Introducción

- ¿Qué es SASS?
- ¿Por qué es útil?

¿Qué es SASS?

Sass significa “Syntactically Awesome Stylesheets”, es una herramienta escrita en Ruby que nos permite crear hojas de estilos estructuradas, limpias y fáciles de mantener.

Con SASS vamos a poder escribir hojas de estilo que nos ayudará a generar ficheros CSS más optimizados, incorporando mayor contenido semántico y permitiendo utilizar funcionalidades que normalmente encontraríamos en lenguajes de programación tradicionales, como el uso de variables, creación de funciones, etc.

¿Por qué es útil?

Normalmente crear una hoja de estilos es relativamente sencillo. Lo malo es cuando el proyecto va creciendo en tamaño: su CSS puede acabar siendo muy extenso.

Sass nos permite una sintaxis más simple, más elegante, implementando además bastantes características extras para hacer más manejable nuestra hoja de estilos.

Instalación

Para poder utilizar SASS necesitamos disponer del compilador que nos permita convertir nuestros ficheros en sintaxis SCSS a CSS, para instalar dicho compilador debemos instalar previamente Ruby.

Antes de comenzar con la instalación; vamos a explicar el motivo: Sass es un gem (joya) de Ruby. RubyGems es el gestor de paquetes de Ruby que provee un formato estándar para distribuir paquetes/librerías (Gems). Fue diseñado para manejar fácilmente la construcción, instalación y distribución de librerías.

Los usuarios de Windows podrán hacerlo fácilmente a través de http://rubyinstaller.org, los usuarios de Mac OS X ya lo llevan instalado de serie, y los usuarios de Linux podrán instalarlo con su gestor de paquetes preferido. Una vez instalado Ruby y su gestor de paquetes Ruby Gems, instalar Sass será tan sencillo como introducir esta línea en nuestra consola de comandos.

La instalación es muy sencilla:

1. Abrimos la **línea de comandos** (en Windows ejecutan el programa “cmd”).

2. Usamos RubyGems para la instalación, recuerden Ruby usa Gems para manejar la variedad de paquetes/librerías. Escribimos lo siguiente:

```
$ gem install sass
```

1. El comando instalará Sass y todas las dependencias requeridas. Si la línea de comandos imprime un mensaje de error, es probable que necesites usar el comando sudo para instalar Sass satisfactoriamente.

```
$ sudo gem install sass
```

Para comprobar que la instalación fue exitosa, copiamos y pegamos el siguiente comando:

```
$ sass -v
```

Nos imprimirá la versión de Sass: **Sass 3.4.7.**

Variables

- Sintaxis
- Crear variables

Sintaxis

En Sass contamos con dos diferentes tipos de sintaxis: **scss y sass**.

La primera y más popular es conocida como SCSS (Sassy CSS), es muy similar a la sintaxis nativa de CSS, tanto así que nos permite importar hojas de estilos CSS (copiar y pegar) directamente en un archivo SCSS y obtener un resultado válido.

Para utilizarla solo debemos crear un archivo con terminación .scss de la siguiente manera: **archivo.scss**

La segunda opción, es conocida como Indented Syntax (sintaxis deindentación). Utiliza la indentación en lugar de corchetes para expresar el anidamiento de selectores y saltos de línea en lugar de punto y coma para separar las diferentes propiedades que se declaren. Usarla también es muy sencillo, creamos un archivo con terminación .sass de la siguiente manera: **archivo.sass**

Crear variables

Las variables son una manera de guardar información que necesites reutilizar en tus hojas de estilos: colores, dimensiones, fuentes o cualquier otro valor. Sass utiliza el símbolo dólar ($) al principio de la palabra clave para crear una variable.

Estas variables se comportan como atributos CSS, y su valor puede ser cualquier valor que pudiera adquirir cualquier atributo CSS.

```
// creando variable
$color: #FF0000;

body {
  // aplicando el valor de $color
  background-color: $color;
}
```

Una variable se podrá definir fuera o dentro de algún selector. Si se define fuera, dicha variable será global y podrá utilizarse en cualquier bloque, pero si se define dentro de un selector, la variable será local y únicamente se podrá utilizar en el selector que la contiene y en sus selectores anidados.

Una buena práctica común consiste en definir todas las variables globales al principio del fichero, para que puedan localizarse rápidamente.

Uso de !default en las variables

Si en el ejemplo anterior, hacemos:

```
$color: #FF0000;
$color: #000000;
```

El color que se usará es el `#000000`. Pero si hacemos:

```
$color: #333333;
$color: #000000 !default;
```

El color que se usará será `#333333`. Ya que esta directiva indicará que la asignación que estamos realizando a la variable solo se haga en caso de que dicha variable no se haya definido anteriormente.

Usos de anidación

La principal caraterística de Sass es poder definir reglas de maquetación de forma anidada, evitando que tengamos que repetir constantemente los prefijos de alcance en los selectores CSS. Uno de los problemas de los selectores CSS es que cuanto más específicos sean estos, más tendremos que repetir una y otra vez la cadena de elementos que conforman el selector.

Ejemplo:

```
#content            { border: 1px solid black; }
#content p.info     { color: #fff; }
#content p.info a   { text-decoration: none; }
```

Como se puede ver en el ejemplo, tenemos que ir repitiendo los elementos base del selector según vamos estilizando los elementos más internos. Esto trae algunos problemas.

1. El documento CSS se hace poco legible.
2. Tenemos que hacer un uso excesivo de elementos repetitivos y por consiguiente copy/paste, lo cual induce a errores.
   Sass evita estos inconvenientes ofreciéndonos la posibilidad de anidar selectores unos dentro de otros.

El ejemplo anterior, se haría de la siguiente manera:

```
#content {
  border: 1px solid black;
  p.info {
    color: #fff;
    a { text-decoration:none;}
  }
}
```

De esta manera, no tenemos que repetir las cadenas de selección completas pues Sass se encargará de introducirlas cuando lo compilemos a CSS.

Además de anidar selectores también podremos anidar propiedades de forma que no tengamos que repetir constantemente cosas como `border-left`. Podemos verlo en este ejemplo:

```
.boxborder {
  border: {
    style: solid;
    left: {
      width: 4px;
      color: #888;
    }
    right: {
      width: 2px;
      color: #ccc;
    }
  }
```

Importar

El uso de @import es diferente en Sass que en CSS. En una hoja de estilos CSS supone una nueva llamada al servidor para cargar otra hoja de estilos y esperar a que se cargue para aplicar los nuevos estilos.

En Sass es diferente. La importación en un archivo .scss o .sass se produce durante la compilación.

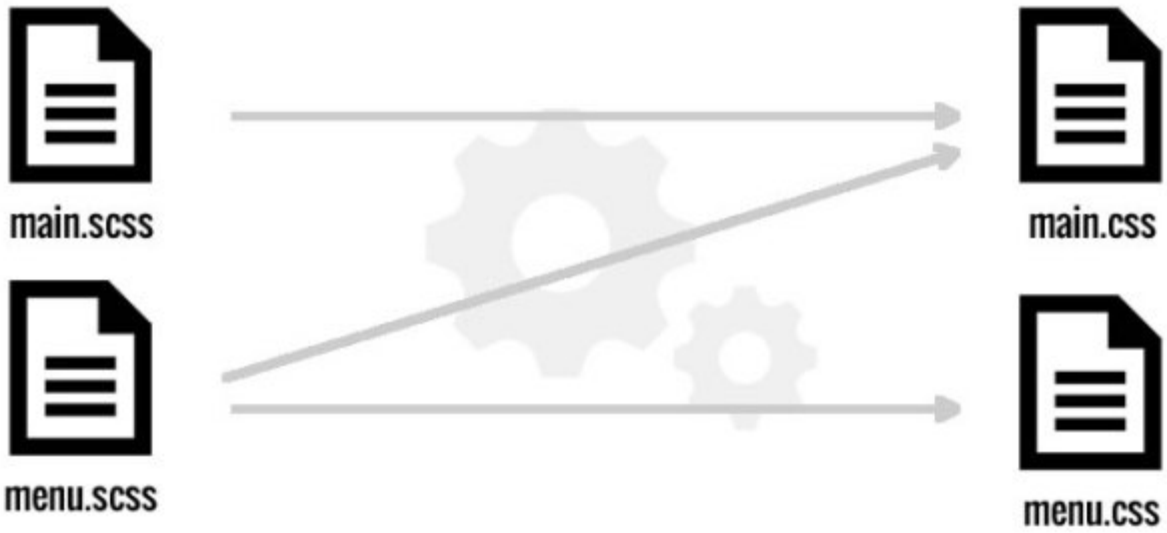
Vamos a crear un nuevo archivo menu.scss que sea por ejemplo así:

```
.menu {
  margin: 0;
  padding: 0;
  list-style-type: none;
}
.menu > li {
  display: inline-block;
  margin: 0 0 10px 10px;
}
```

Y en main.css ponemos:

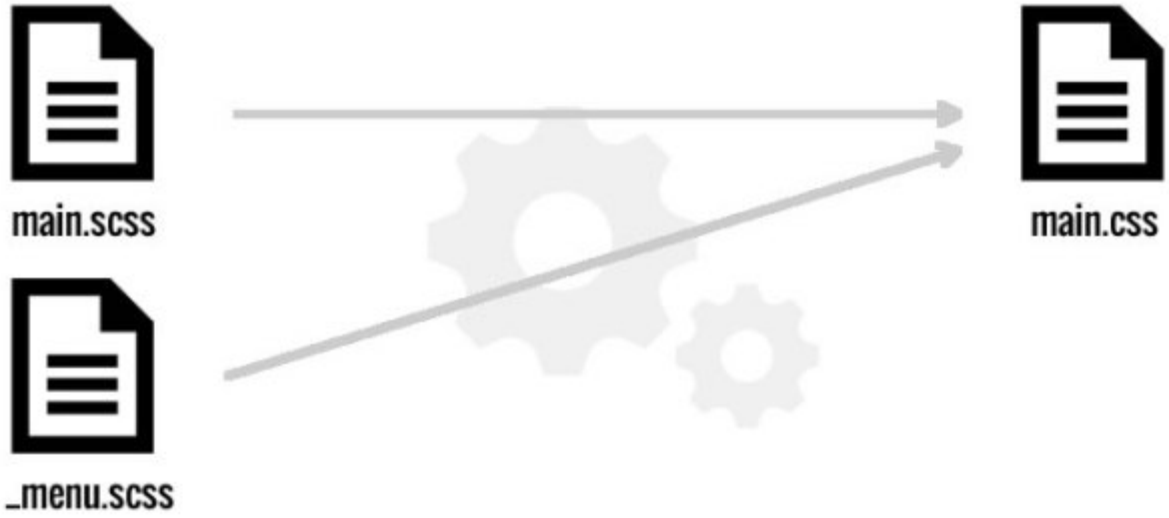
```
// Importamos los estilos de menu.scss
// cuando se compile main.scss
@import "menu";
```

Esto es lo que ocurre:



En el CSS resultante main.css estarán compilados tanto el contenido de main.scss como el de menu.scss. Pero como vemos también se creará una hoja de estilos independiente menu.css sólo con los estilos de menu.scss.

Para evitar esto debemos grabar el archivo menu.scss con un guión inferior previo, es decir _menu.scss. Es lo que se llama un “partial” (parcial).



La sintaxis dentro del archivo .scss sigue siendo la misma:

```
// Importamos los estilos de _menu.scss
// cuando se compile estilos.scss
@import "menu";
```

De esta forma los estilos de _menu.scss pasarán a formar parte una vez compilado de main.css sin crear una hoja de estilos aparte. Por regla general, un archivo como main.scss debería incluir únicamente importaciones de diferentes archivos scss (por ejemplo, _reset.scss, _menu.scss, _header.scss, etc..) sin tener estilos propios.

Mixins

Nos permiten definir estilos que puedan ser reutilizados en nuestro proyecto, sin necesidad de recurrir a las clases helpers que ya comentamos, también pueden contener complejas reglas de CSS.

Cómo usar los Mixins en Sass:

Para crearlos debemos utilizar `@` seguido de `mixin` y por último el nombre que queramos darle al mixin. Luego, para declararlo dentro de un componente, usamos `@include`, seguido de un espacio, el nombre y por último entre paréntesis `()` escribimos los argumentos requeridos por nuestro Mixin. Los argumentos son opcionales.

Por ejemplo, para evitar la repetición de alto y ancho de un componente a lo largo de un proyecto, podemos crear un pequeño Mixin que transforme las habituales dos líneas de código en una sola:

```
@mixin sizes($width, $height: $width) {
  height: $height;
  width: $width;
}

.box {
  @include sizes(100px);
}
```

Es necesario recordar que los Mixins deben ser declarados en la última línea de nuestros componente.

Extends / Placeholders:

Es una de las más poderosas características de Sass. Nos permite crear un fragmento de estilos que luego podamos reutilizar fácilmente en cualquier componente.

Cómo usarlo:

Para declarar y crear un fragmento de estilos que necesitamos reutilizar, usamos `%` seguido de un nombre, sin espacios. Luego, para imprimir o reutilizar el Extend dentro de un componente, usamos `@extend` seguido de un espacio y por último, el nombre que asignamos a nuestro fragmento.

Por ejemplo, si queremos crear un componente básico para imprimir mensajes del sistema, debemos tener una clase para mensajes exitosos, de advertencia y de errores.

```
// vars
$bg-success: green;
$bg-error: red;
$bg-warning: orange;

// extend
%message {
  border-radius: 2px;
  color: white;
  padding: 5px;
  text-align: center;
}

.message-success {
  @extend %message;
  background-color: $bg-success;
}

.message-error {
  @extend %message;
  background-color: $bg-error;
}

.message-warning {
  @extend %message;
  background-color: $bg-warning;
```

Sass nos imprimirá el siguiente CSS:

```
.message-success, .message-error, .message-warning {
  border-radius: 2px;
  color: white;
  padding: 5px;
  text-align: center;
}

.message-success {
  background-color: green;
}

.message-error {
  background-color: red;
}

.message-warning {
  background-color: orange;
}
```

Sass, agrupará las clases o atributos que tengan estilos comunes, evitando repetirlos por separado.

Operadores:

Realizar operaciones matemáticas en CSS, es posible gracias a Sass. Contamos con un puñado de operadores: (+, -, *, / y %) que trabajan de la misma forma que las operaciones matemáticas básicas.

Por ejemplo, para crear un simple grid basado en 960px y que el ancho de cada elemento sea expresado en porcentajes:

```
$wrap: 960px;

article[role="main"] {
  float: left;
  width: 630px / $wrap * 100%;
}

aside[role="complimentary"] {
  float: left;
  width: 330px / $wrap * 100%;
}
```

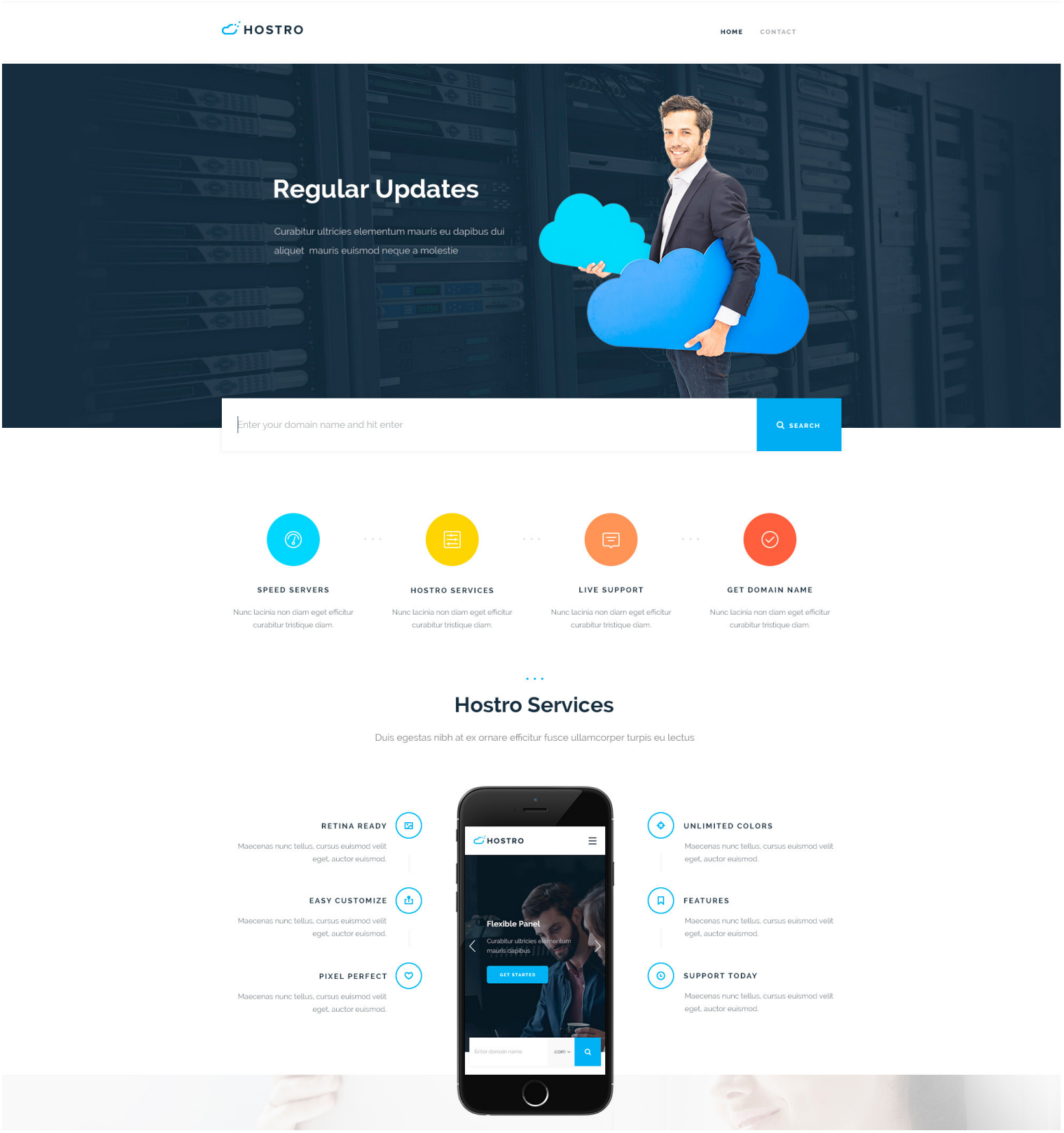
Sass nos imprimirá el siguiente CSS:

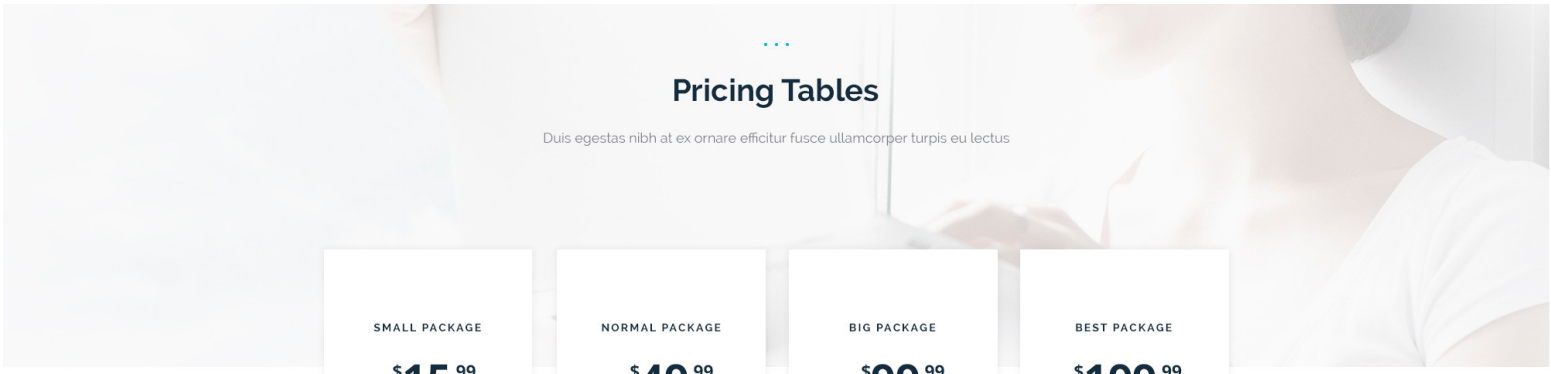
```
article[role="main"] {
  float: left;
  width: 65.625%;
}

aside[role="complimentary"] {
  float: left;
  width: 34.375%;
}
```

Trabajo Práctico 8 SASS

Descripción:

Realizar nuestra web en el lenguaje sass aplicado también a media queries.



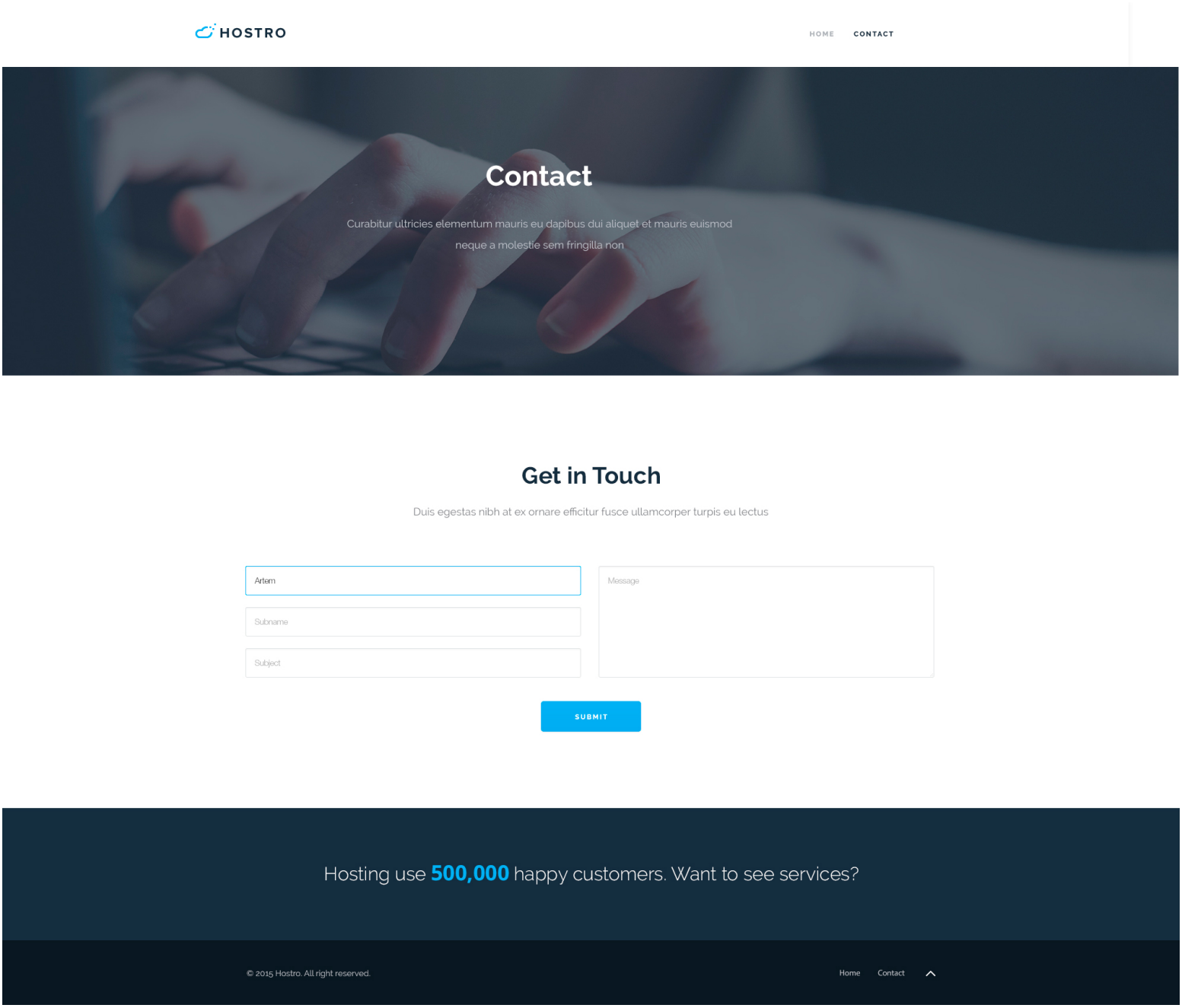


Pricing Tables

Duis egestas nibh at ex ornare efficitur fusce ullamcorper turpis eu lectus

SMALL PACKAGE	NORMAL PACKAGE	BIG PACKAGE	BEST PACKAGE
$15.99 /mount	$49.99 /mount	$99.99 /mount	$199.99 /mount
Hard drive, GB 2	Hard drive, GB 5	Hard drive, GB 10	Hard drive, GB 15
Transfer, GB 1	Transfer, GB 2	Transfer, GB 3	Transfer, GB 4
Data bases 1	Data bases 2	Data bases 3	Data bases 4
Dashboards 1	Dashboards 2	Dashboards 3	Dashboards 4
SELECT PLAN	SELECT PLAN	SELECT PLAN	SELECT PLAN

Hosting use **500,000** happy customers. Want to see services?



Objetivo:

Utilizar el lenguaje sass con mayor facilidad aplicándolo en diferentes modelos de web.

Requerimientos:

1. Haber realizado los trabajos anteriores
2. Tener conocimientos en lenguaje SASS.
3. Realizar la conversión de archivos con la terminal/cmd.
4. Utilizar al menos 3 variables y crearlas dentro de otro archivo.
5. Subir la resolución de tu trabajo a tu repositorio de Github en la subcarpeta “SASS TP8”.

Bonus:

Less: lenguaje dinámico de hojas de estilo!

Recursos:

Less (bonus)
Sass