen, cuantos más dispositivos y tamaños de pantalla su sitio web funcione, mayor será el alcance que tendrá el usuario.

Desventajas:

- Limitar nuestros diseños a una sola audiencia, sin importar cuán extendidos estén, sigue siendo limitante.
- Los sitios atienden a navegadores antiguos en donde su diseño puede pasar de ser aceptable a ridículo.
- Las personas tienen que estar actualizando sus navegadores.

Progressive enhancement vs Graceful degradation

PE difiere de Graceful Degradation (GD) en que GD es el viaje de la complejidad a la simplicidad, mientras que PE es el camino de la simplicidad a la complejidad. La PE se considera una mejor metodología que la GD porque tiende a cubrir una mayor gama de problemas potenciales como línea de base. PE es la lista blanca de la lista negra de GD. Parte del atractivo de PE es la fuerza del resultado final . PE lo obliga a planificar inicialmente su proyecto como un sistema funcional utilizando solo las tecnologías web más básicas. Esto significa que usted sabe que siempre tendrá una base sólida a la que recurrir a medida que se introduce la complejidad en el proyecto.

Graceful degradation y progressive enhancement son una técnica útil que permite a los desarrolladores web centrarse en el desarrollo de los mejores sitios web posibles, al tiempo que equilibra los problemas en los sitios web a los que acceden múltiples agentes de usuarios desconocidos. Graceful degradation / Progressive enhancement está relacionada, pero es diferente, a menudo se considera que va en la dirección opuesta a la progressive enhancement. En realidad, ambos enfoques son válidos y con frecuencia se complementan entre sí.

La degradación elegante significa pensar atrás y ser rehén del pasado, mientras que la mejora progresiva significa mirar al futuro siendo consciente del estado de la cuestión en el presente.

Últimamente ha surgido una discusión en los sitios de desarrollo web sobre la diferencia en los conceptos de “degradación gradual” y “mejoramiento progresivo”, que en realidad son conceptos bastante similares, pero observados desde diferentes perspectivas. La diferencia básica es que en el “mejoramiento progresivo”

se comienza con un conjunto simple, lógico y compatible de contenido marcado y después, encima, se colocan, en capas, las funciones mejoradas para los navegadores modernos, mientras que la “degradación gradual” comienza con un sitio avanzado, con todas sus características, lleno de accesorios, y uno se asegura de que también contenga contenido accesible si las características elegantes no funcionan para un usuario en particular. De acuerdo a esto, la actitud que yo promuevo encaja más en mejoramiento progresivo que en degradación gradual, aunque yo usé la terminología de degradación gradual porque el otro término no se había inventado en ese momento. En cualquier caso, si se practican eficaz, hábil y cuidadosamente, ambas técnicas deben dar lugar a sitios muy similares.

La degradación elegante a diferencia de la mejora progresiva, que se basa en un mensaje enfocado al que cualquier persona puede acceder, la degradación elegante se dirige a una experiencia de visualización específica, pero tiene en cuenta los dispositivos menos capaces.

La cuestión de cuándo usar Graceful Degradation o Progressive Enhancement depende en gran medida de la funcionalidad que ofrece el sitio web. En general, Progressive Enhancement brinda una experiencia de usuario básica a todos los navegadores y, a medida que avanza la versión, también lo hace la experiencia. Por otro lado, Graceful Degradation es el proceso de creación de sitios web para tecnologías posteriores al tiempo que permite a navegadores antiguos mostrar algunas funciones básicas. En este sentido, si su sitio web tiene contenido estático sin gráficos ni videos, entonces Graceful Degradation funcionará porque no hay mucha diferencia entre las tecnologías. Lo contrario es cierto para los sitios web de uso intensivo de la aplicación. Si sus páginas web tienen mucha funcionalidad con la transmisión de videos, imágenes, chat en vivo, etc. entonces es importante que utilice la mejora progresiva porque todos los usuarios tendrán una experiencia de visualización básica independientemente de sus navegadores. Aunque es posible que no puedan acceder al chat y a otras funciones avanzadas, aún pueden solucionarlo y, en tal caso, esta es su mejor opción.

Progressive enhancement es lo opuesto a Graceful degradation. En lugar de desarrollar todas las características desde el principio, una página web se construye a partir de una línea de base de las características admitidas por todos los navegadores (y versiones de navegador). Luego, se agregan características más avanzadas como capas, por lo que la página web aprovecha la funcionalidad que ofrecen los navegadores más nuevos.

CSS

Mejora progresiva con CSS.

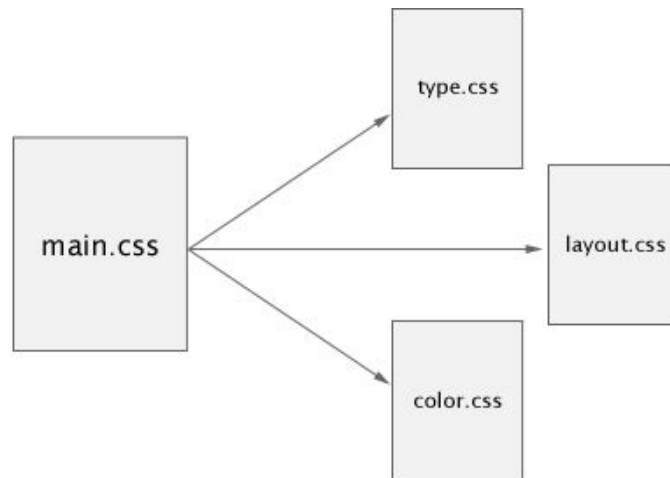
- **Organización de hojas de estilo**

Muchos diseñadores y desarrolladores web no piensan mucho acerca de cómo incorporan hojas de estilo en sus documentos, pero hay un verdadero arte en ello. Con los métodos correctos, puede obtener de inmediato muchos de los beneficios de la mejora progresiva.

- **Usa múltiples hojas de estilo**

Hay muchos beneficios para obtener una pequeña separación en sus estilos. Obviamente, las hojas de estilo que tienen más de 1500 líneas son un poco difíciles de mantener y separarlas en varias hojas de estilo puede mejorar su flujo de trabajo (y su cordura). Otro beneficio que no se suele considerar es que esta separación puede ayudarlo a obtener una mayor consistencia de presentación en los tipos de medios a los que se dirige.

Considere tomar su archivo `main.css` , que contiene todas las reglas de estilo para su sitio, y `main.css` en hojas de estilo individuales que contengan información tipográfica, de diseño y de color. Nombra los archivos según corresponda: `type.css` , `layout.css` , `color.css` .



Una vez que haya hecho eso, puede usar un poco de magia para proporcionar automáticamente una experiencia de "baja fidelidad" a navegadores más antiguos (como Internet Explorer 5 / Mac) y muchos otros navegadores que carecen de soporte sólido para diseños de CSS . ¿Cómo? Bueno, todo depende de cómo se vincule a los archivos. Supongamos que comenzamos con un archivo `main.css` incluido a través de un elemento de link :

```
<link rel = "stylesheet" type = "text / css" href = "main.css" />
```

Primero, dividiremos en llamadas separadas a nuestras tres hojas de estilo contextuales:

```
<link rel = "stylesheet" type = "text / css" href = "type.css" />
```

```
<link rel = "stylesheet" type = "text / css" href = "layout.css" />
```

```
<link rel = "stylesheet" type = "text / css" href = "color.css" />
```

- **Trabajando con tipos de medidas alternativos.**

Dado que la entrega de contenido es el enfoque principal de la mejora progresiva y queremos brindar una experiencia "mejorada" a cualquier dispositivo que la respalde, realmente deberíamos comenzar a pensar más allá del navegador; lo más importante es que deberíamos pensar en la impresión y los dispositivos móviles.

Ahora, normalmente, agregaremos estilos de impresión con otro elemento de link :

```
<link rel = "stylesheet" type = "text / css" media = "print" href = "print.css" />
```

Tradicionalmente, esta hoja de estilo contendría todas nuestras reglas relacionadas con la impresión, incluidas las reglas tipográficas y de color. En el caso de la tipografía en particular, las reglas en nuestra hoja de estilo de impresión probablemente reflejen las de nuestra hoja de estilos principal. En otras palabras, tenemos mucha duplicación.

Aquí es donde la separación de los estilos tipográficos y de color de los estilos de diseño se convierte en un gran activo: ya no necesitamos esas reglas en nuestra hoja de estilo de impresión. Además de eso, podemos utilizar otra pequeña técnica de organización para mejorar la escalabilidad de nuestro sitio y ocultar ciertos estilos de diseño de los navegadores problemáticos.

Comencemos revisando nuestras hojas de estilo. Considera lo siguiente:

```
<link rel = "stylesheet" type = "text / css" href = "type.css" />
```

```
<link rel = "stylesheet" type = "text / css" href = "layout.css" />
```

```
<link rel = "stylesheet" type = "text / css" href = "color.css" />
```

Ahora, dado que no tenemos un tipo de medio declarado, Netscape 4.x leerá cualquier estilo en estos tres archivos, pero podemos usar su comprensión básica de CSS y proporcionar una organización adicional de nuestras reglas de estilo moviendo todos los estilos. los estilos que layout.css contenía en una nueva hoja de

estilo, apropiadamente llamada screen.css. Finalmente, actualizamos los contenidos de layout.css para importar screen.css y NS4 .x y su tipo no será más inteligente (ya que no entienden la directiva @import).

```
@import 'screen.css';
```

Sin embargo, todavía hay un poco de margen de mejora, realmente deberíamos declarar a qué medio pertenece esta hoja de estilo, y lo haremos añadiendo un tipo de medio a la declaración @import :

```
@import 'screen.css' pantalla;
```

El problema es que IE7 y los siguientes no entenderán esta sintaxis e ignorarán la hoja de estilo, pero si también quieres proporcionar estos estilos a esos navegadores (lo cual probablemente harás), puedes hacerlo fácilmente usando los Comentarios condicionales, que cubriremos a continuación. Aquellos de ustedes con ojos de águila también pueden haber notado el uso de comillas simples (') en lugar de comillas dobles (") alrededor del nombre de la hoja de estilo. Este es un truco ingenioso para hacer que IE5 / Mac ignore una hoja de estilo. Y dado que el diseño CSS de IE5 / Mac es bastante incompleto (especialmente cuando se trata de flotadores y posicionamiento), ocultar las reglas de diseño es una forma perfectamente aceptable de tratarlo. Después de todo, todavía obtendrá la información de color y tipografía, lo que cuenta para algo. Utilizando la misma técnica, podemos importar nuestro archivo print.css (que contiene, lo adiviné, reglas específicas de diseño de impresión).

```
@import 'screen.css' pantalla;
```

```
@import 'print.css' imprimir;
```

Ahora no solo tenemos hojas de estilo muy bien organizadas, también tenemos un medio eficaz para mejorar progresivamente el diseño de nuestro sitio.

Ejemplos de CSS

Ex. 1: las nuevas propiedades de CSS3 son un buen ejemplo de mejora progresiva. Supongamos que tiene un área rectangular con un poco de color de borde y codifica el CSS de acuerdo con el navegador base (IE7 o IE8).

```
.whiteRectanglePalceHolder {  
border: 1px solid # 000;  
fondo: #fff;  
relleno: 0;  
margen: 0;  
}
```

Ha escrito la clase CSS anterior para el navegador IE7, que es su navegador base. Se representa en los navegadores IE7 + como un rectángulo y se agrega un atributo más (gris resaltado) dentro de esta clase, como en el ejemplo a continuación. El resultado en los navegadores IE7 e IE8 será el mismo que en un rectángulo. Pero, en el navegador IE9, el resultado aparece como un cuadro rectangular con esquinas redondeadas.

```
.whiteRectanglePalceHolder {  
border: 1px solid # 000;  
fondo: #fff;  
relleno: 0;  
margen: 0;  
  
radio del borde: 10px; / ***** agregado de nueva propiedad ***** /  
}
```

Ex. 2: el mejor ejemplo de mejora progresiva es el **diseño receptivo**, donde configura el HTML según los dispositivos y su navegador, que utiliza la consulta de medios. La técnica de diseño receptivo se basa en la estructura semántica de HTML, consulta de medios de CSS y dispositivos visibles. El código está escrito de una manera compatible tanto con los navegadores web como con los dispositivos móviles. Aquí, el diseño del sitio web se representa en función del ancho de la pantalla. En una consulta de medios de CSS, otorga las condiciones para ajustar el diseño o los elementos cosméticos en la página web.

```
@media solo pantalla y (max-device-width: 480px) {  
  
    cuerpo {  
color: # 000;  
    }  
  
}
```

El código o las clases de CSS anteriores se aplicarán solo a aquellos dispositivos o navegadores que tengan un ancho máximo de pantalla de 480 px, y el siguiente CSS solo funcionará para aquellos que tengan un ancho mínimo de 481 px y un máximo de 1024 px.

```
@media solo pantalla y (min-width: 481px) y (max-width: 1024px) {  
  
    cuerpo {  
color: #ccc;  
    }  
  
}
```

Está definiendo las condiciones para que sus navegadores o dispositivos puedan responder al mismo elemento con diferentes comportamientos.

Degradación elegante con CSS

¹La degradación elegante no se trata de permitir que los sitios web se vean mal en los navegadores más antiguos, sino de hacer que se vean bien en los modernos, eso significa aprovechar las características útiles de CSS3.

Los elementos periféricos en un diseño web generalmente pueden permitirse degradar de forma natural, mientras que los elementos centrales no pueden. Decidir qué características de CSS3 son prescindibles es un proceso de tres pasos:

- Determine qué elementos son visualmente esenciales.
- Si el estilo va a disminuir, determine en qué navegadores la degradación es aceptable. Tome esta decisión en función del uso del sitio web en diferentes navegadores y versiones.
- Prestar especial atención a los elementos de UI.

¹ Jon Raasch, "Graceful degradation con CSS"
<http://jonraasch.com/blog/graceful-degradation-with-css3>

Buenas Prácticas

Implementar PE de forma ordenada con capas.

La primera capa es HTML semántico limpio. Esto permite que los agentes de usuario basados en texto, hablados, anticuados y robóticos naveguen por el contenido de su sitio web de manera adecuada.

```
<form action="save_order.php" method="post">
  <fieldset>
    <legend>Order of Navigation</legend>
    <ol>
      <li id="homepage-12">Homepage <label for="menu-id-12">Change the order for
Homepage</label><input type="text" name="homepage-12" id="menu-id-12" value="1" /></li>
      <li id="contact-23">Contact Us <label for="menu-id-23">Change the order for Contact
Us</label><input type="text" name="contact-23" id="menu-id-23" value="2" /></li>
      <li id="about-16">About Us <label for="menu-id-16">Change the order for About
Us</label><input type="text" name="about-16" id="menu-id-16" value="3" /></li>
      <li id="latest-14">Latest News <label for="menu-id-14">Change the order for Latest
News</label><input type="text" name="latest-14" id="menu-id-14" value="4" /></li>
    </ol>
  </fieldset>
  <p><input type="submit" value="Save new order" /></p>
```

Order of Navigation

1. Homepage Change the order for Homepage

1

2. Contact Us Change the order for Contact Us

2

3. About Us Change the order for About Us

3

4. Latest News Change the order for Latest News

4

Save new order

La segunda capa es CSS. Esto permite a los usuarios-agentes basados en la visualización mostrar o alterar la representación visual del contenido de su sitio web.

```
name="code">
<style type="text/css">
  form { width: 50%; margin: 0 auto; }
  fieldset { background: #555555; padding: 1em; }
  legend { border: 1px #513939 solid; background: #FAFAFA; }
  label { position: absolute; margin-left: -999em; }
  ol { list-style: none; position: relative; }
  body { font: 100% serif; }
  ol li { border: 1px #FFF solid; background: #FAFAFA; padding: 0.7em; }
  ol li:hover { border: 1px #513939 solid; }
  input[type='text'] { width: 2em; text-align: center; position: absolute; left: 40%; }
</style>
```

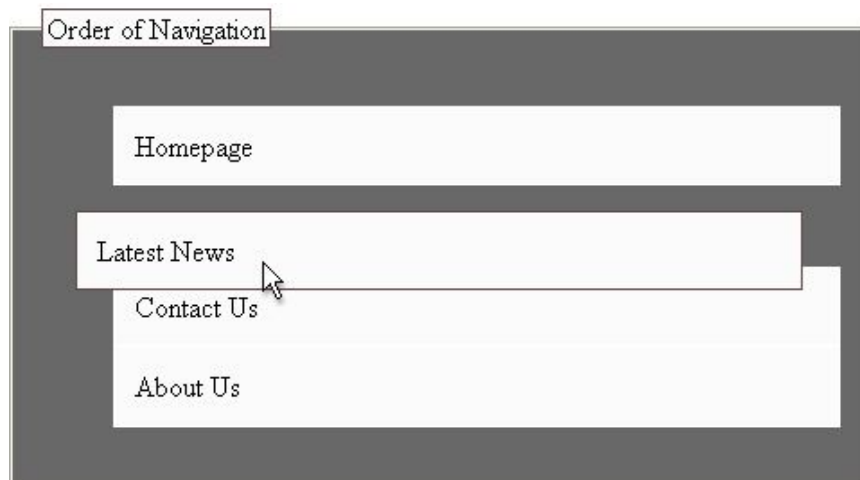
Order of Navigation

Homepage	1
Contact Us	2
About Us	3
Latest News	4

Save new order

La tercera capa es JavaScript. Esto permite a los usuarios-agentes que son capaces de usarlo proporcionar a sus usuarios una usabilidad mejorada.

```
<script type = "text / javascript">
  $ (document) .ready (function () {
    $ ('input'). hide ();
    $ ('ol'). ordenable ({elementos: 'li',
      actualización: función (evento, ui) {
        var new_order = $ ('ol'). sortable ('toArray');
        $ .each (new_order, function (i, element) {
          $ ('input [name =' + element + ']'). attr ('value', i + 1);
        });
        $ .post ("save_order.php", {
          'new_order': $ ('form'). serialize ()
        })
      }
    });
  });
</ script>
```



Java Script

¿Qué hacer si falla Javascript?


Es raro que hoy en día haya gente que tenga Javascript desactivado, pero podría haber motivos aparte por los cuales deje de funcionar (errores en el código, problemas de red...).

Pensemos en las *Single Page Apps* que se pueden crear con frameworks como Angular. Generalmente, estas aplicaciones cargan un *html* vacío con los *scripts* necesarios para ejecutarse en la aplicación. **Si por alguna razón no se ejecuta Javascript, lo único que se verá es una página en blanco.** Eso no es bueno.

Hay técnicas que nos ayudan a resolver este problema. Podemos generar el HTML de la página en el servidor, de forma que si Javascript falla, al menos el contenido de la página se podrá ver, aunque no se encuentre totalmente funcional. También tenemos que tener en cuenta otros elementos como los enlaces. Con Javascript funcionando, al pulsar en un enlace la aplicación enviará una petición AJAX al servidor, que devuelve un JSON con los datos a mostrar en la nueva página. Sin Javascript funcionando, al pulsar el enlace simplemente recarga la página y muestra la nueva página. Vemos, el funcionamiento de toda la vida de los enlaces.

¿Cómo habilitar o deshabilitar Javascript en Google Chrome?

Habilitar JavaScript en Google Chrome

1. Abra Chrome en el ordenador.
2. En la esquina superior derecha, haga clic en Más  > **Configuración**.
3. En la parte inferior, haga clic en **Configuración avanzada**.
4. En "Privacidad y seguridad", haga clic en **Configuración de contenido**.
5. Haga clic en **JavaScript**.
6. Active **Permitido (recomendado)**.

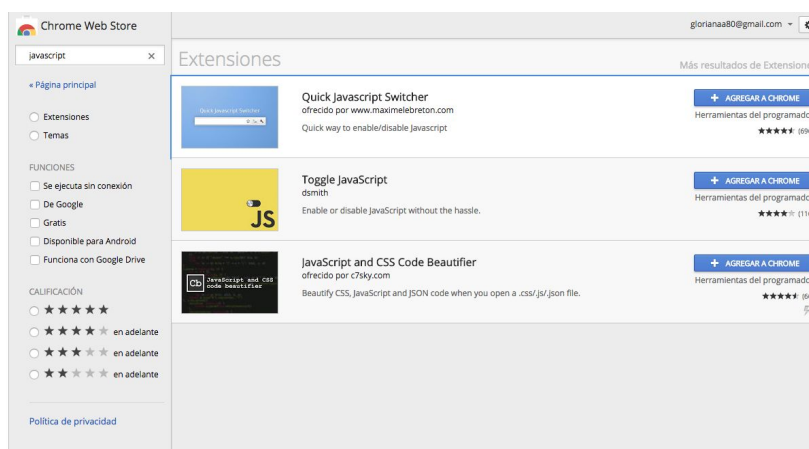
Pruebas

¿Como saber si una página web contiene Progressive enhancement en el navegador de Google Chrome?

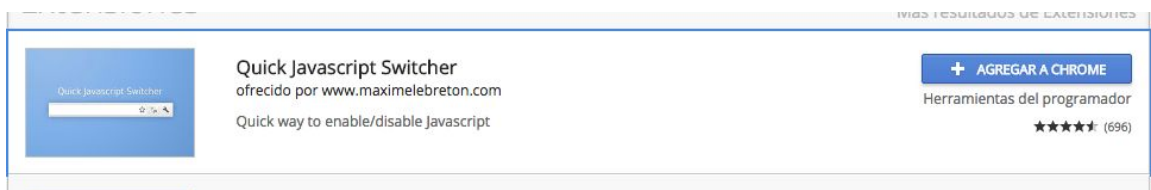
1. Buscar “Google Extensions”
2. Darle click.


Chrome Web Store - Extensiones - Google
<https://chrome.google.com/webstore/category/extensions?hl=es-419> ▾
Extensión de Google Keep para Chrome. (5802) ... Extensiones favoritas de Chrome. Ver todos
Extensiones que logran que te apropiés de Chrome.
[Temas](#) · [Juegos](#) · [Actualizaciones recientes](#) · [MEGA](#)
Visitaste esta página 2 veces. Última visita: 19/07/18

3. Buscar JAVASCRIPT.



4. Dar click en agregar en Chrome.



5. Ya instalada a la par de la estrella de favoritos  el siguiente símbolo.

Está

activo



Está desactivado



Pasos a seguir para la aplicación de la Mejora Progresiva

Crearemos un documento con el contenido necesario y marcado de acuerdo a nuestros objetivos. Asegurémonos de no incluir ningún tipo de estilo, así como de utilizar los elementos HTML apropiados a cada pieza de contenido. Verifiquemos que los párrafos vayan como elementos `< p >`, las listas, según su tipo, vayan como `< ul >`, `< ol >` o `< dl >`.

Asegurémonos también de no incluir elementos como `< br >`, `< b >` (utilicemos `< strong >`) o `< i >` (utilicemos `< em >`).

Cuando el documento esté correctamente marcado, aplicamos nuestra primera hoja de estilo ('minima.css') y la enlazamos vía `< link >`. Esta hoja utilizará unos estilos básicos que cualquier navegador preparado para CSS podrá aplicar sin problemas. Además, en esta hoja podremos especificar reglas específicas para NN 4.x, utilizando para ello el hack llamado Fabrice's Inversion o reglas que excluyen específicamente a NN 4.x, con el Caio's Hacks.

Ahora recurriremos al High Pass Filter creado por Tantek Celik. Este filtro nos asegura que nuestra hoja de estilos sólo será enlazada por aquellos navegadores con un soporte sólido de CSS1 (concretamente, por todos aquellos que pasen con éxito la sección 7.1 de la CSS1 Test Suite Para aplicarlo enlazaremos desde nuestro documento a (vía) con una hoja de estilo llamada 'filtro.css' que contendrá sólo las siguientes dos líneas:

```
@import "null?\"{";  
@import "máxima.css";
```

Para no extender más de lo necesario, baste con dar por sabido que los navegadores que superen este test, importaran la hoja de estilo 'máxima.css', donde aplicaremos el grueso de nuestra información de estilos, con la seguridad de saber que el resto de navegadores no se verán afectados por una serie de reglas que no aplicarán o aplicarán incorrectamente.

Ahora es el momento de perfeccionar nuestras hojas de estilo, sea con hojas para otros medios (por ejemplo, impresora), sea con la aplicación de otros conocidos hacks.

/* Página que se le va todo si no tiene JS activado*/

<https://zurb.com/word/graceful-degradation>

Progressive enhancement in action

Cuando construimos un sitio web es importante usar the x-ray perspective para asegurarnos de que todos los usuarios obtengan la mejor experiencia dentro de las capacidades y restricciones de sus browsers. Progressive enhancement hace esto posible mediante los beneficios que la codificación avanzada permite a los browser modernos, mientras sigue funcionando en los browser no tan modernos, todo esto con una única base de código.

X ray perspective es una metodología usada para evaluar un diseño de sitio complejo, dividiéndolo en sus partes modulares más básicas, y reconstruyéndolo de tal manera que una página codificada individualmente funcione en browsers modernos con capacidades funcionales completas tanto como en otros browsers y dispositivos que pueden entender sólo HTML básico y nada más.

El proceso de x ray perspective inicia con un diseño de destino que muestra cómo debería de verse y comportarse la página final en un browser moderno. Con el diseño de destino en mano, seguimos un proceso de planificación y desarrollo comprendido en tres partes:

1. Definir la jerarquía y prioridad del contenido general, y asignar componentes a un equivalente html básico.
2. Construir un “foundation markup” que proporcione todo el contenido esencial y funcional con estilos seguros mínimos y no con JavaScript.

3. Escribir un markup avanzado, CSS, y JavaScript para las mejoras visuales y funcionales de la capa, en browsers que puedan soportarlo.

Construir las experiencias básicas y mejoradas en código es siempre un proceso iterativo; mientras escribimos el primer borrador del foundation markup y comenzamos a trabajar por dónde agregar las mejoras, siempre encontramos casos donde ligeros retoques del foundation markup van a ser necesarios para hacer esas mejoras posibles.

La mejora progresiva es un enfoque del desarrollo web que tiene como objetivo ofrecer la mejor experiencia posible al público más amplio posible, y simplifica también la codificación y las pruebas. Ya sea que los usuarios vean tus sitios en un iPhone, el último y mejor sistema de alta gama, o incluso los escuchen en un lector de pantalla, su experiencia debe ser fácil de entender y usar, y lo más completa y funcional posible.

Diseñar con Mejora progresiva le mostrará cómo hacerlo. Es una guía práctica para comprender los principios y beneficios de la mejora progresiva, y una exploración detallada de ejemplos que le enseñará, ya sea diseñador o desarrollador, cómo, dónde y cuándo implementar los enfoques específicos de codificación y scripting. que encarna la mejora progresiva.

Construyendo el “foundation markup” y estilos seguros

Cuando la jerarquía general de una página y el propósito de cada componente es planeado, es mucho más fácil escribir la semántica en HTML que va a servir como la base para las experiencias básicas y mejoradas. Nos referimos a este HTML como el “foundation markup” porque literalmente forma parte de la fundación en la cual vamos a construir las mejoras de Css y JavaScript para dar el diseño final.

Cuando codificamos el foundation markup, primero abordamos las principales secciones de diseño; incluyendo todos los elementos principales como el header,

footer, y las configuraciones de columna, para solidificar la estructura de la página inicial antes de agregar los componentes más detallados. Luego agregamos el contenido y la funcionalidad de todos los elementos individuales en la página con un enfoque particular en el uso de la semántica HTML para crear una experiencia básica que sea fácil de usar.

Escribiendo un markup significativo

Estos días es tentador pasar por alto un apropiado markup y semántica: existe un arsenal de tecnologías web que pueden transformar completamente un HTML, y diseñadores y desarrolladores pueden aplicar en CSS y JavaScript una pila de **divs** y **spans** para imponer jerarquía, hacer que luzca bien, o hacer que se comporte como ellos quieran. Pero ¿qué pasa cuando desactivamos CSS y JavaScript? ¿la jerarquía del contenido va a seguir en su lugar? ¿la información más importante va a aparecer al inicio de la página? ¿dónde está la navegación? ¿es fácil de encontrar? ¿todo va a seguir funcionando?

Por esta razón es recomendado integrar progressive enhancement y pruebas de capacidades en nuestro proceso de desarrollo para asegurarnos de que todo el contenido y la funcionalidad alcance a cualquiera con cualquier dispositivo habilitado para web. Pero esto sólo se puede lograr cuando el sitio es construido en una base adecuada de limpieza, bien organizada, y un markup semántico estructurado.

El markup semántico se considera en cuatro secciones básicas:

- Marcando texto e imágenes: La construcción de los bloques básicos de cualquier sitio trabajan mejor cuando se marca lo más preciso posible. Esta sección explora el rango desde los tags semánticos hasta el contenido web inteligentemente identificado e imparte significado jerárquico.
- Marcando elementos interactivos: Formularios y otros elementos interactivos incorporan habilidades muy específicas, y elegir un markup adecuado es

crucial para soportar comportamientos esperados y brindar al usuario una interacción óptima.

- Creando contexto en la página: Cómo agrupar elementos individuales de contenido y objetos de formulario relacionados ofrecen oportunidades para ayudar al navegador y a la audiencia a entender el mensaje.
- Configurar un elemento de HTML: Finalmente, usando el markup que define el documento HTML e identifica la página ofreciendo sus propios métodos para codificar el significado e informar al browser o a otro usuario agente cómo interpretarlo.

¿Realmente necesito hacer esto en mi web?

Depende del proyecto que estés creando, puede ser algo útil o puede que no sea necesario. Por lo general es buena idea que las aplicaciones dirigidas al público general se desarrollen de esta forma, siempre que se pueda. Conseguir que la aplicación pueda funcionar en la mayor cantidad de dispositivos posible siempre es bueno.

Sin embargo, si trabajamos para un entorno empresarial que controlamos, en el que sabemos que todos los usuarios de la aplicación usan Chrome(por ejemplo), no es necesario complicarse la vida.

Pero ten en cuenta que es más sencillo tener una base que funcione y agregarle nuevas características según aparezcan mejoras en los navegadores, que tener que compatibilizar ciertos navegadores una vez la aplicación ya está creada.

El ***Progressive Enhancement*** dice que los sitios webs no tienen que tener la misma apariencia en todos los navegadores, pero aprovechar las capacidades del navegador para que el mayor número posible de usuarios tenga la mejor experiencia posible. Aquí lo más importante es el contenido y cómo este contenido es accesible, independientemente de los dispositivos utilizados por el usuario.

La Mejora progresiva sería lo inverso a la Degradación gradual; empezar por lo básico (HTML + CSS) e ir evolucionando “progresivamente”, utilizando las mismas herramientas de **user-agent** y detección de características.

En la Mejora progresiva el sitio web se desarrolla en ‘capas’, utilizando un abordaje de “de abajo a arriba” empezando “con una capa más simple y mejorando las capacidades del sitio en las capas sucesivas, cada una utilizando más características.” Los beneficios del Progressive Enhancement son inúmeros, entre otros, permite un desarrollo más rápido, es más económico de mantener y permite que sitios webs desarrollados con este abordaje puedan reaccionar y ser optimizados de acuerdo con el ambiente en que son visualizados, lo que hace que las páginas se carguen, normalmente, más rápido.

Además de Modernizr o WURFL.js que son bibliotecas de Javascript, se pueden utilizar otras herramientas de Javascript y PHP para la detección de *user-agents*. Por ejemplo MobileDetect(PHP).

Obviamente la necesidad del proyecto y/o la audiencia, dictarán el abordaje a utilizar, pero para algunos desarrolladores la Mejora progresiva, además de ser el mejor, es también una cuestión de “derechos humanos”. ¿Qué te parece?

Es una estrategia particular de diseño web que acentúa la accesibilidad, margen de beneficio semántico, y tecnologías externas del estilo y el scripting, en una manera adecuada que permite que cada uno tenga acceso al contenido y a la funcionalidad básica de una página web, usando cualquier navegador web o conexión a Internet, mientras que también permite a otros con un mayor ancho de banda o un navegador web más avanzado experimentar una versión mejorada de la página.

Progressive enhancement es una filosofía de diseño que se centra en proporcionar una línea de base de contenido y funcionalidad esencial para la mayor cantidad de usuarios posible, al tiempo que va más allá y ofrece la mejor experiencia posible solo a los usuarios de los navegadores más modernos que pueden ejecutar todo código requerido.

Bibliografía

/* Definición de GD*/

<https://searchnetworking.techtarget.com/definition/graceful-degradation>

/* Cómo implementar PE con ejemplos de código en forma de capas*/

<https://www.smashingmagazine.com/2009/04/progressive-enhancement-what-it-is-and-how-to-use-it/>

/* Definición de PE y GD y cual es la mejor */

<https://www.mavenecommerce.com/blog/progressive-enhancement-vs-graceful-degradation/>

/* Ejemplos de código de Graceful Degradation */

<https://www.7binaryoptions.com/translations/graceful-degradation-spanish/>

/* Cómo implementar lo de cortar la mostaza */

<https://justmarkup.com/log/2015/02/cut-the-mustard-revisited/>

<http://responsivenews.co.uk/post/18948466399/cutting-the-mustard>

/* Descripción de PE y GD */

<https://inusual.com/blog/degradacion-gradual-vs-mejora-progresiva>

/* Ejemplo de código */

<https://www.hcltech.com/blogs/engineering-and-rd-services/paradigms-graceful-degradation-and-progressive-enhancement>

/* Diferencia de Graceful degradation vs progressive enhancement *

https://www.w3.org/wiki/Graceful_degradation_versus_progressive_enhancement

/* link de un video que tengo que terminar de ver sobre progressive enhancement*/

<https://www.youtube.com/watch?v=r038QioMtxI>