

Flujo de trabajo (Parte II)

Creando estilos base del proyecto

Continuemos con la creación de estilos para nuestro proyecto. Esta vez conoceremos los pasos para crear los estilos bases para nuestro proyecto, esto significa que le daremos algunas reglas de estilos iniciales a modo de crear la plantilla base de nuestra maqueta.

Ingresemos al directorio base y luego presionemos el parcial `_reset.scss`

Reset

Las reglas reset son usadas para reestablecer los valores CSS que vienen por defecto en los navegadores. Crear estas reglas serán nos serán útiles para comenzar a crear nuestros de manera homogénea.

Asimismo es importante saber que personas con muy inteligentes ya crearon algunos estilos que nos ayudarán a resetear y normalizar nuestros estilos a través de los navegadores.

Existen dos estilos muy usados para formatear nuestros elementos de HTML. El primero fue creado por se llama CSS reset el cual es una plantilla creada por Eric Meyer que nos permitirá eliminar todo elemento incorporado al HTML. El segundo llamado Normalize.css fue creado por Nicolas Gallagher y nos ayudará a eliminar las inconsistencias que hay en el HTML, manteniendo algunas reglas útiles.

Cualquiera de las dos son útiles para resetear nuestros HTML y es bueno saber que las tenemos a disposición para mejorar estos aspectos, pero ahora nosotros crearemos algunas reglas reset que nos serán útiles para este proyecto tan pequeño.

Agreguemos dentro del parcial `_reset.scss` las siguientes reglas:

Primero hagamos un reset del body reseteando el margen a 0.

```
/* Reset*/
body {
    margin: 0;
}
```

Y luego, prevengamos que la imagen sobrepase el 100% de ancho.

```
img {  
  max-width: 100%;  
}
```

Con esto tendremos los reset suficientes para trabajar en nuestro proyecto.

Base

Ahora sigamos el archivo `base.scss`.

Este archivo nos será útil para agregar los archivos bases a elementos de nuestro HTML. En nuestro solamente deberemos agregar el fondo de maqueta, de modo que deberíamos:

- Primero definir el tamaño del body a un 100% de su ancho, para eso podremo usar un valor llamado viewport width.
- Agregar un `background-color` específico por si no carga la imagen. Este le daremos un valor de blanco, utilizando las variables de color.
- Luego agregaremos un `background-image` que contendrá la url relativa de la imagen. En este caso esta se encuentra en `../images/background.png`.
- Para que no se repita la imagen usaremos un `background-repeat: no-repeat;`.
- Para que el fondo cubra todo el tamaño de body usaremos el valor `cover` para conseguirlo.
- Finalmente para posicionar el fondo usaremos la propiedad `background-position` con un valor de un 80%.

```
/* Base Styles */  
  
body {  
  height: 100vh;  
  background-color: $white;  
  background-image: url("../images/background.png");  
  background-repeat: no-repeat;  
  background-size: cover;  
  background-position: 80%;  
}
```

Typography

Para finalizar fijemos algunos parámetros en los títulos, ya que el diseño no contiene más elementos tipográficos.

para los títulos desde h1, hasta h6 usarán la familia tipográfica "Open sans", para ello usaremos la variable `$open-sans` como valor.

Además, agregaremos un grosor regular a la fuente, por lo tanto esta tendrá como valor la variable `$regular`.

Finalmente definamos el tamaño base de los títulos. En este caso usaremos la variable `$medium` como tamaño.

```
/* Base Typography*/
h1, h2, h3, h4, h5, h6 {
  font-family: $open-sans;
  font-weight: $regular;
  font-size: $medium;
}
```

Si recargamos el sitio veremos que los cambios que hemos realizado han funcionado.

Como vimos este paso es importante para formatear los estilos de nuestro proyecto antes de comenzar a trabajar directamente con el layout y sus componentes.

Creando estilos del layout del proyecto

Comencemos a dar los estilos que componen el layout de la maqueta. Como sabemos este layout está compuesto de tres elementos que definen el diseño: Primero el header, luego el main y finalmente el formulario.

Creando los estilos del header

Para crear los estilos del header necesitaremos hacer dos cosas: Primero debemos revisar la representación visual para identificar los espacios que tiene el diseño y el logo que lo contiene. Después deberemos aplicar algunos estilos CSS para definir los estilos de este layout.

Veamos el mockup para ver sus dimensiones:

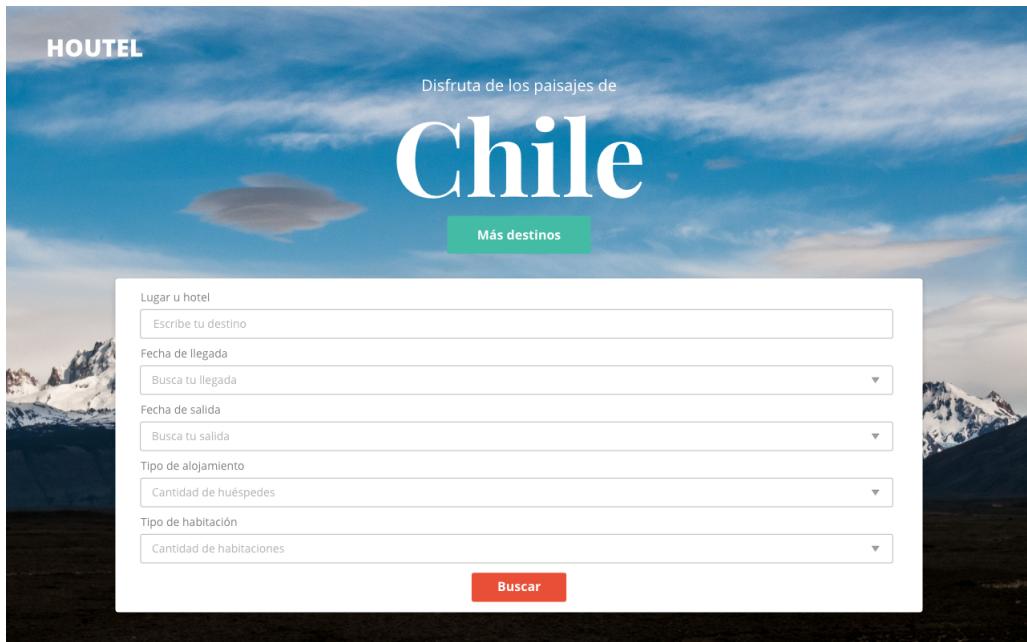


Imagen 1. Mockup.

- En logo de la sección Header, se puede apreciar un margen top y left, esto nos permite generar una separación desde el viewport.
- En cuanto al logo el tamaño lo mediremos mientras estemos

Con esto listo apliquemos los estilos pertinentes al layout del header.

- Agreguemos la clase `.header` y agreguemos un `padding-top` y `padding-left` para dar los espacios pertinentes al header
- Ahora agreguemos un tamaño correcto al logo. Para este caso un tamaño que quedará bien son `120px`

```
/* Header Layout */
.header {
    padding-top: 2rem;
    padding-left: 2rem;
}

.logo {
    width: 120px;
}
```

Recarguemos la página para ver los nuevos estilos.

Creando los estilos de main

Ahora sigamos con los estilos del contenido principal.

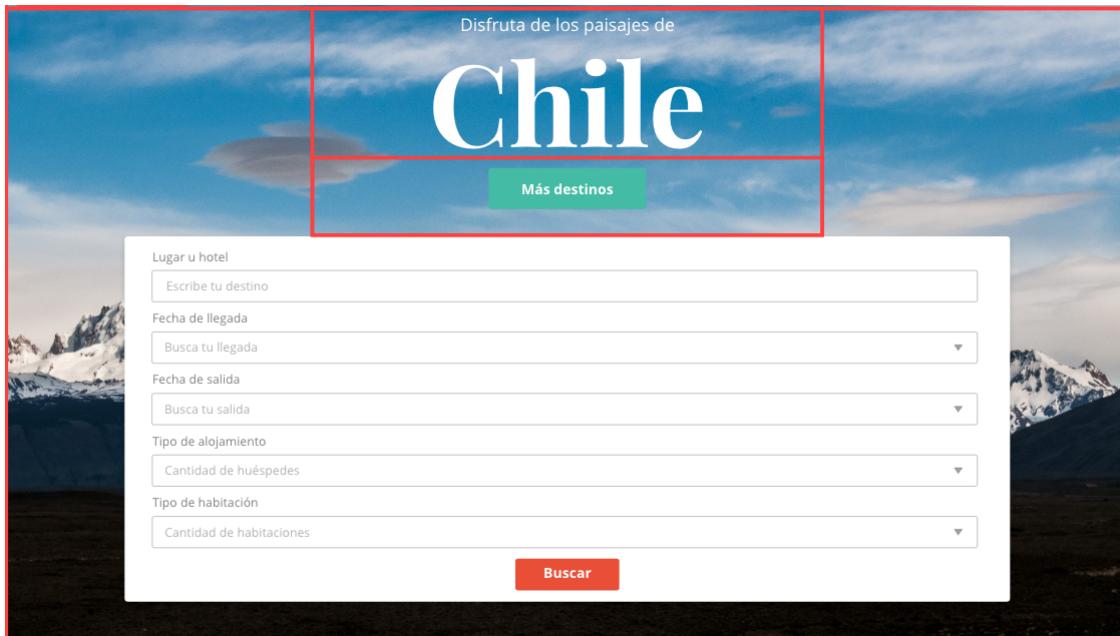


Imagen 2. Estilos del main.

Main: En este bloque principal posicionaremos el contenido usando un valor absoluto en el layout a modo de hacer más fácil el movimiento del contenido interior como el título y el botón.

Main container: En cuanto al contenedor de los elementos ajustaremos su ancho a `200px` y posteriormente ordenaremos y definiremos el layout usando `display:flex` el cual nos permitirá trabajar con el diseño de este contenedor.

Main title: El título aún no le hemos definido el color, de modo que lo haremos usando la variable de color llamada `$white`, además la centraremos usando `text-align: center;`.

Main title emphasis: Para el énfasis de la palabra Chile usaremos como estilos base el tipo de fuente que usará, el cuál será la la variable `$playfair`. Además, este tendrá un grosor de letra `$bold`.

Main title emphasis size xxxl: Para el tamaño de la palabra usaremos la variable de tamaño creada anteriormente llamada `$xxxx-large`.

Main button: Finalmente, al contenedor del botón le daremos un margen negativo de `12px` para mover hacia arriba un poco el botón.

```

.main {
  position: absolute;
  left: 50%;
  top: 9.5vh;
  transform: translateX(-50%);
}

.main__container {
  width: 200px;
  display: flex;
  flex-direction: column;
  align-items: center;
}

.main__title {
  color: $white;
  text-align: center;
}

.main__title-emphasis {
  font-family: $playfair;
  font-weight: $bold;
}

.main__title-emphasis_size-xxl {
  font-size: $xxxx-large;
}

.main__button {
  margin: -20px;
}

```

Refresquemos la página el navegador para ver los estilos de main.

Creando los estilos del formulario

Los últimos estilos de layout que definiremos serán los del formulario.

Lugar u hotel
Escribe tu destino

Fecha de llegada
Busca tu llegada ▾

Fecha de salida
Busca tu salida ▾

Tipo de alojamiento
Cantidad de huéspedes ▾

Tipo de habitación
Cantidad de habitaciones ▾

Buscar

Imagen 3. Estilos de layout del formulario.

Form: Comencemos con el bloque principal de este layout. Lo que haremos acá será posicionar el elemento usando un `position: absolute;`. Luego centraremos este elemento usando un dos propiedades, la primera es `left`, la cual nos ayudará a mover el elemento hacia la derecha en un `50%`, y el segundo llamado transform que fijará totalmente el elemento en el centro. Para bajar el bloque usaremos una propiedad `top` con un valor de `45vh`.

Ahora le daremos un tamaño al contenedor. De alto tendrá `370px` y `70vw` de ancho. Asimismo, mantegamos en margen del contenedor a `0` agreguemos un poco de espacio dentro del bloque a modo de posicionar mejor los elementos del formulario usando un `padding: .5rem 1rem;`.

En cuánto a los bordes del bloque usaremos un `border-radius: 3px;` y su color lo ajustaremos con `background-color: $white;`.

Form container: Ahora organizaremos el contenido del contenedor del formulario usando `display: flex;` y ajustando su contenido a columnas.

Form button: Finalmente en el contenedor del botón usaremos `padding-top: .5rem;` para dar un poco de espacio entre el botón y el último input. Además de lo anterior, centraremos el botón usando `text-align: center;`.

```
/* Form Layout */
.form {
  position: absolute;
  left: 50%;
  transform: translateX(-50%);
  top: 45vh;
  height: 370px;
  width: 70vw;
  margin-left: 0;
  padding: .5rem 1rem;
  border-radius: 3px;
  background-color: $white;
}

.form__container {
  display: flex;
  flex-direction: column;
}

.form__button {
  padding-top: .5rem;
  text-align: center;
}
```

- `display: flex`: es una propiedad css para alinear elementos, se verá con más detalle más adelante.

Al finalizar de agregar todos estos estilos tendremos un resultado que casi roza al diseño presentado en el mockup.



Imagen 4. Todos los estilos agregados.

Creando los estilos de componentes del proyecto

Los últimos estilos que debemos agregar en nuestra maqueta son los de los componentes. Estos se encuentran separados en dos parciales: los botones y los inputs.

Comencemos con los botones:

Creando los estilos de los botones

Los botones que contiene nuestra maqueta tienen en primer lugar los estilos bases basados que estarán en la clase `.button` y los colores que dependerán de su uso, siendo la clase `.button_default` el color por default y `.button_positive` el color alternativo.

Comencemos a dar los estilos a los botones:

Button: Primero definamos los estilos base del botón. Estos tendrán un espacio en el interior de `.7rem` de alto y de ancho `2rem`.

Luego redondearemos el botón usando un `border-radius: 3px;`.

El botón además tendrá una familia de fuentes `$open-sans`, un tamaño pequeño de fuente (`$small`) y un color blanco(`$white`).

```
/* Buttons */  
.button {  
    padding: .7rem 2rem;  
    border-radius: 3px;  
    font-family: $open-sans;  
    font-size: $small;  
    color: $white;  
}
```

Button default: El primer esquema de color del botón tendrá como color de fondo el color `$carmine-pink` visto en la guía de estilos y para que el botón sea totalmente de este color le daremos usaremos esta misma variable pero esta vez al borde.

Ahora, para mejorar la usabilidad del botón usaremos las difuminaciones del color vistas en la guía de estilos, haciendo que cambie de color el botón cuando el usuario pase el cursor por arriba de el.

Para hacer primero deberemos conocer lo básico de los pseudo clases.

¿Qué son las pseudo clases?

Las pseudo clases definen un estado específico de un elemento HTML.

Por ejemplo, cuando pasamos el cursor por arriba de un link este cambia de color. Este cambio de estado se conoce como pseudo clase. En este caso la pseudo clase que se activa se llama `:hover`.

Para usarla debemos escribir el nombre del elemento a afectar y seguido de este deberemos agregar el pseudo selector.

En el caso del link sería:

```
a:hover {  
  color: peru;  
}
```

Para aplicar estos psudo selectores en los botones usaremos una de las funciones más útiles de Sass llamada nesting.

¿Qué son los nesting?

Los nesting nos permiten anidar los selectores CSS de manera que sigan una jerarquía visual similar a HTML.

Asimismo ayudan a prevenir conflictos de especificidad entre reglas CSS evitando que se cancelen entre sí.

Estos funcionan de la siguiente manera:

Imaginemos que tenemos un `div` con clase `.hero__title` el cuál tiene un `span`.

Si quisieramos dar estilos al `span` sin necesidad de escribir otro selector podríamos usar los nestings de Sass para agregar estos estilos.

```
.hero-title {  
  display: flex;  
  flex-direction: column;  
  color: $primary-color;  
  text-align: center;  
  font-family: $primary-font-family;  
  span {  
    font-size: $font-size-large;  
    font-family: $secondary-font;  
    font-weight: bold;  
  }  
}
```

Si nos fijamos la forma en la cual se hacen las anidaciones es muy similar a HTML, de modo que crearlas es muy intuitivo si ya hemos trabajado con este sistema de marcas.

A pesar de que los nesting son una gran solución debemos tener en cuenta que existen algunas consideraciones en su uso:

La sobre anidación de elementos usando nesting puede ocasionar problemas de no funcionamiento de algunos elementos, además hacer que el código sea caótico y desordenado.

La solución a este problema es usar una cantidad máxima de anidaciones no superior a las tres anidaciones. Esta regla se conoce como "*la regla de no más de 3*".

Otro cosa a tener en cuenta sobre el uso de nesting es que cuando usamos BEM no es necesario el uso de las anidaciones debido a que la estructura de bloques y elementos usada por BEM separa los estilos de tal manera que estos nunca tendrán conflictos debido a su estructura modular. Esto sumado al sistema de clases hace que el uso de nesting con BEM termine siendo un problema más que una solución.

Sin embargo existe una utilidad de los nesting que podremos utilizar en nuestro proyecto aún cuando usemos BEM.

El anpersand de Sass

Esta característica usa el ampersand (&) para poder crear selectores más específicos usando clases, pseudo clase y pseudo elementos.

Lo genial de esta funcionalidad es que podremos aprovecharnos de ella para cambiar el color de los botones de nuestro proyecto usando pseudo elementos.

¿Cómo cambiar el estado de los botones?

Para hacerlo usaremos el & y luego de este pondremos la pseudo clase :hover . Dentro de este selector agregaremos la primera difuminación del color \$carmine-pink . Usaremos las mismas reglas usadas en la clase .button_default .

```
&:hover {  
    background-color: $salmon;  
    border-color: $salmon;  
    color: $black-olive;  
}
```

Provemos si funcionó el hover del botón.

Button positive: Ahora sigamos con el botón positive. Este tendrá la misma estructura de reglas css que el botón anterior, lo que cabiará en estos botones será el color el cuál será para el botón el color \$verdigris y para el :hover el \$medium-aquamarine .

```
.button_positive {  
    background-color: $verdigris;  
    border-color: $verdigris;  
  
&:hover {  
    background-color: $medium-aquamarine;  
    border-color: $medium-aquamarine;  
}
```

Creando los estilos de los inputs

Luego de terminar de crear los estilos de los botones es momento de darle estilo a los inputs.

Los inputs están divididos por el label con clase .form_label , los inputs de texto y calendario y finalmente el select del formulario.

Comencemos a darle estilo al label del formulario.

Primero demos al label algo de margen arriba. Usemos un valor de .5rem . También agreguemos un poco de espacio abajo. Con .3rem es suficiente.

```
/* Inputs */
.form__label {
  margin-top: .5rem;
  padding-bottom: .3rem;
  font-family: $open-sans;
  font-size: $x-small;
  color: $granite-gray;
}
```

Ahora configuremos su familia de fuentes, usemos `$open-sans`. Damos un tamaño mínimo de letra (`$x-small`) y finalicemos con un color de fuente `$granite-gray`.

Ahora, para ahorrar líneas de código utilizaremos una tipo de selector muy útil cuando existen elementos que tendrán las mismas reglas CSS.

En nuestro caso `.form__input-text`, `.form__input-date` y `.form__select` tendrán algunas reglas similares, de modo que usaremos el selector `,` para darle los mismos estilos a estas tres clases.

Escribamos las clases seguidas por coma y cuando estén estas tres abriremos llaves e incluiremos:

Un ancho de `30px`, un `padding-left:5px`, un tamaño de fuente `$small` y un color de fuente `$old-silver`.

```
.form__input-text, .form__input-date, .form__select {
  height: 30px;
  padding-left: 5px;
  font-size: $small;
  color: $old-silver;
}
```

Haremos lo mismo con `.form__input-text` y `.form__input-date`.

Redondeemos los inputs con un `border-radius: 5px;` y luego ajustemos el borde con un `border: 1px solid $gray;`

```
.form__input-text, .form__input-date {
  border-radius: 5px;
  border: 1px solid $gray;
}
```

Finalmente para homogeneizar el color de select le daremos un color de fondo blanco.

```
.form__select {
  background-color: $white;
}
```

Revisemos el resultado final de nuestra maqueta:



Imagen 5. Maqueta final.

Como vemos el resultado de nuestra maqueta es similar a lo visto en la representación visual.

Creando Mixins

Los mixins son una de las características más utilizadas dentro de todo el lenguaje Sass. Son la clave para la reutilización y los componentes DRY (Don't repeat yourself). Y por una buena razón: los mixins permiten a los maquetadores definir estilos CSS que pueden volver a usarse a lo largo de toda la hoja de estilo sin necesidad de recurrir a las clases no semánticas, como por ejemplo .float-left.

Pueden contener reglas CSS completas y casi todo lo que se permite en cualquier parte de un documento Sass. Incluso pueden pasarse argumentos, al igual que en las funciones. Sobra decir que las posibilidades son infinitas.

Se debe advertir el abuso del poder de los mixins. De nuevo, la clave aquí es la simplicidad. Puede ser tentador construir mixins extremadamente poderosos y con grandes cantidades de lógica. Esto se llama exceso de ingeniería. No pienses demasiado tu código y sobre todo haz que sea simple. Si un mixin ocupa mas o menos unas 20 líneas, debes dividirlo en partes más pequeñas o revisarlo completamente.

Los mixins son extremadamente útiles y deberían estar usando algunos. La regla de oro es que si se detecta un grupo de propiedades CSS que están siempre juntas por alguna razón (es decir, no es una coincidencia), puedes crear un mixin en su lugar. Ejemplo válido sería un mixin para darle un color, tamaño y peso a un texto, pero debe tener un background color distinto de acuerdo a la sección en la que se encuentre. No solo hace que el código sea más fácil de escribir, sino que también será más fácil de leer.

```
@ mixin important-text {
  color: red;
  font-size: 25px;
  font-weight: bold;
}

.danger {
  @include: important-text;
  background-color: green;
}
```

```
.black {  
  @include: important-text;  
  background-color: black;  
}
```

Autoprefixer

Como su nombre indica, Autoprefixer nos coloca los prefijos de los navegadores necesarios en el CSS final. Le podemos indicar los navegadores a los que dar soporte, bien por número de versiones anteriores o por porcentaje de uso.

Autoprefixer en línea: respuesta web para Autoprefixer original. Analiza su CSS y agrega prefijos de proveedor a las reglas de CSS utilizando valores de Can I Use. Es recomendado por Google y utilizado por Twitter y Taobao.

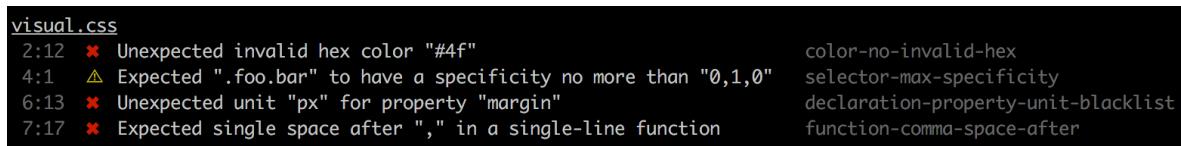
El Autoprefixer utiliza datos sobre la popularidad de los navegadores y la compatibilidad con los prefijos de proveedores de los navegadores. En base a esta información, organiza y elimina los prefijos. Puede ayudar a obtener prefijos para: animaciones, transición, transformación, cuadrícula, flex, flexbox y otros.

<https://autoprefixer.github.io/>

- Otras herramientas en el editor de texto:

StyleLint es un linter que nos muestra errores y nos hace escribir nuestro CSS de manera correcta. Una poderosa y moderna interfaz que te ayuda a evitar errores y hacer cumplir las convenciones en tus estilos.

- Este linter nos permite comprender la sintaxis CSS más reciente, incluidas las propiedades personalizadas y los selectores.
- Extrae estilos incrustados de HTML, markdown y literales CSS-in-JS de objetos y plantillas
- Analiza sintaxis de tipo CSS como SCSS, Sass, Less y SugarSS
- Tiene más de 170 reglas integradas para detectar errores, aplicar límites y aplicar convenciones estilísticas



A screenshot of the StyleLint interface showing errors in a CSS file named `visual.css`. The errors are listed with their line numbers and descriptions, along with the corresponding CSS code and the rule that triggered the error. The errors are:

Line	Description	Rule
2:12	Unexpected invalid hex color "#4f"	color-no-invalid-hex
4:1	Expected ".foo.bar" to have a specificity no more than "0,1,0"	selector-max-specificity
6:13	Unexpected unit "px" for property "margin"	declaration-property-unit-blacklist
7:17	Expected single space after "," in a single-line function	function-comma-space-after

Imagen 6. StyleLint, linter que muestra errores.

Autoprefixer

Como instalar Settings → Install → Search for autoprefixer

<https://github.com/sindresorhus/atom-autoprefixer>

- Abra la Paleta de comandos y escriba Autoprefixer: Prefix.
- En un archivo HTML, seleccione el CSS, abra la Paleta de comandos y escriba Autoprefixer: Prefix.

Bootstrap Sass

Ahora que finalizamos la maqueta y vimos que los resultados obtenidos son realmente fantásticos usando todas las herramientas, métodos y técnicas vistas a lo largo de este capítulo. En este capítulo revisaremos cómo integrar y modificar Bootstrap para utilizarlo en nuestro proyecto.

Lo primero que conoceremos será cómo integrar este framework CSS a nuestro proyecto.

Para hacerlo:

Descargando Bootstrap Sass

Debemos ir a getbootstrap.com. Luego debemos presionar el botón `Download`. Búsquemos la sección "Source files" y descarguemos esta versión.

Esperemos a que descargue. Cuándo este listo, descomprimamos el archivo y luego ingresemos. Aquí veremos diversos archivos y directorios de los cuáles sólo utilizaremos el directorio SCSS.

Ingresemos para ver su estructura:

Lo primero que veremos al entrar es el hecho que los parciales se encuentran esparcidos en la raíz de la carpeta SCSS, así como también el manifiesto y dos directorios que contiene las clases utilitarias de Bootstrap y los mixin usados para reutilizar reglas CSS.

Los parciales que vemos contienen diversas tipos de estilos CSS. Por ejemplo podremos encontrar acá componentes, grillas, reset, entre otros elementos que componen Bootstrap.

Integrando Bootstrap al proyecto

Ahora que conocemos que contiene este directorio lo integraremos a la maqueta que hicimos hacer poco a modo de entender el proceso que conlleva agregarlo en la estructura de directorio que tenemos construída.

Para hacerlo simplemente debemos copiar el directorio SCSS de Bootstrap y luego pegarlo dentro de la carpeta `vendors` de nuestro proyecto.

Cuándo terminemos este paso el siguiente será cambiar el nombre SCSS por alguno que nos haga sentido en este caso podría ser `bootstrap_v.4.1`.

Luego de hacer este cambio de nombre deberemos importar el manifiesto de bootstrap hacia nuestro manifiesto, o sea que en el manifiesto deberemos usar `@import` para que bootstrap funcione.

El procedimiento es sencillo. Primero ingresemos a `main.scss` y en el lugar que dejamos el comentario `//vendors` agregaremos el `@import` de manifiesto de bootstrap.

Escribamos `@import` y luego agreguemos la ruta específica del manifiesto.

```
// Vendors
@import 'vendors/bootstrap_v4.1/bootstrap';
```

Cuando se encuentre listo, guardemos los cambios y agreguemos algún componente de Bootstrap para probar si funcionó la integración.

Ingresemos a la sección componentes de bootstrap y búsquemos los botones.

Copiamos su código y peguemos este en el index.html de nuestro proyecto.

Recarguemos el navegador para ver los cambios.

Si funciona veremos que los botones incluidos se verán a todo color en la página.



Imagen 7. Botones incluidos en maqueta final.

Ahora que tenemos integrado Bootstrap en nuestro proyecto podremos usar todos los componentes y utilidades que incluye este framework, hasta incluso podremos personalizarlo.

Cómo personalizar Bootstrap

Para hacerlo deberemos ir a la carpeta de Bootstrap y luego ingresar al parcial variables.

Aquí encontraremos todas las variables usadas dentro del framework y lo único que deberemos hacer para modificar su valor usando el que nosotros deseemos.

Esto tiene un gran potencial, ya que cambiando solamente las variables podremos personalizar mucho como los colores, tamaños de fuentes, espaciados, entre otras cosas.

Hagamos una prueba de esto modificando el color de los botones por los colores de nuestros botones.

Vamos a parcial variables de nuestro proyecto, copiemos el color default peguemos este en la línea 73 del parcial variables de bootstrap.

```
$primary:      #E94F37 !default;  
$secondary:    $gray-600 !default;  
$success:     $green !default;  
$info:        $cyan !default;  
$warning:     $yellow !default;  
$danger:      $red !default;  
$light:       $gray-100 !default;  
$dark:        $gray-800 !default;
```

Si recargamos la página veremos que cambiará el color del botón.



Imagen 8. Cambio de colores de botones.

Con esto hemos terminado de conocer todas las herramientas que veremos en esta unidad.

Como vimos crear una maqueta siguiendo las mejores prácticas y usando las herramientas correctas hará que traducir la investigación y diseño de UX/UI sea fácil, rápido y organizado. Además con las técnicas y metodologías que usamos el trabajo posterior de los desarrolladores será más ágil y consistente.

Los invito a seguir practicando el uso de estas herramientas a modo de realizar de manera más fluída el proceso visto en la unidad.