SVG - Scalable Vector Graphics

Gráficos Vectoriales en HTML y CSS

¿Qué es SVG?

El formato SVG, que viene de las siglas Scalable Vector Graphics, es un estándar abierto desarrollado por el consorcio web internacional, que está basado en un XML que define las formas vectorialmente, y que tiene un lenguaje de etiquetas compatible con HTML, por tanto por CSS y por el DOM de JavaScript.

Esto implica que el SVG es un markup normal y corriente que funciona igual que HTML, y que es accesible desde el CSS y desde JS, que nos permite meter formas vectoriales en la web.

```
<svg width="300" height="300">
    <circle cx="40" cy="40" r="100" />
    <polygon points="200,10 250,190 160,210" />
</svg>
```

¿Cómo se crea el SVG?

El SVG se puede crear de varias maneras:

- Podemos crear SVG escribiendo el markup directamente, familiarizándonos con las etiquetas, y escribiéndolas como si fueran HTML
- Podemos crear el SVG en un programa de edición vectorial, exportándolo al formato SVG, y luego copiándo el contenido en el HTML.
- Podemos crearlo de manera procedural con librerías como D3.js
- Podemos crearlo con código con librerías como Snap.js, o SVG.js.

Plug-in para VSCode

Para poder trabajar con VSCode y que tengamos una ayuda para trabajar con SVG vamos a usar el plugin SVG Snippets.

Este plugin nos ayuda a escribir más rápido SVG y a descubrir sus atributos.

< svg >

La etiqueta svg es la etiqueta que indica que comienza un elemento svg. A partir de esta etiqueta todas las demás tienen sentido, si no, el navegador las ignoraría o no sabría que hacer con ellas.

Svg acepta bastantes atributos, aunque veremos algunos:

- Width y height: Son los atributos que nos dan la altura y anchura del elemento svg que es el contenedor de los vectores.

- Viewbox: es un sistema que tiene SVG para poder modificar los tamaños de manera relativa. Funciona como una especie de zoom. En realidad es una propiedad un poco compleja de manejar y no se usa mucho. Si los valores son 0 0 width height entonces no tiene efecto. En muchos casos es un atributo que se puede obviar.

```
<svg width="100" height="100"
     viewBox="0 0 100 100"
     version="1.1"
     xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     class=""
     id="">

     <!-- //... -->
</svg>
```

< svg >

En realidad muchos de estos atributos no son necesarios si introducimos el SVG en el HTML. Al final la etiqueta SVG nos quedaría simplemente así:

```
<svg class="mySVG" width="100" height="100">
    <!-- //... -->
</svg>
```

Y en caso de poner las medidas del SVG desde CSS (recordemos que svg es un elemento inline para CSS, se posiciona igual que una img)

```
<svg class="mySVG">
    <!-- //... -->
</svg>
```

Estilos

Vemos que al poner elementos svg sin ningún tipo de referencia a los estilos salen en color negro y sin trazo. Para dar estilo a los elementos SVG podemos usar CSS de tres maneras:

- Inline: es decir ponemos el atributo style en cada elemento. Esta se usa cuando queremos manipular SVG desde JS, que los cambios de estilo se ven reflejados en la etiqueta por su atributo style.
- In-SVG: Poner la declaración en la etiqueta SVG la información de CSS, se suele usar cuando servimos el SVG como imagen.
- O de manera normal en una hoja de estilo de CSS o de SASS. Cuando contamos con el SVG como un elemento más de nuestra maquetación. Yo personalmente recomiendo esta opción.

```
<svg class="mySVG">
    <rect x="100" y="100"
          width="200" height="200"
          style="fill: salmon;" />
</svg>
```

```
<svg class="mySVG">
    <style type="text/css">
        rect {
            fill: salmon;
        }
    </style>
    <rect x="100" y="100" width="200" height="200" />
</svg>
```

```
svg.mySVG rect {
    fill: salmon;
}
```

Propiedades CSS que se suelen usar en SVG con shapes

Hay muchas, ya que se pueden hacer muchas cosas con SVG (símbolos, degradados, máscaras), pero vamos a centrarnos en las de presentación.

fill: color de relleno de la forma

fill-opacity: opacidad de color de relleno de la forma;

stroke: color de trazo de la forma;

stroke-width: grosor de trazo de la forma;

stroke-opacity: opacidad del trazo de la forma;

stroke-dasharray: línea discontinua, marca la distancia entre segmento y segmento

stroke-dashoffset: nos permite recolocar los segmentos

stroke-linecap: los finales de linea se redondean o se dejan cuadrados

stroke-linejoin: los encuentros de linea y ángulos se redondean o se dejan cuadrados

stroke-miterlimit: en caso de tener un encuentro muy agudo, a veces se estira el trazo y queda mal. A partir de cuantos grados cortamos el vector. Si ponemos linejoin round (que lo recomiendo siempre) no hace falta.

Propiedades CSS que se suelen usar en SVG con shapes

Hay muchas, ya que se pueden hacer muchas cosas con SVG (símbolos, degradados, máscaras), pero vamos a centrarnos en las de presentación.

transform: Aplicamos transformaciones al vector

transform-origin: Las transformaciones se aplican desde la esquina superior izquierda del svg, con lo cual nos conviene modificar esto por center

transition: Transición como vimos anteriormente

animation: Animación como vimos anteriormente

visibility: hidden - visible

opacity: opacidad de todo el elemento en general

```css
svg.mySVG rect {
    fill: salmon;
    stroke: black;
    transform-origin: center center;
    transform: translate(30%, 20%) rotate(45deg);
}
```

line y paths

Podemos dibujar líneas y trazados de diferentes tipos. (Recordad poner stroke y stroke-width en la línea si queremos verla!)

```
<line x1="0" y1="0" x2="300" y2="200" />
```

Y así es como se dibuja un trazado complejo con curvas.

```
<path d="M36 6.5c-4 11.5-8.5-16-22.5-2S-2 25.5 4 36s32.787 13.856 40 0c5.511-10.588-4-41-8-29.5z" />
```

line y paths

Como vemos, no somos capaces de entender qué está pasando ahí. Son números decimales con espacios con puntos y con letras. Todo esto obedece a un estándar de descripción de curvas, cuya tabla os pongo aquí para que la leáis con calma.

https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths#Curve_commands

https://codepen.io/lingtalfi/pen/yaLWJG

```
<path d="M36 6.5c-4 11.5-8.5-16-22.5-2S-2 25.5 4 36s32.787 13.856 40 0c5.511-10.588-4-41-8-29.5z" />
```

Herramientas externas para hacer SVG

Pero obviamente nadie dibuja SVG haciendo números con la tabla. Sería un trabajo horrible. Para ello contamos con herramientas de dibujo vectorial y herramientas de optimización de vectores.

Herramientas de trabajo con vectores:

- Adobe Illustrator: Es el rey (y seguirá siéndolo) del dibujo vectorial. Se pueden guardar los archivos como svg.
- InkScape es la opción Open Source de Illustrator que funciona para vectores svg e interactividad mejor si cabe.
- Sketch y Figma: Son mis opciones favoritas, porque aún no teniendo tantas opciones como Illustrator o InkScape, tienen las necesarias para dibujar vectores, permiten exportar fácilmente y están bastante optimizados ya de por sí.

Herramientas externas para hacer SVG

En caso de usar alguna herramienta para dibujar vectores, suele suceder que nos crean bastante código innecesario, o ponen demasiados decimales en el cálculo de puntos, etc… para ello conviene optimizar o limpiar el código.

Hay una herramienta online estupenda para ello:
SVG-OMG: https://jakearchibald.github.io/svgomg/

Ejemplo con Illustrator

Creamos un vector con Illustrator con la herramienta pluma o mediante operaciones booleanas, y lo guardamos como SVG.

Si miramos el markup que nos genera es un poco complejo con muchos elementos. Esto es porque no es el mismo tipo de markup que usamos si es para cargarlo como imagen externa tipo MIME, o si lo vamos a usar como markup en HTML.

En caso de usarlo en HTML conviene optimizarlo, para ello usamos la herramienta SVG-OMG online: https://jakearchibald.github.io/svgomg/

Lo colocamos en HTML y voilá, limpio, claro y accesible a CSS y JS.

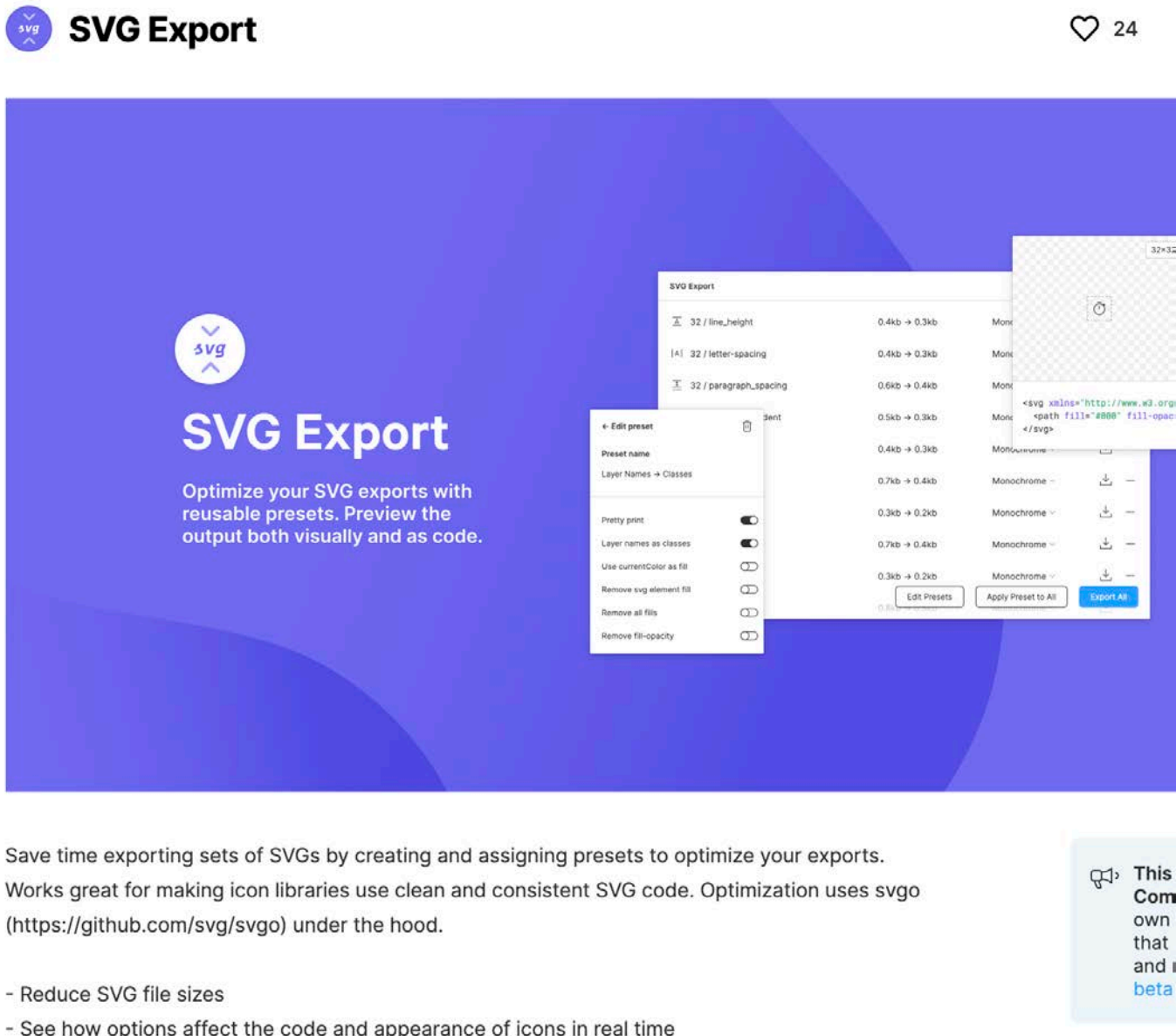
Ejemplo con Figma

Creamos un vector con Figma con la herramienta pluma o mediante operaciones booleanas (no olvidemos darle a flatten vector), y exportamos el vector con el plugin SVG Export

SVG export como plugin incorpora todo lo necesario para la optimización como en SVG-OMG

https://www.figma.com/community/plugin/814345141907543603/SVG-Export

Lo colocamos en HTML y voilá



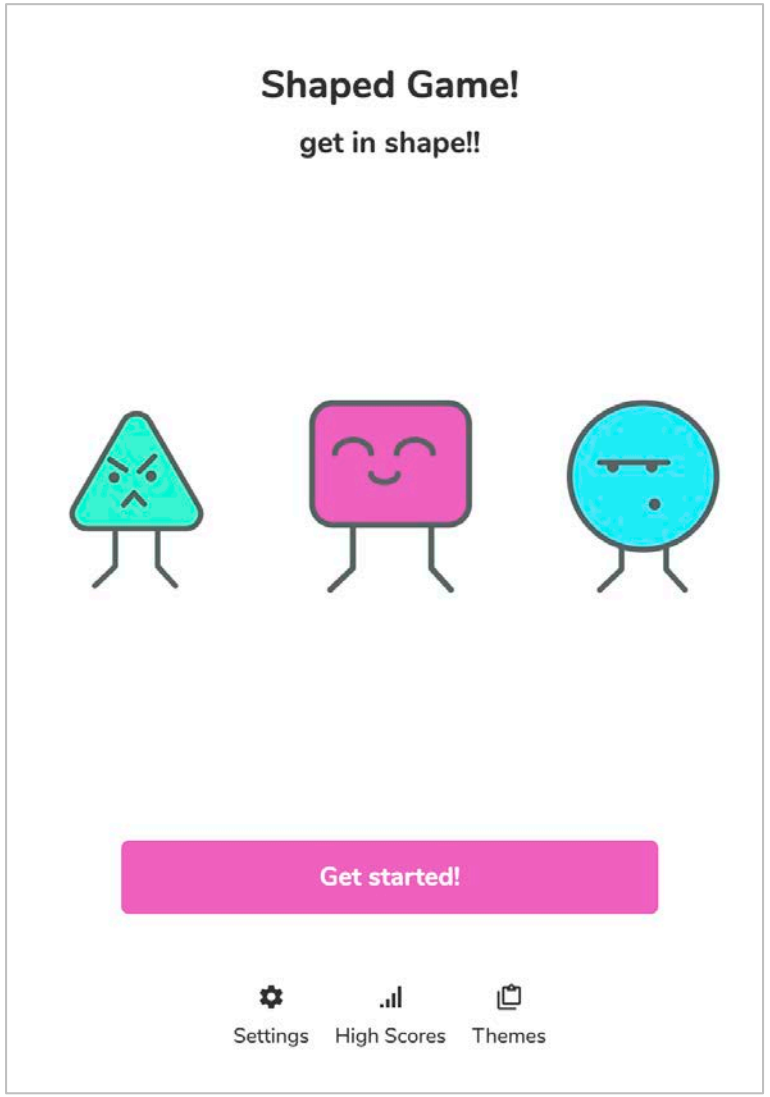
Por qué usar SVG inline

SVG inline es la manera de decir que estamos metiendo el SVG junto con el HTML.

Siendo el SVG un sistema de marcación válida para la web, tenemos dos cosas interesantes:

- La primera es que podemos usar estilos CSS para sobreescribir los estilos propios del SVG y controlarlos a nuestro gusto
- La segunda y super interesante es que siendo un elemento de marcación en el HTML, es accesible desde el DOM, y por tanto recibe eventos, se puede editar desde JS, etc…
- La tercera, que puede trabajarse las cuestiones de accesibilidad de una manera más elaborada.

Echa un ojo al ejemplo para ver una manipuación de SVG con JS!!



Para aprender más

Hay dos personas clave en el mundo del SVG que son Sara Soueidan y Sarah Drasner.

A parte de ser unas super desarrolladoras, han generado un volumen enorme de información de cosas que se pueden hacer con SVG.

Merece la pena seguir a estas dos desarrolladoras y diseñadoras.

https://www.youtube.com/watch?v=lf7L8X6ZBu8

https://www.youtube.com/watch?v=4laPOtTRteI

Para hacer graphs e infografías

Hemos visto seguramente en periódicos y revistas digitales un montón de infografías interesantes basadas en vectores.

En este caso se usan maneras procedurales de crear vectores, esto es, maneras programadas.

Hay varias librerías potentes que nos permiten crear y manejar vectores como Snap.svg o SVG.js y para trabajar con SVG y datos al mismo tiempo, D3.js o Chartist