TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE:

- Preprocesadores de CSS.
- Sass.
- Sintaxis de Sass.
- Cómo utilizar Sass.
- Estructurando un proyecto con Sass.
- Características de Sass que permiten estructurar nuestros proyectos.
- Crear un proyecto con Sass.

PREPROCESADORES DE CSS

Éstos corresponden a programas que permiten generar CSS a partir de distintos archivos escritos con la sintaxis propia del preprocesador. Permiten ahorrar tiempo al momento de realizar una maqueta y separar la lógica visual, además de crear hojas de estilos más legibles, consistentes y fáciles de mantener.

Existen diversos preprocesadores de CSS:

- Sass.
- Less.
- Stylus.
- PostCSS.

Durante este curso utilizaremos el preprocesador Sass.

SASS

Siglas de “Sintactically Awesome Style Sheets”, es una poderosa herramienta que permite crear hojas de estilo legibles, consistentes y fáciles de mantener. Para emplearla no es necesario aprender una sintaxis nueva, por lo que puedes seguir utilizando los archivos que ya tienes. Reutilizar código ayuda a no repetir clases, lo cual es muy beneficioso para evitar la redundancia y mantener su simplicidad.

Además, permite trabajar en proyectos colaborativos de gran magnitud, reduciendo el tiempo de creación y mantenimiento de las hojas de estilo. Esta herramienta entrega organización y escalabilidad tanto a proyectos personales como colaborativos, reduciendo la cantidad de código.

Gracias a que Sass es compatible con todas las versiones de CSS, se puede implementar sin problemas en cualquiera de sus bibliotecas disponibles. Para mantener esta compatibilidad con versiones futuras, tiene un soporte general que cubre casi todas las reglas de forma predeterminada. Una regla CSS en Sass se escribe `@` o `@{…}`.

Otra de sus ventajas, es la creación de variables y el anidamiento. El uso de dichas **variables** permite modificar de manera rápida y sencilla el valor de un atributo de estilo en más de un sitio del archivo, solo modificando el valor de la variable, evitando así tener que reemplazar el valor sitio por sitio.

```scss
// SCSS
$font-stack: helvetica, sans-serif;
$primary-color: #333;

body {
    font: 100% $font-stack;
    color: $primary-color;
}
```

```css
// CSS
body {
    font: 100% helvetica, sans-serif;
    color: #333;
}
```

Por su parte, el anidamiento permite anidar selectores y así evitamos repetir prefijos coincidentes en cada selector CSS.

```scss
// SCSS
nav {
    ul {
        margin: 0;
        padding: 0;
        list-style: none;
    }

    li {
        display: inline-block;
    }

    a {
        display: block;
        padding: 6px 12px;
        text-decoration: none;
    }
}
```

```css
// CSS
nav ul {
    margin: 0;
    padding: 0;
    list-style: none;
}

nav li {
    display: inline-block;
}

nav a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
}
```

El **selector** & (parent selector), se utiliza en selectores anidados para referirse al más externo. Permite reutilizarlo de maneras más complejas, como añadiendo una pseudo-clase o un selector antes del "padre".

```scss
// SCSS
.message {
    // El selector padre puede ser usado para añadir pesudo-clases al
    // selector más externo
    &:hover {
        color: red;
    }

    // También puede ser usado para agregar estilo al selector externo
    // en cierto contexto
    p & {
        font-weight: bold;
    }

    // También se puede utilizar como argumento de selectores de
pseudo-clases
    :not(&) {
        font-size: 0.8rem;
    }
}
```

```css
// CSS
message:hover {
    color: red;
}

p message {
    font-weight: bold;
}

:not(message) {
    font-size: 0.8rem;
}
```

También se pueden **extender** atributos desde otras declaraciones, lo que nos permite tener dos selectores con el mismo estilo, pero que presenten alguna diferencia.

```scss
// SCSS
.success {
    font-size: 1.5rem;
    color: green;

    &_bold {
        @extend .success;
        font-weight: bold;
    }
}
```

```css
// CSS
.success, .success_bold {
    font-size: 1.5rem;
    color: green;
}

.success_bold {
    font-weight: bold;
}
```

Además, se pueden importar ficheros con `@import`. En Sass, los ficheros se dividen en varios más pequeños y fáciles de mantener. Esta herramienta extiende la regla @import para importar hojas de estilo Sass y CSS, entregando acceso a mixins, funciones y variables, además de combinar CSS de varias hojas de estilo. A diferencia del `@import` de CSS, que requiere que el navegador realice múltiples solicitudes HTTP durante la carga de una página, las importaciones Sass se realizan durante la compilación.

Por otro lado, están los `mixin`, los cuales permiten definir estilos que pueden ser reutilizados a lo largo de tu hoja de estilos. Éstos ayudan a evitar el uso de clases no semánticas, como `.float-left`, y contribuyen a distribuir colecciones de estilos en librerías. Están definidos por la regla `@mixin <nombre> {…}` o `@mixin nombre(<argumentos>){…}`. Y, son incluidos en el contexto donde se necesiten, utilizando la regla `@include`, que se escribe `@include <nombre>` o `@include <nombre>(<argumentos>)`, con el nombre del `mixin` incluido.

```scss
// SCSS
@mixin transformar($propiedad) {
    -webkit-transform: $propiedad;
    -ms-transform: $propiedad;
}

.box {
    @include transform(rotate(30deg));
}
```

```css
// CSS
.box {
    -webkit-transform: rotate(30deg);
    -ms-transform: rotate(30deg);
    transform: rotate(30deg);
}
```

SINTAXIS DE SASS

Existen dos tipos para escribir el código en Sass:

- *Sintaxis Sass*: ésta no difiere mucho de CSS estándar. Evita colocar punto y coma al final de los valores de las propiedades, así como tampoco se utilizan llaves, y en su lugar se utilizan indentados.
- *Sintaxis SCSS*: es muy similar a CSS (de hecho, el código CSS es código SCSS válido). Se puede decir que es un código CSS con elementos extras. Su ventaja es que se puede utilizar código CSS anterior, o de otros proyectos, y será válido para el preprocesador. En este curso utilizaremos SCSS.

CÓMO UTILIZAR SASS

Existen tres alternativas:

1. Utilizar aplicaciones de pago, las cuales entregan herramientas de interfaz gráfica.
2. Utilizar la terminal y líneas de comando. Es la opción más usada por los desarrolladores, ya que no requiere comprar licencias y se integra fácilmente al ecosistema de herramientas.

3. Utilizar herramientas automatizadas, las que permiten optimizar el flujo de trabajo del maquetador, compilando archivos CSS, JavaScript, optimizando imágenes, entre otros. Existen paquetes como: Gulp, Grunt, PostCSS y Webpack, que tienen todo lo necesario para cubrir las necesidades de trabajo con distintos lenguajes web, no solo Sass. Es una muy buena alternativa, pero requiere de conocimientos avanzados que no se desarrollarán en este curso.

ESTRUCTURANDO UN PROYECTO CON SASS

Permite aumentar el orden y mantenibilidad de nuestros estilos, lo cual es necesario ya que otras personas los utilizarán y mantendrán en el futuro. Mantener un buen orden en un proyecto en Sass implica que los estilos estarán organizados de manera que buscar un elemento específico sea fácil, evitando errores por cascada de CSS y haciendo sencilla la mantención del código para otros.

Para lograr lo anterior, es necesario reconocer cómo se separa la lógica visual de una interfaz de usuario. Esto contempla identificar los componentes y elementos que conforman un diseño:

- Colores, tipografías, tamaños, entre otros contenidos visuales.
- Diseño base de la interfaz: de qué manera están estructurados sus componentes.
- Elementos que componen la interfaz: botones, inputs, navegación, entre otros.
- Interacciones de la interfaz.

CARACTERÍSTICAS DE SASS QUE PERMITEN ESTRUCTURAR NUESTROS PROYECTOS

1. **Archivos parciales:** son aquellos en los que se pueden agregar pequeños fragmentos de estilos.
   Para ser reconocidos por Sass deben comenzar con un guion bajo `(_)`, esto además permite reconocer que los archivos no deben compilarse a CSS, a menos de que nosotros lo establezcamos.
   Un ejemplo de archivo parcial podría ser el estilo de los subtítulos de un sitio web, los cuales se encontrarán modularizados en este `(_subtitles.scss)`.

   Para poder compilar un archivo parcial a CSS se utiliza `@import`.

- **@IMPORT:** corresponde a una extensión de la opción `@import` de CSS, la cual toma el archivo que se quiere importar y lo combina con otro. Un ejemplo: hemos terminado los subtítulos del sitio y ahora debemos compilarlos a CSS. Para poder realizar esto, es necesario utilizar `@import` dentro del archivo SCSS `main.scss` de la siguiente manera:

```
// El archivo parcial se encontrará en el directorio
"components",
// dentro de la carpeta raíz de Sass
@import "components/subtitles"
```

  Como se observa, se pueden utilizar rutas relativas y no es necesario explicitar la extensión del archivo. Además, debemos crear una estructura de directorios que divida los archivos parciales del principal `(main.scss)`.

2. **Manifiestos:** se refiere a la hoja principal de estilos, dentro de la cual se realizarán los `@import` necesarios para llamar a los archivos parciales del proyecto. Un manifiesto en Sass, corresponde a un archivo que en su interior contiene un grupo de archivos importados, los cuales conforman una unidad que ensamblada representa al archivo principal o input `(main.scss)` que será compilado posteriormente a CSS.
   Es importante ordenar de manera correcta la importación de archivos parciales al manifiesto, ya que algunas reglas CSS podrían depender de otras para funcionar.

3. **Comentarios:** son necesarios para ordenar y explicar nuestro código. En Sass, existen dos tipos:
   - Comentarios de bloque: se usan comúnmente en CSS nativo. Se compilan a CSS, es decir, se mantendrán en el archivo luego de realizar la compilación.

     `/* Este es un comentario de bloque */`

   - Comentarios de línea: son exclusivos de Sass, por lo que los comentarios de este tipo que realicemos en nuestros archivos SCSS no compilarán a CSS.

     `// Este es un comentario de línea`

CREAR UN PROYECTO CON SASS

Antes de poder crear un proyecto con Sass es necesario conocer el patrón 7-1.

Patrrón 7-1

Es una estructura creada por Hugo Giraudel, que se basa en un sencillo principio: 7 directorios, 1 archivo. Es decir, separa los parciales en 7 directorios diferentes y deja solo un archivo que funciona como manifiesto.

La estructura base de esta técnica contempla 7 directorios, los cuales se encuentran dentro de la carpeta raíz. Además, dentro de ésta se aloja el manifiesto (*main.scss*), que contendrá a su vez los archivos parciales de nuestro proyecto Sass.

Dichos archivos parciales se dividirán dentro de los distintos directorios:

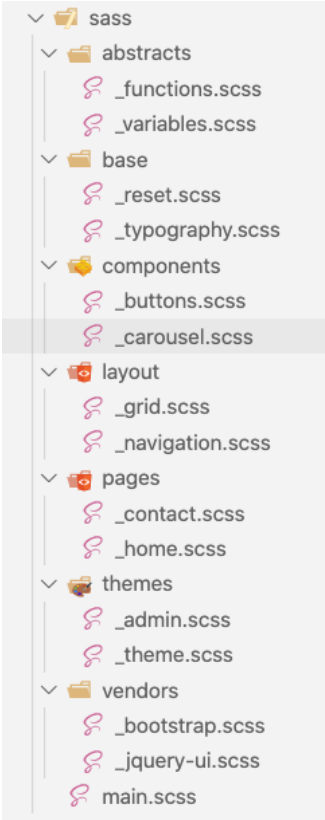

Imagen 1. Patrón 7-1

A continuación, se definirán los diferentes directorios que conforman el patrón 7-1.

- **ABSTRACTS**

  Contiene los parciales relacionados a las herramientas y helpers de Sass, utilizados en el proyecto. Encontraremos variables globales, funciones y mixins. Es importante que ninguno de los elementos incluidos en este directorio compile a CSS.

Imagen 2. Directorio "abstracts".

- **BASE**

  En éste podemos guardar los estilos principales, tales como: tipografías, reset y estilos comúnmente usados en HTML.

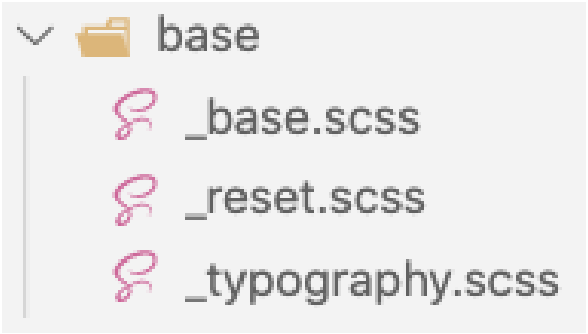

Imagen 3. Directorio "base".

- **COMPONENTS**

En éste irán los parciales que representen pequeños componentes de nuestra interfaz. Por ejemplo: carruseles, dropdown, botones, thumbnails, entre otros.

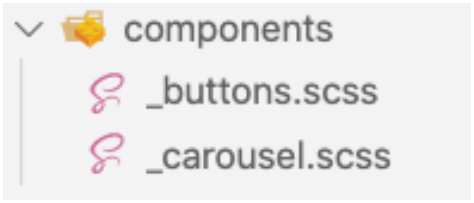

Imagen 4. Directorio "components".

- **LAYOUT**
  Contiene todo lo relacionado con el layout o diseño de la interfaz. Incluye elementos como: header, footer, sidebar, navegación y grillas.

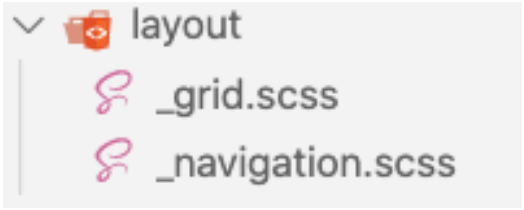

Imagen 5. Directorio "layout".

- **PAGES**
  Se utiliza en el caso de tener distintos tipos de diseño en las páginas. De no ser así, se puede dejar vacío.

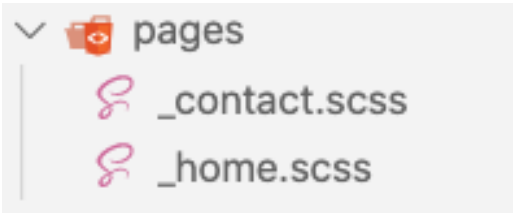

Imagen 6. Directorio "pages".

- **THEMES**

Se utiliza generalmente en proyectos grandes, donde existen diferentes formas de definir un tema dependiendo de factores como el tipo de usuario o el lugar geográfico.

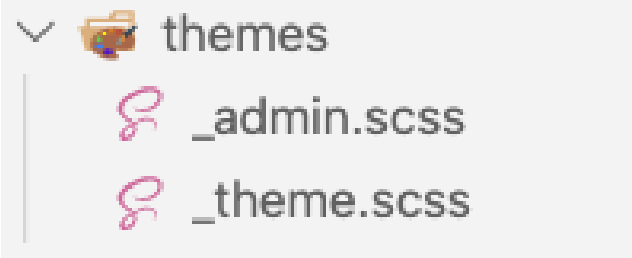

Imagen 7. Directorio "themes".

- **VENDOR**
  En éste se encontrarán todos los archivos CSS de bibliotecas, plugins y frameworks externos que se requiera utilizar, tales como: Normalize, Bootstrap, entre otros.

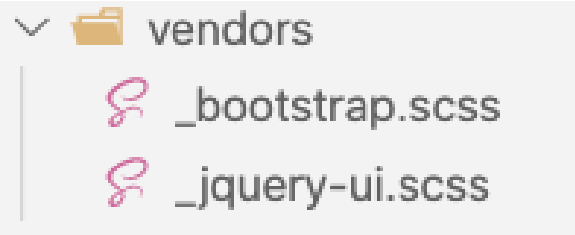

Imagen 8. Directorio "vendors".

- **MAIN.SCSS**
  Como ya se mencionó más arriba, este archivo corresponde al manifiesto de nuestro proyecto, el cual contendrá todo el código que compilará a CSS.

Para evitar errores de carga o cascada, es importante que los directorios se importen de la siguiente manera:

1. abstracts/
2. vendors/
3. base/
4. layout/
5. components/
6. pages/
7. themes/

Además, es importante saber que para mantener la legibilidad del manifiesto se seguirán las siguientes reglas:

- Un archivo por `@import`.
- Un `@import` por línea.
- No se debe dejar una línea después de importar archivos de la misma carpeta.
- La extensión del archivo y el guion bajo deben ser omitidos en la ruta.

Ahora, nuestro manifiesto debería lucir de la siguiente manera:

```scss
// main.scss # Manifiesto

@import 'abstracts/variables';
@import 'abstracts/functions';
@import 'abstracts/mixins';
@import 'abstracts/placeholders';

@import 'vendors/bootstrap';
@import 'vendors/jquery-ui';

@import 'base/reset';
@import 'base/typography';

@import 'layout/navigation';
@import 'layout/grid';
@import 'layout/header';
@import 'layout/footer';
@import 'layout/sidebar';
@import 'layout/forms';
```

```
20
21 @import 'components/buttons';
22 @import 'components/carousel';
23 @import 'components/cover';
24 @import 'components/dropdown';
25
26 @import 'pages/home';
27 @import 'pages/contact';
28
29 @import 'themes/theme';
30 @import 'themes/admin';
```