# liquidStrip

## Manual del Programador

Índice

# 1.- Introducción

## Por qué

La creación de liquidStrip nace de la creencia fundamental de que leer cómics no debería ser un suplicio, así como de la necesidad de adaptar esta forma de comunicación a los dispositivos móviles que tanto usamos hoy en día. No queremos que liquidStrip sirva de reemplazo al cómic tradicional, sino que permita que este se expanda a lo largo (y ancho) de otras pantallas. Nuestro objetivo es proveer de una plataforma, una rejilla de rejillas sobre la que artistas como tú puedan contar sus historias sin que estas estén limitadas por los bordes de un folio... o los píxeles de una pantalla.

liquidStrip es un sistema adaptativo para la creación de cómics. En este manual encontrarás toda la información necesaria para utilizar las rejillas base que incluimos en el kit del artista, así como tutoriales, ideas y conceptos que puedes aplicar para facilitar que tus lectores lean tu cómic de la manera en la que tú quieres que sea leído. La filosofía detrás de liquidStrip es adaptar los cómics al siglo XXI sin romper el pacto autor/lector, esto es: narrativas gráficas que fluyan en tamaño y disposición entre los diferentes dispositivos que usamos constantemente. Queremos que nuestros lectores empiecen a disfrutar de un cómic en su PC, y que cuando tengan que coger el metro para ir al trabajo, sigan leyéndolo en su móvil o tablet.

## Para quién

Para crear rejillas con liquidStrip no necesitas ser un genio de la programación, aunque si esperas poder realizar proyectos más complejos con parallax, muchas animaciones, etc. es posible que pases muchas horas delante de un archivo de código y varias pestañas de codepen y github. Si vas a crear tus propias rejillas tendrás que tener una cierta idea sobre narrativas de cómic, o juntarte con alguien que pueda desarrollar la parte teórica por ti. Si, por otra parte, sólo quieres jugar con nuestras plantillas, es suficiente con que alguna vez hayas tocado el código de tu web, blog, o foro.

## Qué necesitas

Conocimientos básicos de HTML/CSS y Javascript (si quieres incluir animaciones), acceso a un ordenador y conexión estable a internet, Sublime Text, Brackets, Notepad++ (o tu editor favorito) y un poco de paciencia.

# 1.1.- HTML/CSS 101

Si nunca has hecho esto antes, si nunca has escrito código o modificado alguno, no tengas miedo: el código no muerde. Además, todo nuestro código está anotado para que entiendas qué hace cada cosa y cómo funciona. ¡Somos chicas ordenadas!

La especificación oficial de HTML y CSS la puedes consultar en las páginas del w3c

https://www.w3.org/TR/html52/
https://www.w3.org/Style/CSS/

Y te recomendamos también consultar la web HTML5 donde encontrás, además del enlace a la especificación oficial, una explicación detallada de cada elemento HTML, su uso y evolución a lo largo de los años, etc.

http://html5doctor.com/

Y para CSS:

https://tympanus.net/codrops/css_reference/
https://developer.mozilla.org/es/docs/Web/CSS/Referencia_CSS
https://cssreference.io/

Como quizás esta documentación te pareza demasido técnica en ocasiones, entendemos que tal vez necesites empezar por alguna parte. Por eso dejamos a tu disposición los siguientes recursos que te recomendamos leer: ellos lo explican todo de manera sencilla.

https://developer.mozilla.org/es/docs/Learn/CSS/Introduction_to_CSS
https://www.csstutorial.net/
https://uniwebsidad.com/libros/xhtml
https://uniwebsidad.com/libros/css

liquidStrip utiliza los siguientes elementos para construir el comic

<img>
<picture>
<source>

Para aplicar los estilos usamos clases, `class="nombredelaclase"`. ¡Y ya está!

El CSS es un poco más complejo. Utilizamos las siguientes propiedades:

grid-template-area
grid-template-columns
align-self
grid-area

Además de algunas otras, pero estas son las más importantes y con las que tú puedes jugar para adaptar mejor las rejillas.

**Nota soporte:**

**Hemos utilizado CSS Grid para la maquetación de los comics, ya que es el método de maquetación que nos permite visualizar el comic como queremos (era imposible utilizando float, inline-block o flexbox). En navegadores antiguos no existe soporte para esta tecnología. Se puede consultar el soporte de esta propiedad en : https://caniuse.com/#feat=css-grid**

# 2.- Rejillas modulares

En esta sección cubriremos tanto cómo rellenar nuestras plantillas así como la creación de tu propia rejilla, desde el punto de vista de la programación: te recomendamos que consultes algunos libros sobre narrativa para la parte más teórica ;)

## 2.1.- Traslación de la imagen al código

A la hora de insertar las imágenes, tendremos dos casuísticas:
a) que en todos los tamaños de dispositivo se muestre el mismo archivo de imagen, o bien b) que en cada tamaño de dispositivo se muestren archivos diferentes.

a) Mismo archivo de imagen en todos los tamaños de dispositivo:
utilizarmos el elemento <img> para insertar la imagen.
Este elemento necesita ir acompañado de dos atributos, la ruta y el texto alternativo (alternativa textual a la imagen en caso de que no se cargue o no podamos ver la imagen):

```
<img src=”ruta/archivo_imagen” alt=”texto alternativo”>
```

b) Diferente archivo de imagen en cada tamaño de dispositivo:
En este caso, como tenemos que cargar diferentes archivos en un único bloque, siendo el navegador quien determine qué archivo se carga en función del tamaño de ventana, no podemos utilizar el elemento <img> tal cual.
Necesitamos utilizar su “hermano mayor”, el elemento <picture>

Este elemento funciona como un contenedor en el cual le indicamos qué archivo de imagen debe cargar según el tamaño de ventana,
Dentro de <picture> vamos a utilizar el elemento <source> y también un <img>
Por ejemplo:

```
<picture>
      <source srcset="ruta/archivo_imagen_grande" media="(min-width: 1500px)">
      <source srcset="ruta/archivo_imagen_mediana" media="(min-width: 800px)">
      <img src="ruta/archivo_imagen_pequeña" alt="texto alternativo">
</picture>
```

Estas “media queries” son un ejemplo, claro. Cada cómic tendrá las suyas propias.

Para pantallas mayores que 1500px, se cargará la imagen grande.
Para pantallas menores que 1500px y mayores que 800px, se cargará la imagen mediana.
Y por último, para pantallas menores que 800px, se cargará la imagen pequeña.

Copiamos y pegamos el HTML correspondiente a la plantilla que queramos usar, y reemplazamos las imágenes de ejemplo con las nuestras.

Copiamos y pegamos el CSS en el tag <head> del documento, si el código tiene que funcionar dentro de un CM o no vamos a subirlo por FTP. Si no, podemos enlazar la hoja de estilos.

Si queremos usar algún tipo de animación o extra, tendremos que enlazarlo en <head> y aplicarlo según las instrucciones.

Guardamos y subimos el documento con nuestro método preferido.

## 2.2.- Cómo crear tu propia rejilla

En primer lugar, necesitaremos tres bocetos: uno para la distribución de las viñetas en el tamaño S, otro para la distribución de las viñetas en el tamaño M y otro para su distribución en el tamaño L. El tamaño S debería tener 480px de ancho, el tamaño M 1330px y el L 1920px. Todos deben tener la misma cantidad de viñetas, pero los tamaños/proporciones de estas pueden cambiar: aunque recuerda, cuantos más cambios de proporciones, más trabajo para el artista gráfico. Recuerda también usar una base de 5, 7 y 8 columnas respectivamente.

Recomendamos encarecidamente leer sobre CSSGrid antes de comenzar este proceso. En particular su documentación y éste artículo sobre sus usos.

Para la creación de las rejillas usaremos la propiedad de matriz de CSSGrid: asignaremos una letra o nombre a cada cuadro/viñeta (nosotros usamos las letras del alfabeto porque son fáciles de recordar y solo escribimos un carácter por lo que facilita la claridad del código, pero podéis usar lo que sea que os funcione), y luego escribiremos una matriz que represente los diferentes bocetos. El tamaño S tiene 5 columnas, el M 7 y el L 9. Usa excel ya que es la manera más rápida de asegurarte de que funciona :)

Lo último que tenemos que hacer es asegurarnos de que la rejilla está mostrando la versión que tiene que mostrarse en función del tamaño de pantalla desde el que se está viendo el cómic con `@media screen`.

Para algunas versiones de los comics en realidad no haría falta utilizar CSSGrid (ya que se podría haber empleado un método de maquetación más sencillo), pero para homogeneizar todo el código hemos optado por utilizarlo siempre.

# 3.- Taxonomía y clases

La arquitectura de estilos de liquidStrip es muy sencilla: cada plantilla, que contiene hasta 9 bloques de imagen (block1, block2, block3... block9), es un “comic” (comic1, comic2, comic3...). Cada comic tiene un grid propio que lo diferencia de los demás. De esta manera, podemos seleccionar cada “celda” de cada cómic para aplicarle estilos, ajustes, etc.

| | | Comics | | |
|---|---|---|---|---|
| | | .comic1 | .comic2 | .comic3 |
| Bloques | .block1 | .comic1 .block1 | .comic2 .block1 | .comic3 block1 |
| | .block2 | .comic1 .block2 | .comic2 .block2 | .comic3 .block2 |
| | .block3 | .comic1 .block3 | .comic2 .block3 | .comic3 .block3 |
| | .block4 | .comic1 .block4 | .comic2 .block4 | .comic3 .block4 |

La estructura de la plantilla queda expresada mediante “grid-template-areas” en una estructura de matriz. Cada tamaño de pantalla tiene una estructura diferente, determinando cuál se muestra con “@media screen” y “max-width”.

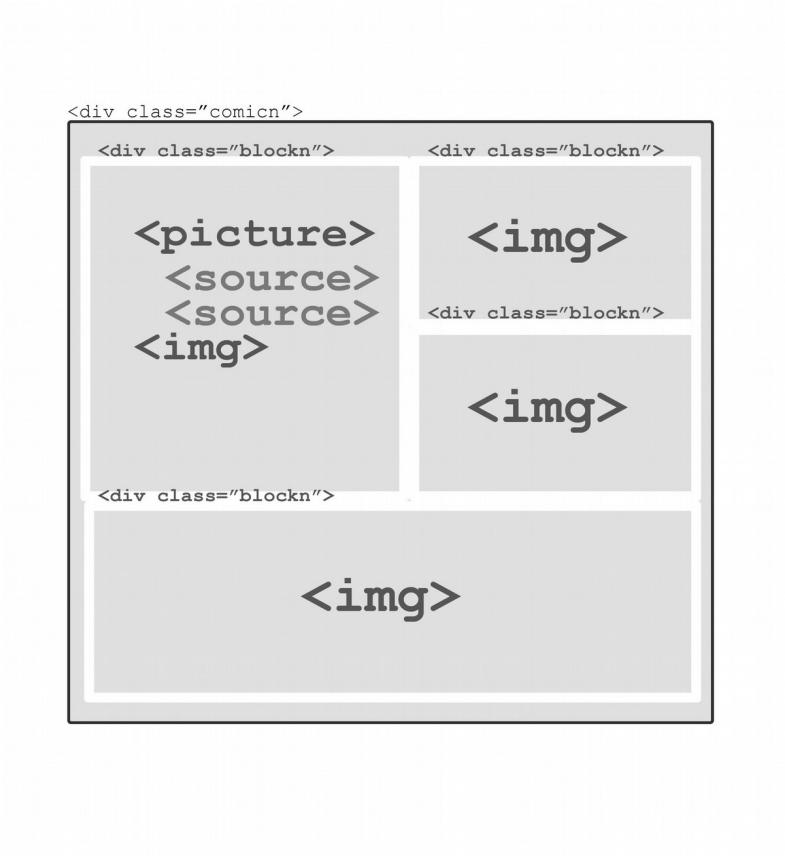
```
@media screen and (max-width:550px){
      .comic1{
            grid-template-columns: 1fr;
            grid-template-areas:
                  "a"
                  "b"
                  "c";
      }

}


@media screen and (min-width:551px) and (max-width:1000px){

      .comic1{
            grid-template-columns: repeat(7, 1fr);
            grid-template-areas:
                  "a a a b b b b"
                  "a a a c c c c";
      }

      .comic1 .block2{align-self:end;}

}


@media screen and (min-width:1001px){

      .comic1{
            grid-template-columns: repeat(8, 1fr);
            grid-template-areas:
                  "a a a b b b b ·"
                  "a a a c c c c ·";
      }

      .comic1 .block2{align-self:end;}

}
```

Cada “block” se corresponde con una letra, en orden alfabético, lo que facilita la identificación y orden de las viñetas.

```
.block1{grid-area: a;}
.block2{grid-area: b;}
.block3{grid-area: c;}
.block4{grid-area: d;}
.block5{grid-area: e;}
```

```
.block6{grid-area: f;}
.block7{grid-area: g;}
.block8{grid-area: h;}
.block9{grid-area: i;}
```

En cuanto a la arquitectura HTML, es también muy sencilla. Cada plantilla es un  de clase “comic”; y cada viñeta es un  de clase “block”, que contiene una imagen en su interior. Además, en ocasiones se usa la etiqueta <picture> con <srcset> para poder cambiar la imagen que se muestra en función del tamaño de la pantalla, permitiendo así optimizar las plantillas para todos los dispositivos:



# 4.- Elementos externos

El tratamiento de los gifs es exactamente igual que el de las imágenes, sólo tenemos que incluir el enlace o la ruta en el que los hayamos guardado.

Los videos tendrán que estar alojados con algún servicio tipo Youtube, Vimeo, etc. Estas páginas nos dan un código para incrustar el video.

```
<iframe width="560" height="315" src="https://www.youtube.com/
embed/jKFNYhpxi6k" frameborder="0" allow="accelerometer;
autoplay; encrypted-media; gyroscope; picture-in-picture"
allowfullscreen></iframe>
```

En este código, vienen ya predefinidas unas anchuras y alturas de video. Sin embargo, los videos no funcionan como las imágenes que, al redimensionar la ventana del navegador, éstas se adaptan en anchura y altura proporcionalmente.

Los videos son elementos diferentes y aunque establezcamos una anchura del 100% del bloque padre, la altura no cambiaría.

Es por ello que tenemos que hacer un pequeño truco para que el video sea también responsive y se adapte al bloque donde vaya insertado.

Cuando insertemos el código, debemos englobarlo además por un elemento  con la clase “rwd-container”, De esa manera, el video se adaptará perfectamente con una proporción de 16:9.

Si queremos que el video se adapte en base a otras proporciones, entonces tendremos que añadir una segunda clase al :

```
.ratio-9-16
.ratio-2-3
.ratio-4-3
.ratio-4-5
.ratio-1-1
```

Por lo tanto, para insertar el video del ejemplo anterior, tendríamos el siguiente código:

```
<div class="rwd-container">
      <iframe width="560" height="315" src="https://www.youtube.com/
      embed/jKFNYhpxi6k" frameborder="0" allow="accelerometer;
      autoplay; encrypted-media; gyroscope; picture-in-picture"
```

```
      allowfullscreen></iframe>
</div>
```

El video se adaptaría en base a una propoción de 16:9,

Si quisiéramos que se adaptase en base a una proporción de 9:16, entonces pondríamos:

```
<div class="rwd-container ratio-9-16">
      <iframe width="560" height="315" src="https://www.youtube.com/
      embed/jKFNYhpxi6k" frameborder="0" allow="accelerometer;
      autoplay; encrypted-media; gyroscope; picture-in-picture"
      allowfullscreen></iframe>
</div>
```

Esto vale para iframe, video, object....

En el caso del sonido, por suerte HTML5 hace realmente sencillo introducir un reproductor en nuestra página. Podemos decidir si la música empieza automáticamente en la página o si el lector puede controlarla, así como si se reproduce en bucle o no. Esta etiqueta está soportada por la mayoría de los navegadores en su versión para MP3

```
<embed height="60" type="audio/midi" width="144" src="audio.mp3"
volume="60" loop="false" autostart="false" />

<audio controls>
     <source
src="/assets_tutorials/media/Loreena_Mckennitt_Snow_56bit.mp3"
type="audio/mpeg">
     <source
src="/assets_tutorials/media/Loreena_Mckennitt_Snow_56bit.ogg"
type="audio/ogg">
     Your browser does not support the audio tag.
</audio>
```

Si el reproductor fuese demasiado grande, tendríais que utilizar un truco similar al de los vídeos.

Por último, necesitaremos importar algunas librerías para realizar animaciones.

Para las animaciones que funcionarán según hagamos scroll (viñetas que aparecen desde el lado, etc.) aprovecharemos que cada viñeta ya está metida en un  y utilizaremos la librería AOS de Mich Alsnik, que sólo necesita que la importemos y utilicemos las etiquetas correspondientes.

https://michalsnik.github.io/aos/

Deberemos copiar lo siguiente en el elemento <head>:

```
<link rel="stylesheet" href="https://unpkg.com/aos@next/dist/aos.css" />
```

Y este script antes de cerrar <body>:

```
<script src="https://unpkg.com/aos@next/dist/aos.js"></script>
  <script>
    AOS.init();
  </script>
```

Después, para aplicar el efecto sólo necesitaremos inicializar AOS e indicar la clase. Un ejemplo es el uso de esta animación en el ejemplo de tira cómica en la web:

```
<div class="parte5" data-aos="zoom-in" data-aos-delay="150">
                    <img
src="https://serving.photos.photobox.com/18841772c4fa9f2732e021801db1fbaffa99a67b41b752f3453abf780c8879d2cfce923f.jpg" alt="">
            </div>
```

Referíos a la documentación para más detalles:

https://github.com/michalsnik/aos

Recomendamos  esta librería porque cubre las necesidades básicas de animación, es muy sencilla de utilizar, es muy ligera, utiliza código estándar y no tiene dependencias de otras librerías javascript (como podría ser jquery o nodejs) para su funcionamiento.
Por supuesto se pueden utilizar otras librerías similares, a gusto del programador.

Por otra parte, si queremos animar una imagen en una capa superior, una viñeta en concreto, etc. podemos usar Animate.css, que no es dependiente de Javascript (por supuesto, en este caso no está preparada para usar javascript para determinar cuándo ocurren los eventos)

https://daneden.github.io/animate.css/