Proyecto corto #2

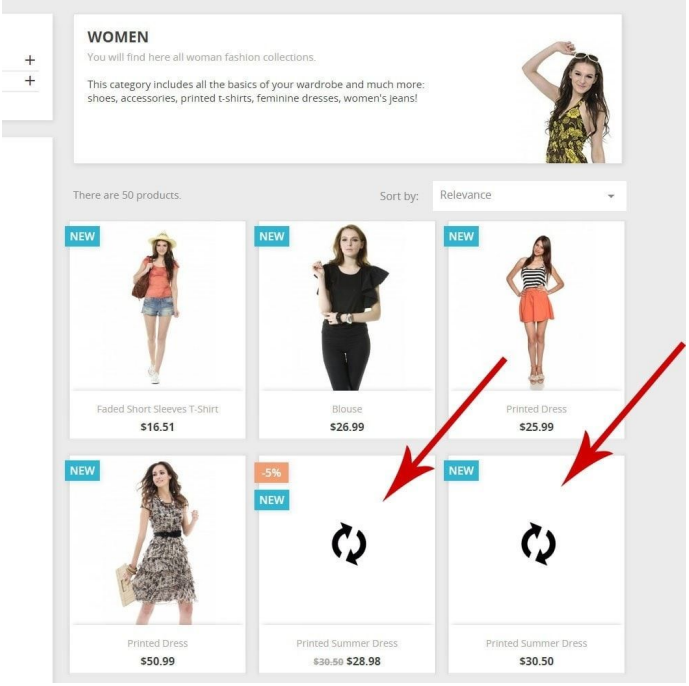
¿Qué es *“Lazy Loading”*?

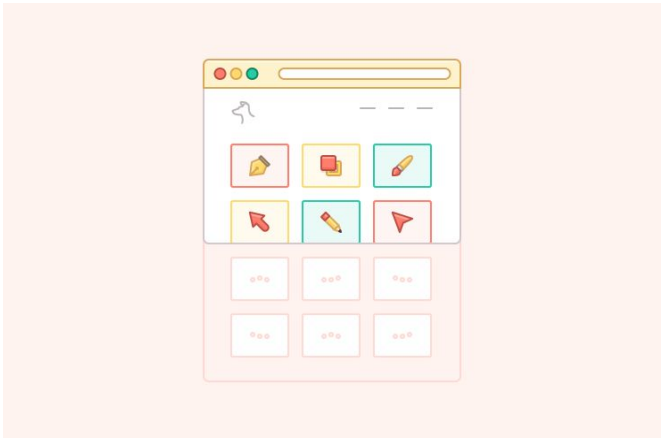
Normalmente cuando un usuario visita una web, todo el contenido que está en ella se descarga y se logra visualizar de inmediato aunque no está totalmente garantizado que vaya a poder ver todo el contenido.

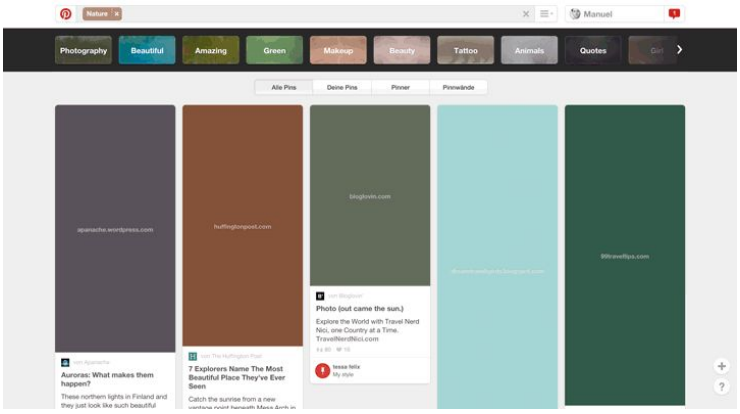
Por ejemplo, si una web es de galería de fotos, todas las fotos se bajan o descargan aunque el usuario no termine no termine de ver toda la galería, esto resulta una pérdida de memoria y de datos que va a consumir la persona aunque no haya visto todo el sitio web.

Para esto existe una opción que es “lazy loading” carga diferida y en este caso sería de imágenes, que permite que el contenido cargue a medida que el usuario vaya haciendo scroll en sección de la página. Por ejemplo es cuando se ve cuando el usuario está en un sitio y hace scroll, entonces se ve como la imagen no carga al instante, pero se va a ver el cuadro y en ese momento el usuario puede percibir que ahí hay una imagen, y de repente aparece la imagen final.

Algunos ejemplos visuales:



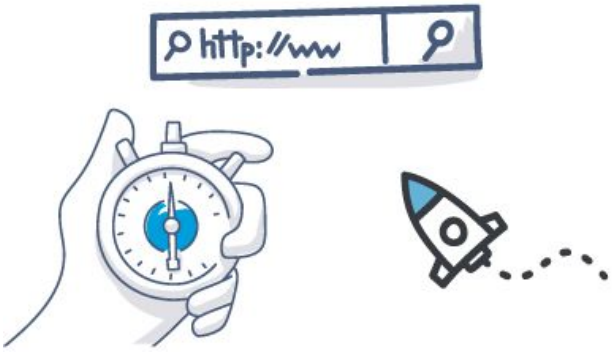




¿Es importante implementar *“Lazy Loading*” a nuestros sitios web?

Usar “Lazy Loading” es bastante importante implementarlo en nuestros sitios web, ya que al bajar el tamaño de los sitios se hacen menos peticiones, lo que significa mayor velocidad en el sitio.

Al tener menos carga en el contenido, hace que el sitio se vuelva más rápido y además se sabe que el tiempo de carga de un sitio es indispensable para tener más visitas y mejor performance en la página.



Algunas técnicas para implementar “Lazy Loading” son

1) TÉCNICA #1: Utilizando eventos de JavaScript

Esta técnica utiliza escuchas de eventos en los eventos de desplazamiento, cambio de tamaño y orientación en el navegador. El evento de desplazamiento es bastante claro porque observa dónde está el usuario en una página a medida que se produce el desplazamiento. Los eventos de cambio de tamaño y orientación son igualmente importantes.

El evento de cambio de tamaño ocurre cuando cambia el tamaño de la ventana del navegador, mientras que la Orientación-Cambio se activa cuando el dispositivo se gira de horizontal a vertical, o viceversa.

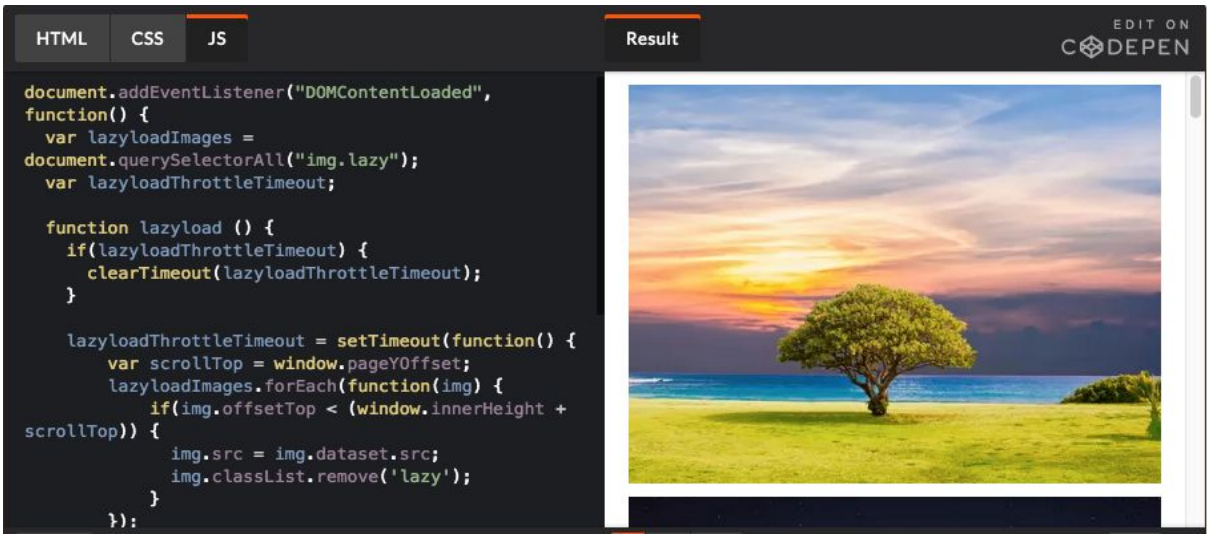
Podemos usar estos tres eventos para reconocer un cambio en la pantalla y determinar el número de imágenes que se vuelven visibles en la pantalla y hacer que se carguen en consecuencia.

Cuando ocurre alguno de estos eventos, encontramos todas las imágenes en la página que se han diferido y, a partir de estas imágenes, verificamos cuáles están actualmente en la ventana gráfica. Esto se hace usando el desplazamiento superior de la imagen, la posición superior del documento actual y la altura de la ventana. Si una imagen ha entrado en la ventana gráfica, seleccionamos la URL del atributo data-src y la movemos al atributo src y la imagen se cargará como resultado.

Tenga en cuenta que le pediremos a JavaScript que seleccione imágenes que contengan una clase perezosa. Una vez que la imagen se haya cargado, eliminaremos la clase porque ya no es necesario activar un evento. Y, una vez que se cargan todas las imágenes, también eliminamos los escuchas del evento.

Las tres primeras imágenes de este ejemplo se cargan por adelantado, ya que la URL está presente directamente en el atributo src en lugar del atributo data-src.

Esto es importante porque hace que haya una buena experiencia de usuario ya que las imágenes se encuentran en la parte superior de la página y es importante que aparezcan apenas el usuario entre a la página.



Código fuente de ejemplo en CodePen:https://codepen.io/imagekit_io/pen/MBNwKB

2) TÉCNICA #2: API Intersection Observer

Facilita la detección cuando un elemento entra en la ventana gráfica y realiza una acción cuando lo hace. En el método anterior, tuvimos que vincular eventos, tener en cuenta el rendimiento e implementar una forma de calcular si el elemento estaba en la vista o no. La API de Intersection Observer elimina toda esa sobrecarga al evitar los cálculos matemáticos y ofrecer un gran rendimiento fuera de la caja.

A continuación se muestra un ejemplo que utiliza la API para cargar de forma lenta imágenes. Adjuntamos el observador en todas las imágenes para que se carguen perezoso. Una vez que la API detecta que el elemento ha entrado en la ventana gráfica, utilizando la propiedad **isIntersecting**, seleccionamos la URL del atributo **data-src** y la movemos al atributo **src** para que el navegador active la carga de la imagen. Una vez hecho esto, eliminamos la clase **lazy** de la imagen y también eliminamos el observador de esa imagen.