

Flujo de trabajo

Introducción

Esta unidad entrega a los alumnos las bases para comenzar a maquetar usando guías de estilos y representaciones visuales entregadas por diseñadores UI y/o UX. Así mismo, se aprenderá a maquetar proyectos web de manera semántica usando HTML5, con hojas de estilos consistentes y mantenibles usando Sass, modularizando elementos del proyecto usando la metodología BEM, así como también usando Bootstrap Sass.

Al finalizar esta unidad los alumnos podrán crear un proyecto web desde cero o adaptar uno existente usando Sass o Bootstrap Sass.

El proceso de crear un sitio web

Antes de comenzar a profundizar en el proceso de maquetación necesitamos entender cómo funciona el flujo de trabajo utilizado para crear las interfaces de usuario que posteriormente traduciremos a código. Estas interfaces son el resultado de un proceso que investiga al usuario a modo de entregar la mejor experiencia a un visitante.

Crear la experiencia del usuario



Para crear una experiencia realmente útil para un usuario es necesario pasar por un proceso que va de los más abstracto a lo más concreto. Según a Garret en el libro "*Los elementos de la experiencia del usuario*", la experiencia de usuario se puede entender en 5 planos primarios. Estos planos representan el proceso que conlleva transformar una hipótesis basada en supuestos a interfaces de usuario tangibles.

Planos abstractos

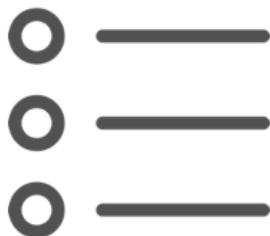


La estrategia es donde todo comienza:

- ¿Qué queremos obtener del sitio?
- ¿Qué quieren nuestros usuarios?

ESTRATEGIA

El primer plano es el más abstracto de este proceso y consta de la ideación del producto. Aquí se aplican diversos métodos que incluyen entrevistas a usuarios, análisis competitivos, entre otras estrategias usadas para conocer qué necesita el usuario.



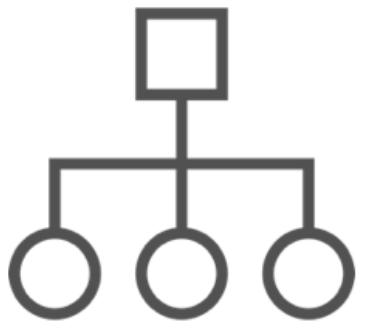
El alcance transforma la estrategia en requisitos:

- ¿Qué funcionalidades debe incluir el sitio?
- ¿Qué contenido requiere el sitio?

ALCANCE

En la segunda etapa se definen las especificaciones funcionales y de contenido de una plataforma, las cuales unidas se transforman en el alcance del sitio. Este alcance debe responder a:

- ¿Cómo podríamos resolver los problemas del usuario con nuestra plataforma?
- Y si las resuelvo ¿cuáles serían las características a usar y cómo podría priorizarlas?



ESTRUCTURA

La estructura da forma al alcance:

- ¿Cómo organizar el contenido?
- ¿Cómo navegará el usuario?

En la siguiente etapa, nuestras ideas comienzan a tomar forma en una estructura que permita a los usuarios interpretar de manera natural la información. En este punto se comienza la arquitectura de la información del sitio, o sea, clasificar y jerarquizar el contenido, etiquetar la información, crear la navegación del sitio y hacer que el contenido sea indexable para los motores de búsqueda.

El último plano, y por qué no decirlo, el más concreto de los planos abstractos es el esqueleto. Este plano busca plasmar el diseño de la interfaz y el diseño de la información.

El diseño de interfaz se utiliza para conocer la disposición de los elementos dentro de una interfaz, a modo de permitir a los usuarios interactuar con las funcionalidades de un sistema, mientras que el diseño de la información se ocupa de la presentación de la información de manera tal que facilite su comprensión.

Diseñadores de experiencia de usuarios (UX)

Teniendo en consideración los planos vistos hasta el momento, es bueno preguntarnos ¿Quién es el responsable de investigar y analizar la información obtenida en los planos abstractos? Pues bien, la persona encargada de estas tareas se denomina como **diseñador UX**.

Un diseñador UX es quien investiga y analiza cómo los usuarios se sienten al usar un producto, sea este tangible, como un smartphone, o intangible, como un sitio web. Los diseñadores UX se encargan de aplicar todo este conocimiento en el desarrollo de un producto a modo de garantizar que el usuario tenga la mejor experiencia posible.

Las responsabilidades asociadas a este rol son:

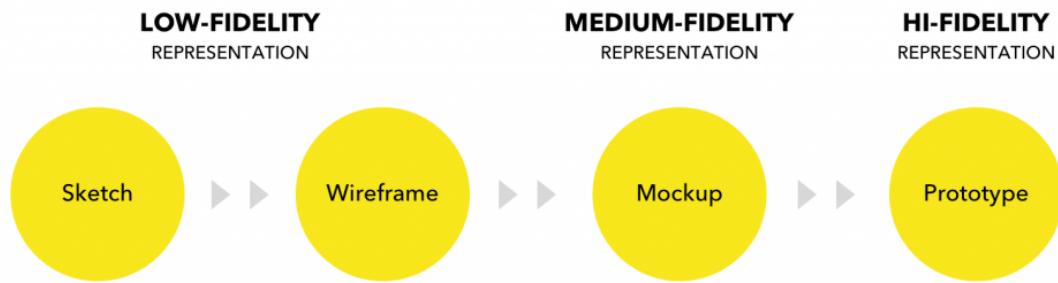
- Investigar la conducta del usuario
- Analizar los resultados de la investigación del usuario
- Informar a los miembros del equipo sus hallazgos
- Monitorear el desarrollo del proyecto
- Asegurar que los resultados sean implementados en el proyecto

Los diseñadores UX, por lo general, entregan a su equipo toda la investigación realizada usando un representación visual llamada Wireframe.

Antes seguir con el último plazo, es importante conocer qué son las representaciones visuales.

Representaciones visuales

Process of designing your first app



Las representaciones visuales podemos definirlas como la aplicación de una idea o concepto en un diseño visual, esto en el contexto visto hasta ahora significa crear un diseño que represente fielmente el proceso ideación de una interfaz de usuario. Estas se pueden segmentar dependiendo del grado de fidelidad con la cuál estén diseñados.

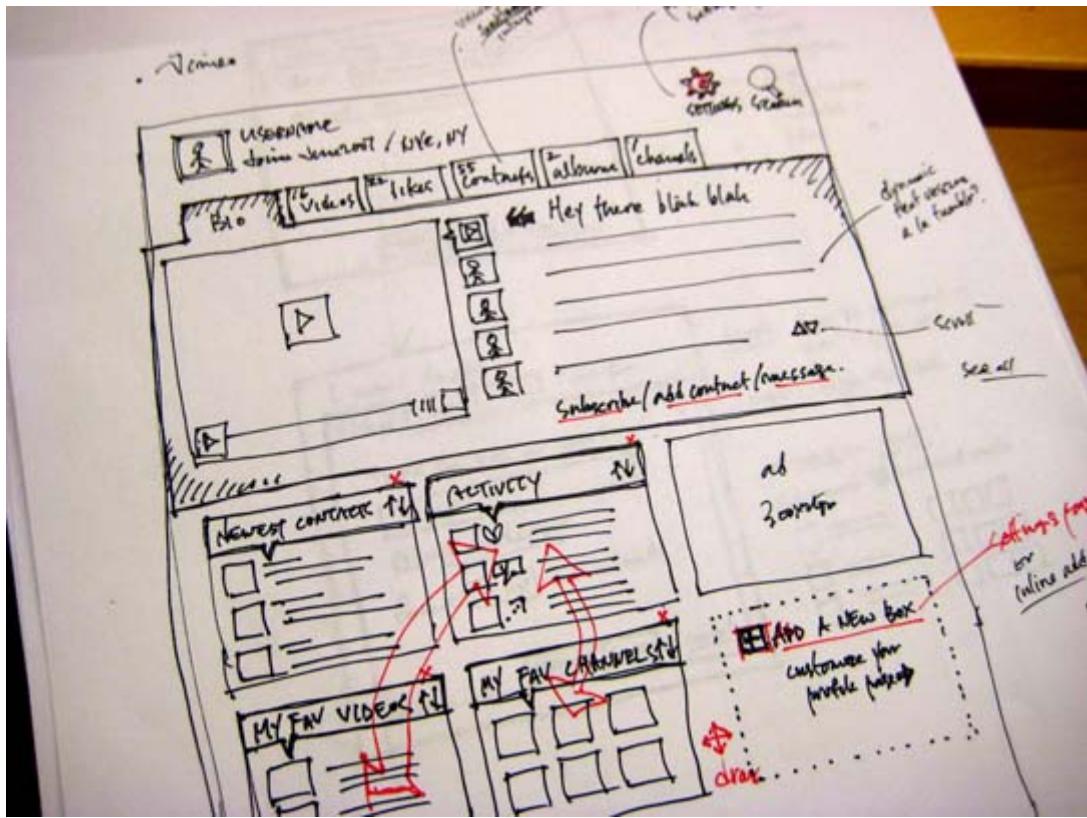
Las representaciones pueden ser:

Fidelidad Baja:

Son representaciones en papel o digital en las cuales se bosquejan estructura y/o contenido de una página web. Estas se desglosan en:

Sketch:

- Es la representación en papel de una página web o aplicación
- Es la forma más rápida de visualizar una página web



Wireframe:

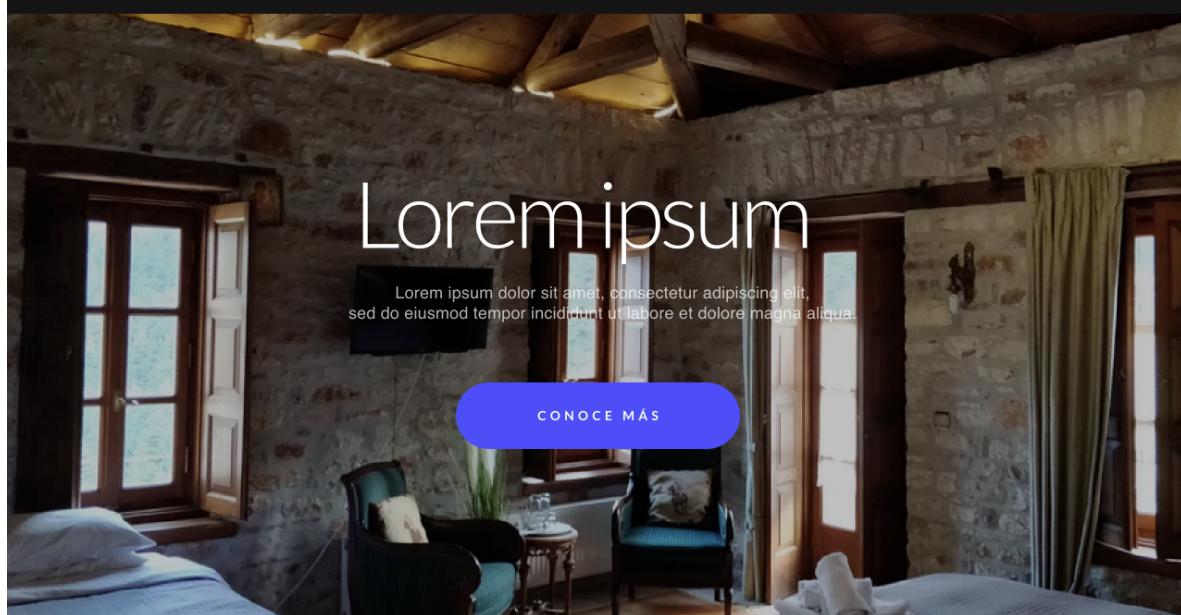
- Es el equivalente a un esqueleto de una web o aplicación.
- Es usado para describir estructura, contenidos y características de una web o aplicación.
- Un wireframe no contiene diseño.
- Es generalmente creado por programas destinados para este tipo de representación (ej. *Balsamiq, Axure R*, etc.)



Fidelidad Media:

Son representaciones digitales, que contienen todo el contenido visual de una web, incluyendo la estructura y contenido de las representaciones de baja fidelidad. Un ejemplo de este tipo de diseño es:

- **Mockup:**
 - Es una representación con diseño interfaz
 - Junta todas las características de un wireframes, con características propias de un diseño visual (colores, fuentes, imágenes, logos, iconos, etc)
 - Se crean usando diversos softwares como Photoshop, Illustrator, Sketch, Adobe XD, etc



Alta fidelidad:

Esta es la representación más fiel de una web sin ser desarrollada en código. En el se simulan las posibles interacciones que pueda hacer el usuario con la interfaz.

Prototipo:

- Un prototipo ofrece una representación de alta fidelidad de una página web o aplicación.
- Es un mockup enriquecido con piezas de la experiencia del usuario como interacciones y animaciones



Ahora que conocemos qué son las representaciones visuales y cuáles son las características más importantes de ellas podemos continuar con el último plano de Garrett.

Planos concretos



La superficie une todo visualmente:

- ¿cómo se verá la página terminada?

DISEÑO VISUAL

El plano más concreto presentado por garret en su libro "*Los elementos de la experiencia del usuario*" es el plano superficial o de diseño visual. En él se une todo de manera visual a modo de lograr una gran experiencia sensorial y de diseño visual.

El objetivo es establecer una conexión profunda con los usuarios mediante la comunicación exitosa de la marca, el producto, el valor y la funcionalidad en una interfaz de usuario.

En este plano encontraremos por lo general a diseñadores UI haciendo la mayor parte de este trabajo utilizando herramientas como Sketch, Photoshop, Adobe Experience Design, Illustrator o Adobe XD.

Diseñadores de Interfaces de usuario (UI)

Un diseñador UI es el encargado de diseñar las interfaces con las cuales el usuario va a interactuar.

Las interfaces de usuario creadas por los diseñadores UI son la suma de la arquitectura de la información, más los elementos visuales de la página y de los patrones de interacción creados por el diseñador UX.

Los diseñadores UI, por lo general, entregan a su equipo una representación visual de mediana y alta fidelidad, junto con una guía de estilos, las cuales nos darán una idea de los elementos que componen la interfaz de usuario.

En resumen, crear interfaces que sean representen las necesidades del usuario es la base de cualquier sitio web exitoso, y por lo mismo, conocer ese proceso es positivo para el maquetador ya que podremos construir interfaces en código que represente fielmente el trabajo realizado por los diseñadores UX y/o UI.

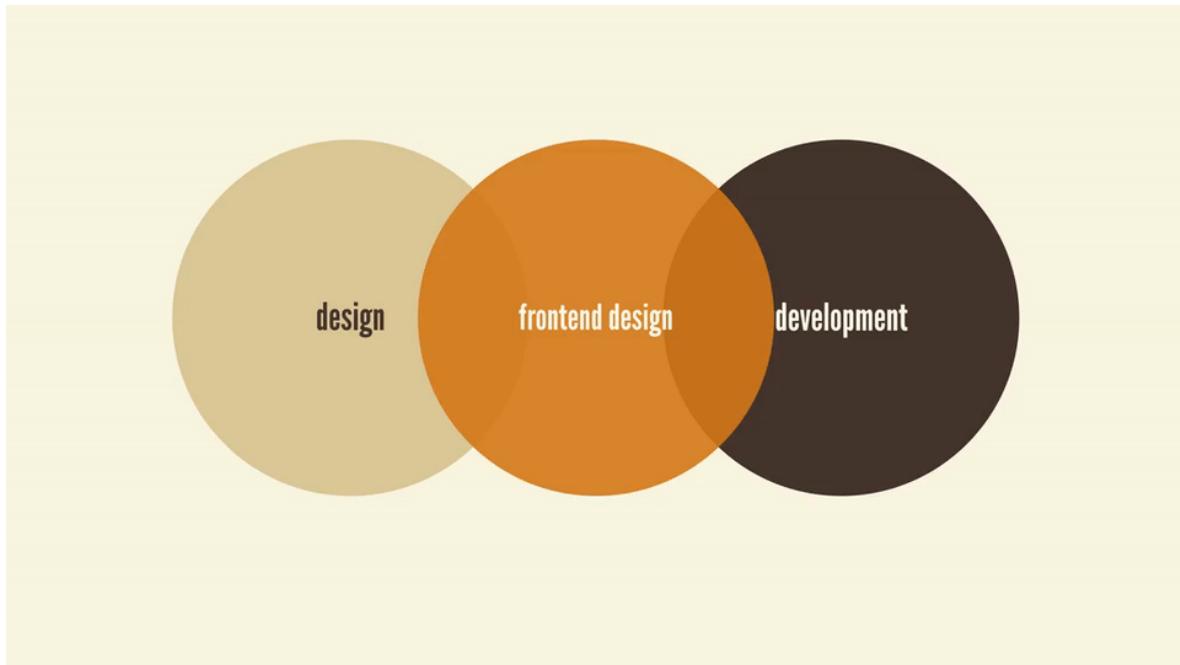
¿Qué hace un maquetador?

Anteriormente conocimos cuál es el proceso que implica crear interfaces de usuario (sitios web, aplicaciones, etc) y quienes son los encargados de investigar y aplicar esa investigación en una representación visual. Ahora conoceremos el trabajo y responsabilidades que tiene los maquetadores (También llamados desarrolladores UI o diseñadores front-end).

¿Qué es la maquetación?

La primera pregunta a responder acerca de la maquetación es saber qué es la maquetación.

La maquetación es la acción de traducir representaciones visuales (wireframe, mockup, prototipos) de manera fiel en código HTML, CSS y Javascript, es decir que esta deberá contener todas las características visuales y funcionales reunidas por los diseñadores UX/UI.



Maquetar significa convivir con el diseño y el desarrollo, ya que estaremos en medio de estas dos disciplinas. Esto implica:

- Comprender los **principios de la experiencia de usuario y sus buenas prácticas** a modo de no usar nuestro tiempo en investigar conductas de uso, escenarios y/o crear flujos.
- Entrenar el ojo para lo **estético** a modo de no usar nuestro tiempo en combinaciones de fuentes, comparando paletas de colores o creando iconos e ilustraciones.
- **Entender la importancia del backend** en el desarrollo a modo de no usar nuestro tiempo escribiendo lógica backend, trabajando en servidores o testeando la carga de una página.

La clave de ser un buen maquetador es reconocer que somos mediadores entre el diseño y el desarrollo, por lo tanto, nuestra tarea es entender como funcionan estos mundos, como conectarlos y de que manera podremos implementar la solución propuesta por nuestro equipo hacia el usuario.

El rol del maquetador

Por lo tanto, para poder crear interfaces que realmente reflejen la investigación y diseño creada por diseñadores UX/UI es necesario comprender algunos conceptos que nos ayudarán a realizar de mejor manera esta tarea.

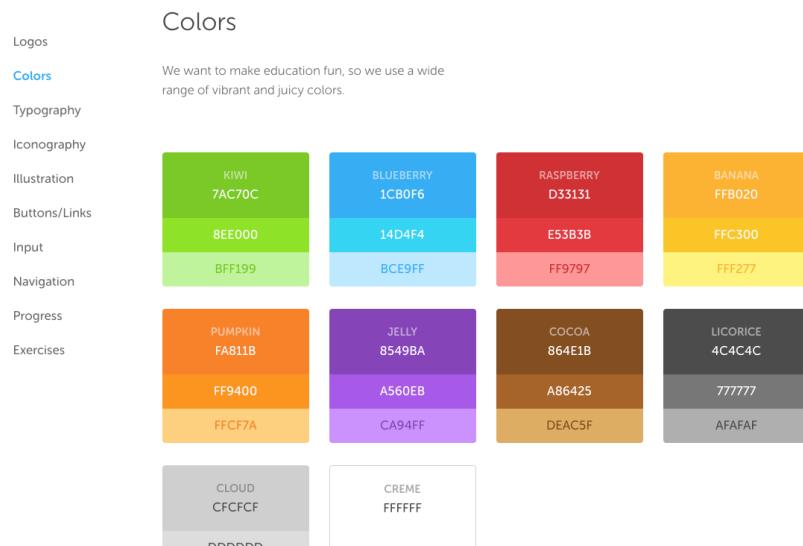
Para realizarla debemos:

Identificar el lenguaje visual de una representación visual:

Esto significa que deberemos conocer todos los elementos que componen la interfaz propuesta por el diseñador, de manera que sea más fácil decidir qué herramientas usar para crear el código de la interfaz.

Para poder identificar estos elementos es importante reconocer cuál es la identidad de la marca, el lenguaje de diseño utilizado y el tono, voz y escritura del contenido. Todo estos elementos descritos anteriormente se encuentran al interior de una guía de estilos.

Guías de estilos



Guía de estilos usada por Duolingo <https://www.duolingo.com/design/>

La guía de estilos es un entregable creado por un diseñador, el cual contiene las líneas, reglas, uso los elementos visuales incluídos dentro de una interfaz de usuario.

Este documento - en el contexto de un maquetador - es importante ya que nos dará las directrices para crear un sistema que nos permita desarrollar maquetas de manera eficiente, sin perder el foco del trabajo realizado en el proceso de diseño.

Algo que no podemos pasar por alto sobre las guías de estilos es el hecho que son optativas, o sea, que la creación de esta dependerá del proyecto, el tiempo, la cantidad de personas y que los roles estén bien distribuidos en un equipo de trabajo.

Si quieras conocer más sobre las guías de estilo y sus partes

¿Qué hacer si no tenemos guías de estilo?

Si no tenemos una guía de estilo para comenzar a maquetar siempre podremos usar algunas tácticas para conocer los elementos visuales de una representación visual.

Por ejemplo, podríamos hacer un sketch del diseño propuesto a modo de comprender las partes que componen esta interfaz de usuario, también podríamos obtener los colores y fuentes usados preguntando a la persona que creo la representación visual, conocer el manual de marca o buscar por internet a la empresa que requiere la interfaz.

Como podemos ver, las opciones son muchas, de modo que si ocurre esto siempre hay soluciones a la mano del maquetador.

Traducir el lenguaje visual a una maqueta:

Después de identificar los elementos que constituyen una representación debemos traducir esos elementos en funcionalidades tangibles usando nuestros conocimientos técnicos como maquetadores.

Estos, como sabemos, tiene que ver con el proficiente uso de lenguajes de marcas como HTML y CSS, además de algunas habilidades particulares que harán de este trabajo más organizado y fácil escalar.

Estos contemplan:

- Separar los elementos de una interfaz en pequeñas piezas de manera que sea más fácil construir estas piezas en código
- Crear un sistema o seguir una metodología de trabajo que nos ayude a organizar nuestros proyectos
- Usar herramientas que nos ayuden a trabajar de manera consistente en la construcción de nuestras maquetas
- Seguir buenas prácticas al escribir código HTML y CSS a modo de hacer fácil la lectura y mantención de nuestras maquetas por terceras personas

Documentar el proyecto maquetado:

Este último punto, aunque suene algo aburrido y tedioso, es importante para poder guiar al desarrollador que implementará nuestra maqueta al backend.

Esta documentación deberá contener una descripción de los elementos incluidos dentro del proyecto, además de presentar una guía de con algunas recomendaciones relacionadas a la estructura y estilos aplicados dentro del proyecto realizado.

En conclusión, el rol del maquetador va más allá del sólo hecho de traducir representaciones visuales ya que seremos responsables conectar dos mundos totalmente opuestos y unirlos para que el usuario final de pueda recibir la mejor experiencia.

Sass

Anteriormente descubrimos el papel que desempeña el maquetador y cuáles son sus responsabilidades dentro de un proyecto, ahora conoceremos cómo crear un flujo de trabajo que contemple las habilidades vistas anteriormente.

Como sabemos trabajar con CSS resultar puede caótico y desordenado cuándo los proyectos son grandes y contienen muchos estilos, por lo tanto es importante usar herramientas que nos ayuden a evitar que trabajar con CSS sea un suplicio.

Desarrolladores con este mismo problema crearon una herramienta la cual soluciona problemas típicos con CSS como la repetición estilos, la cascada de CSS y el tedium de mantener estilos por parte de terceros. Esta herramienta tan poderosa se llama preprocesador de CSS.

¿Qué son los preprocesadores de CSS?

Los preprocesadores de CSS son programas que nos permitirán generar CSS a partir de varios archivos que contienen una sintaxis propia de un preprocesador. Estos como mencionamos anteriormente tienen características que no existen en CSS puro como los mixin, el anidamiento, los selectores de herencia, entre otras.

Estos nos permitirán ahorrar tiempo a la hora desarrollar una maqueta, también nos ayudará a separar la lógica visual, o sea, separar los elementos que contiene el diseño de las representaciones visuales, así como también crear hojas de estilo más claras, consistentes y mantenibles.

A la fecha existe una variada selección de preprocesadores de CSS que podremos usar en nuestros proyectos como Sass, Less, Stylus y PostCSS. Nosotros durante todo el curso utilizaremos Sass como herramienta para trabajar con CSS.

¿Por qué usar Sass?

Trabajaremos con SASS debido a su gran compatibilidad con CSS, por sus características, por su madurez como preprocesador, y por su puesto, por la popularidad que tiene este en la industria en la industria del desarrollo.

¿Qué es Sass?

Sass es el acrónimo de "Sintactically Awesome Style Sheets". Este es una poderosa herramienta que nos ayudará a crear hojas de estilo claras, consistentes y fáciles de mantener.

¿Cómo instalar Sass?

Para instalar Sass necesitaremos realizar algunos pasos que finalizarán con la instalación de Sass en nuestro computador.

Antes de comenzar con la instalación es importante saber que todo el proceso será realizado por la terminal de nuestro computador, a modo de homogeneizar nuestros ambientes de trabajo. Si tienes Linux o Mac estarás de suerte, ya que la terminal viene por defecto en estos sistemas operativos. Por su parte, si eres usuario de windows deberás instalar la terminal en tu computador siguiendo la lectura que viene adjunta a esta experiencia.

El primer paso para tener en nuestros computadores Sass es instalar Node JS, el cual es un framework de Javascript del lado del servidor con el que podremos usar NPM, un administrador de paquetes con el cual descargaremos e instalaremos Sass.

Aunque parezca confuso, todo lo anteriormente visto no conlleva gran dificultad, ya que sólo deberemos seguir algunos pasos para usar Sass en nuestros computadores.

Instalar Node JS y NPM

Comencemos revisando si en nuestro computador tenemos instalado Node JS y NPM y Sass. Para hacerlo ingresaremos a nuestra terminal y escribiremos el siguiente comando:

```
node --version
```

Si lo tenemos instalado nos deberá aparecer la versión de Node JS:

```
node --version  
v8.12.0
```

Ahora revisemos si tenemos instalado NPM:

```
npm --version
```

Nos deberá aparecer un mensaje con la versión instalada:

```
npm --version  
6.4.1
```

Finalmente, revisemos si se encuentra instalado Sass:

```
sass --version
```

Si encuentra instalado nos deberá aparecer el siguiente mensaje:

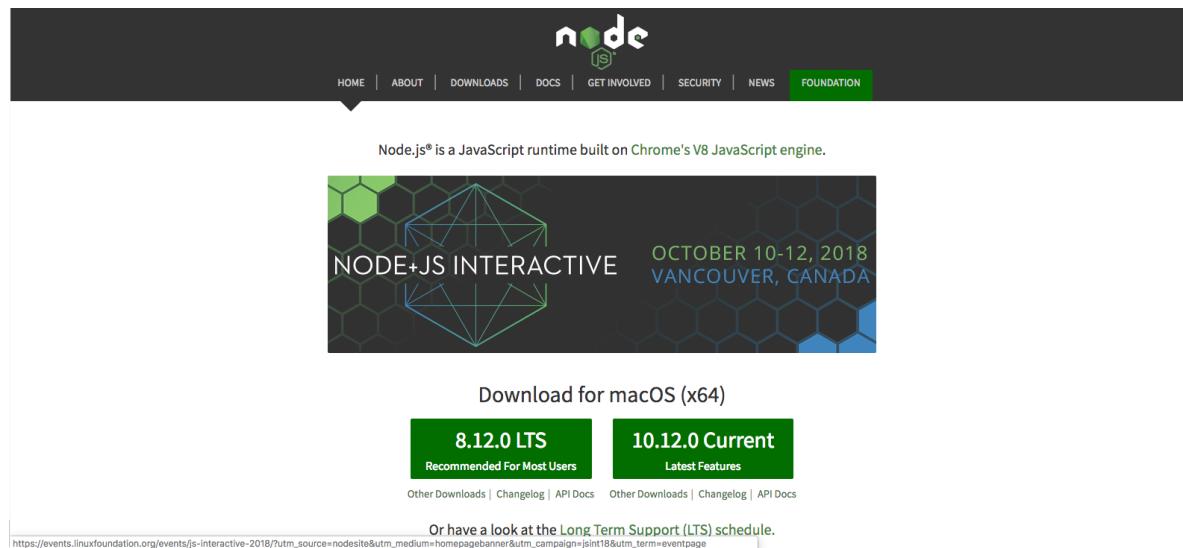
```
sass --version  
1.14.1 compiled with dart2js 2.0.0
```

En todos los casos vistos, si el mensaje de las versiones nos muestra que no se encuentra el comando especificado es necesario seguir adelante haciendo las instalaciones pertinentes para comenzar a trabajar con Sass.

Comencemos instalando Node en nuestro computador. Es importante notar que NPM viene incluído dentro de Node, de modo que no necesitaremos instalarlo.

Para instalar node deberemos ingresar a <https://nodejs.org/> y luego buscar la versión de node más estable.

Presionemos en esta versión para descargarla.



Luego de descargar deberemos seguir la instrucciones de instalación las que cambiarán visualmente dependiendo de nuestro sistema operativo.

Cuando terminemos instalar verifiquemos que se encuentre instalado Node y NPM escribiendo los mismos comandos vistas anteriormente.

```
node --version
```

```
v8.12.0
```

```
npm --version
```

```
6.4.1
```

Instalar Sass

Ahora, para finalizar este proceso instalaremos usando NPM la última versión de Sass. Para eso escribiremos el siguiente comando:

```
npm install -g sass
```

Deberemos esperar a que el paquete se descargue desde NPM.

```
npm install -g sass
```

```
/usr/local/bin/sass -> /usr/local/lib/node_modules/sass/sass.js
```

```
+ sass@1.14.2
```

```
updated 2 packages in 8.245s
```

Finalmente cuando se encuentre instalado deberemos revisar si se encuentra escribiendo el comando que vimos anteriormente:

```
sass --version
```

```
1.14.1 compiled with dart2js 2.0.0
```

Iniciar un proyecto con Sass

A continuación, cuando terminemos de instalar Sass deberemos probar que realmente funcione la compilación a CSS. Para hacerlo crearemos un directorio que contendrá los archivos bases de un proyecto web usando la terminal a modo de aconstumbrarnos a trabajar con ella.

Comencemos ingresando al home de nuestra terminal escribiendo `cd ~`. Desde aquí ingresaremos al directorio desktop (`cd desktop`) y crearemos la carpeta test escribiendo `mkdir test_sass`.

Con ello listo, ingresaremos a la carpeta (`cd test_sass`) para crear dos archivos. El primero se llamará `style.css` y el segundo `style.scss`.

Antes de continuar es importante conocer las extensiones usadas en Sass para sus archivos.

Sass vs SCSS

Sass tiene 2 sintaxis disponibles:

- Sass con una extensión de archivo `.sass`
- SCSS (Sassy CSS) con una extensión de archivo `.scss`.

Nosotros usaremos la extensión `.scss` para crear nuestros archivos debido a su:

- **Legibilidad:** La sintaxis es muy similar a CSS

- **Curva de aprendizaje:** Solo agrega algunas características adicionales sobre CSS
- **Compatibilidad:** Un archivo CSS es válido como archivo SCSS
- **Recursos:** Existe una gran cantidad de artículos y librerías de Sass que lo utilizan
- **Escalabilidad:** Es fácil pasar de SCSS a CSS

Teniendo esto claro, es momento de seguir con la creación de los archivos de prueba.

Crearemos los archivos escribiendo `touch` y luego el nombre de los archivos separados por un espacio.

```
touch style.css style.scss
```

Cuando estén listos los archivos deberemos volver a la raíz (`cd ..`) para comenzar a testear si funciona el compilado de Sass.

¿Cómo compilar Sass?

Para poder compilar Sass deberemos usar el comando `sass --watch`. Este comando le dirá a Sass que examine nuestros archivos en busca de cambios, a modo de compilar en CSS cada cambio hecho en Sass.

Para comenzar a compilar debemos escribir en la terminal el nombre del comando, luego el nombre del archivo SCSS (input), seguido por `:` y finalmente el nombre del archivo CSS (output).

```
sass --watch estilos_a_compilar.scss:estilos_compilados.css
```

Ahora que sabemos que hace el comando `sass --watch`, comencemos a compilar el proyecto de prueba escribiendo en la terminal:

```
sass --watch style.scss:style.css
```

Cuando comience a examinar veremos un mensaje similar a este:

```
sass --watch style.scss:style.css
Sass is watching for changes. Press Ctrl-C to stop.
```

Probar el compilado de Sass

La última que nos queda para comenzar a construir un proyecto con Sass es probar si funciona el compilado escribiendo algún estilo en el archivo SCSS para revisar si se replica en el archivo CSS.

Escribamos dentro del HTML (`index.html`) su estructura base y un título.

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Test Sass</title>
</head>
<body>
  <h1>El compilado ha funcionado</h1>
</body>
</html>

```

Además, agreguemos dentro de `<head>` una etiqueta `<link>` para poder ver los estilos.

```

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <link href="style.css">
  <title>Test Sass</title>
</head>

```

Cuando terminemos demos algunos estilos al título. Dentro del archivo SCSS agreguemos el selector del elemento `<h1>`. Dentro de el agreguemos una fuente del tipo monospace `font-family: monospace;`, y otra cambiando el color de la fuente a un valor peru `color:peru;`.

```

h1 {
  font-family: monospace;
  color: peru;
}

```

Como resultado veremos que en la terminal un mensaje que confirma la correcta compilación a CSS de nuestros estilos

```

sass --watch style.scss:style.css
Sass is watching for changes. Press Ctrl-C to stop.

Compiled style.scss to style.css.

```

Para terminar cerramos el proceso de compilación de Sass (`sass --watch`) escribiendo en la terminal `ctrl + c`.

Estructurar un proyecto en Sass

Ahora que terminamos de probar que Sass es totalmente funcional en nuestro computador es momento de realizar una de las tareas más importantes a la hora de trabajar con Sass ya que realizarlo de buena manera en nuestro proyecto hará que podamos explotar todo el potencial de Sass.

¿Por qué estructurar un proyecto con Sass?

La primera pregunta que deberíamos responder antes de comenzar es entender por qué es necesario estructurar nuestro proyecto con Sass.

La razón tiene que ver con el orden y la mantenibilidad de nuestros estilos, la separación lógica de los elementos visuales de la interfaz que deseemos traducir a código y porque Sass posee características que nos permitirán organizar nuestro proyecto.

Para ser más precisos:

Orden y mantenibilidad

El orden y la mantenibilidad de nuestros estilos es muy importante ya que habrán personas que posteriormente usen y mantengan nuestros estilos, de modo que es importante una buena organización para ellos.

Un buen orden en un proyecto en Sass significa que los estilos deberán estar organizados de tal manera que buscar estilos de un elemento específico sea fácil, que evitemos errores por cascada de CSS, y que mantener para otros no sea un tedio.

Para poder lograr este objetivo es importante reconocer cómo separar la lógica visual de una interfaz de usuario.

Separar la lógica de los elementos visuales

Separar la lógica visual de una interfaz de usuario contempla identificar los componentes y elementos que conforman un diseño. Estos a grandes rasgos se pueden separar en:

- Los colores, tipografías, tamaños, entre otros contenidos visuales.
- El diseño base de la interfaz, o sea, de qué manera se encuentran estructurados los componentes de la interfaz
- Los elementos que componen una interfaz (botones, inputs, navegación, etc)
- Las interacciones de la interfaz

Entender una interfaz de manera modular hará que crear estilos en nuestro proyecto será más organizado, ya que el flujo de trabajo será enfocado a crear pequeños trozos de código y no a grandes hojas de estilo desordenadas. Esto en proyectos grandes es fundamental para mejorar la consistencia y mantenibilidad de nuestros estilos.

Uso de características de Sass para estructurar proyecto

Finalmente, para poder realizar todo lo anteriormente planteado es importante poder generar una estructura para nuestros proyectos que cree esa lógica. Sass como herramienta tiene las características suficientes ayudarnos en esta tarea.

¿Qué características podemos usar para estructurar nuestro proyecto?

La primera característica que podremos usar para nuestro beneficio son los archivos parciales.

Archivos parciales:

Los archivos parciales son archivos de Sass (SCSS en nuestro caso) en los que podremos agregar pequeños fragmentos de estilos.

Estos archivos para poder ser reconocidos por Sass deberán comenzar con guión bajo (`_`). Asimismo el guión bajo permite a Sass reconocer que estos archivos no deben compilarse en un archivo CSS, a menos que nosotros lo explicitemos.

Un ejemplo de un archivo parcial podrían ser los estilos de los botones de un sitio web lo cuáles se encuentran modularizados en este archivo.

```
_buttons.scss
```

Para poder compilar este archivo parcial a CSS deberemos usar una directiva llamada `@import`.

@import:

Es una extensión de la opción `@import` de CSS, que toma un archivo que se desea importar y lo combina con otro archivo el cual se desee importar. Es importante acotar que `@import` de Sass no requiere de una solicitud HTTP como lo hace CSS para importar un archivo, esto hace una gran diferencia de uso entre estas dos instrucciones.

Teniendo clara esta diferencia, veamos un ejemplo de su uso:

Supongamos que hemos terminado los botones del sitio y ahora queremos compilarlos a CSS. Para poder hacerlo deberemos usar la directiva `@import` dentro del archivo SCSS input (`main.scss`) agregando esta de la siguiente manera:

```
// En este caso el archivo parcial se encuentra en un directorio llamado
"components", dentro de la carpeta raíz de Sass
@import "components/buttons"
```

Como vemos su uso e implementación es bastante simple, pero existen algunas consideraciones que debemos tener en cuenta para usar de la mejor manera esta directiva:

1. Podremos agregar archivos parciales en otro archivo usando rutas relativas con extensión (.scss) y sin ellas, como en el ejemplo anterior. La segunda opción es la más flexible de las dos ya que no necesitaremos conocer su extensión.
2. Debemos tener cuidado en identificar correctamente la dirección de la ruta a agregar a modo de evitar problemas posteriores
3. Debemos crear una estructura de directorio que separé los archivos parciales del archivo principal (input).

Tener en consideración estas recomendaciones hará que evitemos problemas posteriores y que por supuesto organicemos de mejor manera nuestro proyecto con Sass.

Manifiestos:

Cuándo hablamos de los `@import` usamos un archivo el cual era el input de Sass, o sea, el archivo que contiene los estilos escritos con el preprocesador, que posteriormente se convertirán a CSS.

Este archivo que hemos usado hasta ahora, es una pieza clave en la organización de un proyecto Sass ya que con el podremos ordenar todos los archivos parciales usados en un proyecto, dentro de una gran hoja de estilos. Esta hoja principal de estilos principal es conocida como manifiesto.

Un manifiesto en Sass es un archivo que contiene un grupo de archivos importados que juntos forman una unidad que ensamblados representan al archivo principal o input con el cual será compilado posteriormente a CSS.

Imaginemos que tenemos la siguiente estructura en nuestro proyecto:

```
 proyecto/
 | - assets
 |   | - css
 |   |   |
 |   |   |     `-- main.css
 |   |
 |   |     `-- sass
 |   | - components/
 |   |   | - _buttons.scss
 |   |   |   | - _inputs.scss
 |   |   |   |     `-- _dropdown.scss
 |   |   |
 |   |   `-- main.scss
 |
 `-- index.html
```

Si quisieramos agregar los archivos parciales que se encuentran dentro del directorio "components" en el manifiesto, o sea, en main.scss, deberemos ingresar al manifiesto e importar los archivos de la siguiente manera:

```
// Componentes del sitio
@import 'components/buttons';
@import 'components/inputs';
@import 'components/dropdown';
```

Algo a notar es que el orden ingresado en los manifiestos es importante, ya que algunas reglas CSS pueden depender de otras para funcionar, por lo tanto organizar los parciales importados puede ser necesario en caso de que exista esa dependencia.

Comentarios:

El último elemento que nos ayudará a organizar nuestros proyectos con Sass son los comentarios. Los comentarios como en muchos otros lenguajes es imprescindible para ordenar y comentar nuestro código.

Con los comentarios podemos:

- Crear la estructura y rol de un archivo
- Mostrar a otros desarrolladores la razón de una declaración
- Ordenar las declaraciones de CSS
- Explicar el proceso detrás de los estilos creados
- Entre otros.

Para escribir comentarios en Sass tendremos dos opciones de sintaxis para usar:

Comentarios de bloques:

```
/**  
 * Esta clase contiene a todos los elementos dentro de ella.  
 * Para un correcto resultado:  
 * - Mantener en display: flex;  
 */  
.container {  
    display: flex;  
    flex-direction: column;  
    align-items: center;  
}
```

La primera se llaman comentarios de bloques. Estos se usan comúnmente en CSS nativo y son totalmente compatibles con Sass. La gran particularidad de estos comentarios en Sass es que estos se compilan a CSS, de modo que los comentarios escritos en SCSS podrán ser vistos en hoja de estilos final.

Comentarios de línea

```
// Variables de color  
$light-salmon-pink: #FFA69E;  
$beige: #FAF3DD;  
$pale-turquoise: #B8F2E6;  
$light-blue: #AED9E0;  
$cadet: #5E6472;
```

Los comentarios de líneas son exclusivos de Sass y con ellos podremos escribir comentarios que sólo vivirán en la hojas de estilos SCSS, de modo que su contenido será visible sólo en Sass.

Tener en cuenta esta diferencia es fundamental para discriminar cuando usar comentarios de línea o de bloque en Sass dependiendo del mensaje o contexto en el que se usará el comentario.

El uso en conjunto de todas estas características serán clave para construir una estructura robusta y eficaz en nuestros proyectos, especialmente si queremos que nuestros estilos sean consistentes, mantenibles y por supuesto que nos hagan reducir la repetición de nuestro código.

Crear un proyecto con Sass

Como vimos anteriormente, estructurar un proyecto con Sass es la base para poder crear estilos consistentes y fáciles de mantener por otras personas.

Ahora crearemos un proyecto usando todas las características vistas a modo de entender mejor este flujo.

Métodos y técnicas para estructurar proyecto con Sass

Lo primero que debemos tener en cuenta al construir esta estructura es que comenzar a realizar puede resultar tedioso y difícil de comprender si aún no tenemos definido como organizar un estructura de directorio para nuestro proyecto.

Como todo los problemas en el mundo, existen personas con muy buenas ideas que ya han depurado sistemas, métodos y técnicas que nos ayudarán en la creación de esta estructura.

Dentro de este tópico podremos encontrar muchas arquitecturas para Sass. Algunos buenos ejemplos son:

Bootstrap:

```
bootstrap/
|- bootstrap.scss    # Manifest file
|- _alerts.scss      # Component file
|- _buttons.scss     # Component file
|- _mixins.scss      # Mixin file - imports all files from mixins folder
|- ...
|- mixins/
| |- _alerts.scss # Alert mixin
| |- _buttons.scss # Button mixin
| |- ...
# Etc..
```

La estructura de Sass usada por Bootstrap separa su estilos por componentes usando archivos parciales por cada uno de ellos. Todos los archivos parciales y directorios se encuentran dentro del manifiesto de Bootstrap.

Zurb Foundation:

```
foundation/
|- foundation.scss   # Manifest file
|- _global.scss       # Global styles
|- components/
| |- _alerts.scss # Alert mixin
| |- _buttons.scss # Button mixin
| |- ...
# Etc..
|- ...
```

Zurb Foundation es un Framework CSS el cuál ha sido la competencia más importante de Bootstrap desde hace bastante tiempo.

La estructura de Sass utilizada por este framework CSS tiene como característica el uso de directorio para separar los componentes y elementos usados. Todos estos elementos son importados al manifiesto llamado `foundation.scss`.

Patrón 7-1:

```
sass/
|
|- abstracts/
|   |- _variables.scss    # Sass Variables
|   |- _functions.scss    # Sass Functions
|   ...
# etc..
|
|- base/
|   |- _reset.scss        # Reset/normalize
|   |- _typography.scss   # Typography rules
|   ...
# Etc.
|
|- components/
|   |- _buttons.scss      # Buttons
|   |- _carousel.scss     # Carousel
|   ...
# Etc..
```

```

|
|- layout/
|   |- _navigation.scss    # Navigation
|   |- _grid.scss          # Grid system
|   ...
|   ...
|
|- pages/
|   |- _home.scss          # Home specific styles
|   |- _contact.scss        # Contact specific styles
|   ...
|   ...
|
|- themes/
|   |- _theme.scss          # Default theme
|   |- _admin.scss          # Admin theme
|   ...
|   ...
|
|- vendors/
|   |- _bootstrap.scss      # Bootstrap
|   |- _jquery-ui.scss       # jQuery UI
|   ...
|   ...
`- main.scss               # Main Sass file

```

El patrón 7 - 1 es una estructura creada por [Hugo Giraudel](#), la cual se basa en un principio muy simple: 7 directorios, 1 archivo. Esta técnica separa los parciales en 7 directorio diferentes y un archivo que funciona como manifiesto.

Todas las opciones vistas anteriormente son totalmente válidas para cualquier proyecto. Nosotros, por su facilidad de implementación y características usaremos el patrón 7-1 para estructurar y organizar nuestro proyecto con Sass.

¿Cómo implementar el patrón 7 - 1?

Para implementar el patrón 7-1 en nuestro proyecto deberemos, en primer lugar, conocer más sobre su estructura.

Ingresemos a sass guilines para conocer más sobre el [patrón 7-1](#).

La estructura base de esta técnica consta de 7 directorios que se encuentran dentro de un carpeta raíz.

```

sass/
  base/
  components/
  layout/
  pages/
  themes/
  abstracts/
  vendors/
  main.scss

```

Esta carpeta raíz aloja en su interior un manifiesto que contendrá a su vez los archivos parciales de nuestro proyecto Sass.

Los archivos parciales que contendrán los elementos de nuestra interfaz se dividen de la siguiente manera:

Abstracts

```
- abstracts/
|   |- _variables.scss      # Sass Variables
|   |- _functions.scss      # Sass Functions
|   |- _mixins.scss         # Sass Mixins
|   |- _placeholders.scss   # Sass Placeholders
```

La carpeta "abstracts" contiene todos los parciales relacionados a todas las herramientas y helpers de Sass usados en nuestro proyecto. Aquí encontraremos las variables globales, funciones y mixin que deseemos guardar.

La regla de oro de este directorio es que ninguno de los elementos incluídos dentro se deben compilar a CSS.

Base

```
|- base/
|   |- _reset.scss          # Reset/normalize
|   |- _typography.scss     # Typography rules
|   |- _base.scss           # Base styles
|   ...                     # Etc.
```

El directorio "base" será, como dice su nombre, la base de nuestro proyecto. En el podremos guardar todos los estilos principales como las tipografías, reset y estilos comunitante usados en HTML.

Components

```
|- components/
|   |- _buttons.scss        # Buttons
|   |- _carousel.scss       # Carousel
|   ...                     # Etc.
```

En "components" irán todos los parciales que representen un pequeño componente de nuestra interfaz. Elementos que cumplan una función de manera modular como carruseles, dropdown, thumbnails serán bienvenidos en este directorio.

Layout

```
|- layout/
|   |- _navigation.scss    # Navigation
|   |- _grid.scss           # Grid system
|   ...                     # Etc.
```

"Layout" contiene todo lo que tenga que ver con el layout o diseño de la interfaz. Esto incluye los elementos típicos de layout como los header, footer, sidebar, navegación, además de las grillas que le dan forma a este diseño.

Pages

```
| - pages/
|   | - _home.scss      # Home specific styles
|   | - _contact.scss    # Contact specific styles
|   ...
|   ...
```

En el caso del directorio "pages", si tuvieramos distintos tipos de diseño en las páginas deberíamos agregar este archivo.

En el caso de no tener distintos tipos de páginas, este puede quedar sin contenido.

Themes

```
| - themes/
|   | - _theme.scss      # Default theme
|   | - _admin.scss       # Admin theme
|   ...
|   ...
```

En interfaces grandes, es común encontrar este tipo de estilos los cuales tienen diferentes formas de definir un tema dependiendo de factores como el tipo de usuario que la usará o si la marca usa diferentes colores dependiendo del lugar geográfico.

Esta carpeta en proyecto pequeños o específicos es probable que no sea necesario usarla, de modo que si ese es el caso, lo mejor que podemos hacer es dejarla vacía.

Vendor

```
| - vendors/
|   | - _bootstrap.scss    # Bootstrap
|   | - _jquery-ui.scss     # jQuery UI
|   ...
|   ...
```

Finalmente encontraremos el directorio vendor. En esta carpeta se encuentran todos los archivos CSS de bibliotecas, plugins y frameworks externos como: Normalize, Bootstrap, entre otras.

El usar esta carpeta es una buena manera de separar el código que es nuestro con el código de otras personas.

main.scss

Un último elemento crucial de esta estructura es el archivo "main.scss". Este como sabemos es el manifiesto que contendrá todo el código que será compilado a CSS.

Orden de carga patrón 7-1

La carga dentro de este archivo deberá tener la siguiente estructura a modo de evitar errores de carga o cascada:

```
abstracts/  
vendors/  
base/  
layout/  
components/  
pages/  
themes/
```

Esto significa que al ingresar los archivos dentro del manifiesto estos deberán ser ingresados comenzando con parciales de abstracts y terminando con themes.

Según nos especifica Sass guideline: Para mantener la legibilidad en el manifiesto, el archivo deberá:

- Ir un archivo por `@import`
- Haber un `@import` por línea
- No hay que dejar una línea después de importar archivos de la misma carpeta
- La extensión del archivo y guión bajo deberá ser omitido de la ruta.

Siguiendo estas recomendaciones obtendremos el siguiente resultado:

```
// main.scss # Manifest File  
  
@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';  
  
@import 'components/buttons';  
@import 'components/carousel';  
@import 'components/cover';  
@import 'components/dropdown';  
  
@import 'pages/home';  
@import 'pages/contact';  
  
@import 'themes/theme';  
@import 'themes/admin';
```

Luego de conocer como funciona este patrón es momento de implementarlo de manera práctica.

Para hacerlo usaremos el siguiente ejemplo:

Imaginemos que una empresa llamada Houstel desea maquetar su nueva página de inicio. Esta página de inicio se encuentra totalmente lista para ser creada, ya que diseñadores UX/UI identificaron las necesidades del usuario y tradujeron esas necesidades en una representación visual.

Los entregables que tendremos a disposición serán:

Guías de estilo:

El primer entregable es la guía de estilos usada en el mockup. Esta incluye la información relacionada a la paleta de colores, tipografías, tamaños de fuentes, así como también los componentes que contiene la representación visual.

HOUTEL

Styleguide

Colors Scheme

White #FFF	Carmine pink #E94F37	Black olive #E94F37	Verdigris #44BBA4
Platinum #E8E8E8	Salmon #EF7F6D	Granite gray #5D6163	Medium aquamarine #66C7B4
Gray #BABABA	Spanish pink #F7FBF6	Old silver #F7FBF6	Perl Aqua #88D3C5

Typography

Typefaces

Playfair Display Bold

Open Sans Bold

Open Sans Semi Bold

Open Sans Regular

Font Sizes

My slave

130px — xxxx-large

My slave human

100px — xxx-large

My slave human didn't give me any food

30px — xx-large

My slave human didn't give me any food

20px — x-large

My slave human didn't give me any food

18px — large

My slave human didn't give me any food

16px — medium

My slave human didn't give me any food

14px — small

My slave human didn't give me any food

14px — x-small

Components

Form

Input text

Select menu

Buttons

Button size

Large button

Medium button

Small button

Button color scheme

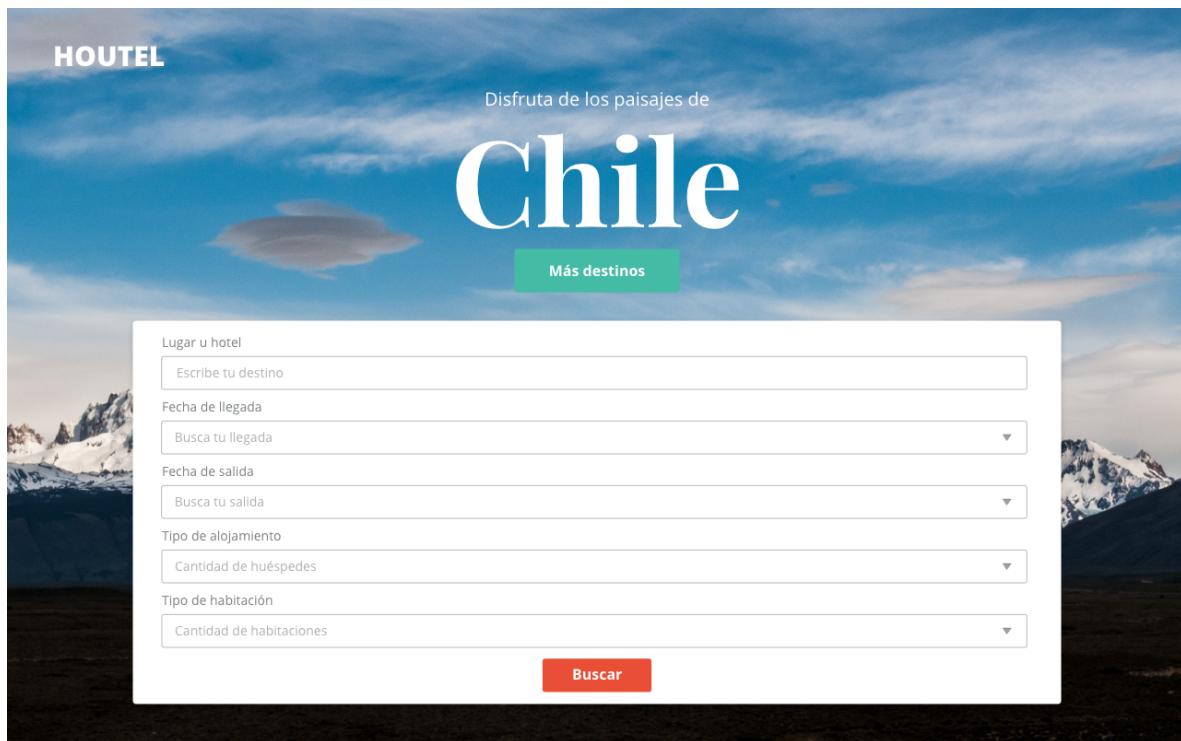
Default

Positive



Representaciones visuales:

El otro entregable cedido por Houstel es la representación visual del home. Este está compuesto por un mockup con la interfaz del sitio.



Mockup sitio en computador

Imágenes:

Las imágenes corresponden al fondo principal y al logo del sitio.

Con los elementos entregados por el UI tendremos el material suficiente para poder comenzar con nuestro proyecto.

Creando la arquitectura del proyecto:

El primer paso es crear la base de nuestro proyecto. Para eso utilizaremos la terminal.

Creemos el directorio principal del proyecto. Lo llamaremos *houstel_site*.

```
mkdir houstel-site
```

Luego ingresemos a él.

```
cd houstel-site
```

Dentro creemos el archivo HTML principal, y luego el creemos el directorio assets.

```
touch index.html  
mkdir assets
```

Ahora ingresemos a assets y agreguemos los directorios css sass e images. Estos directorios contendrán nuestras imágenes y estilos.

```
cd assets  
mkdir css sass images
```

Los últimos dos pasos serán crear el archivo output para css y luego crear la estructura de directorio de Sass.

Ingresemos al directorio CSS y creamos un archivo llamado `main.css`.

```
cd css  
touch main.css
```

Por último volvamos a assets, ingresemos a Sass y creamos las carpetas y archivo manifiesto necesario para crear el patrón 7-1.

```
cd ..  
cd assets  
mkdir abstracts base components layout pages themes vendors  
touch main.scss
```

Organizando parciales del proyecto

Para organizar los archivos parciales que utilizaremos en nuestro proyecto es necesario identificar en qué directorio irá cada uno de los elementos que componen nuestra interfaz.

Primero revisemos que elementos componen esta interfaz.

Layout:

Lo primero que podremos hacer es definir las partes que componen el diseño del sitio a maquetar.

Este sitio lo podemos separar de la siguiente manera.

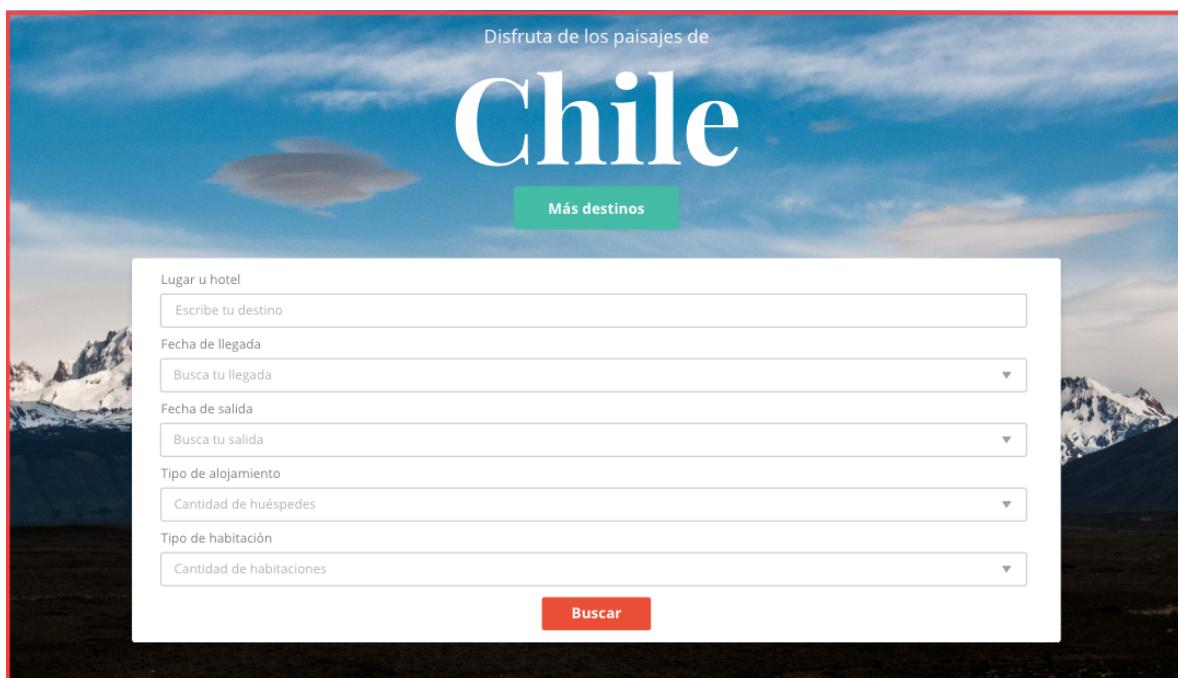
Header:



La primera sección que podremos identificar es el header o cabecera del sitio.

Main:

La siguiente parte del diseño que podemos identificar es el main o contenido principal del sitio.



Form:

El último layout identifiable es el form o 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

Estos elementos que vimos anteriormente son la base de los componentes integrados al interior de cada uno de los layout.

¿En dónde integro estos layout dentro del patrón 7-1?

Integrar estos layout dentro del directorio de Sass es fácil. Primero debemos ingresar a la carpeta especializada para estos estilos llamada "Layout" y luego crear tres archivos que con nombres que representen su tipo. En base a lo visto anteriormente sus nombres deberán ser: `_header.scss`, `_main.scss` y `_forms.scss`.

Con el archivo `_main.scss` es mejor usar un nombre más específico debido al alcance de nombre del manifiesto, por lo tanto un buen nombre para este archivo es `_main-section.scss`.

Ahora sí, ingresemos a la carpeta layout y creamos los archivos `_header.scss`, `_main-section.scss` y `_forms.scss`.

```
cd layout
touch _header.scss _main-section.scss _forms.scss
```

Components

Para agregar los componentes usaremos la guía de estilos facilitada por diseñador UI. En el encontraremos:

Inputs

El primer componente que encontraremos serán los inputs. Esto como sabemos forman parte del formulario de búsqueda.

Input text

placeholder text

Select menu

placeholder text ▾

Botones

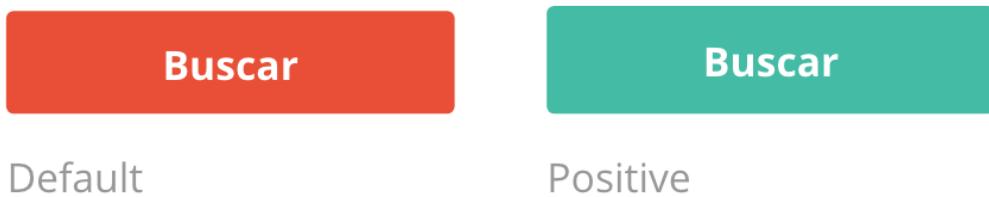
El otro componente que incluye esta interfaz son los botones.

Estos botones contienen información muy importante como el tamaño de los botones, ademas de los colores que tienen estos dependiendo de la acción que realice el usuario.

Button size



Button color scheme



¿Cómo integrarlos en el proyecto?

Para integrarlos al proyecto deberemos realizar la misma acción hecha anteriormente, o sea, volver a la raíz de la carpeta Sass, ingresar a components y finalmente crear los dos componentes vistos en la guía de estilos.

```
cd ..  
cd components  
touch _inputs.scss _buttons.scss
```

Base

Ahora que ya tenemos definido el diseño y los componentes que podremos utilizar en nuestra maqueta el siguiente paso es agregar los archivos que serán la base de nuestros estilos.

Si revisamos Sass Guidelines, podremos encontrar que dentro de este directorio se pueden incluir tres parciales.

El primero es llamado `_base.scss`, en este archivo se pueden definir algunas reglas de estilo para elementos comunitarios usados por HTML.

El segundo es `_reset.scss`. En él podremos agregar reglas para resetear y normalizar nuestros estilos en todo tipo de navegadores. Este tema lo abordaremos con más detalle más adelante.

El último parcial que contendrá este directorio es `_typography.scss`. Aquí podremos agregar todas las reglas relacionadas a las tipografías del proyecto.

De manera optativa, y si el proyecto lo requiere, podremos agregar un parcial que integre en su interior las animaciones de nuestra maqueta.

¿Cómo integrarlos en el proyecto?

Agreguemos estos parciales volviendo al directorio sass, luego ingresando a base y finalmente agregando los archivos al directorio.

```
cd ..
cd base
touch _base.scss _reset.scss _typography.scss
```

Abstracts

El último paso será agregar los parciales dentro del directorio abstracts del patrón 7-1. Como sabemos en este directorio se encontrarán todos las herramientas de Sass que reutilizaremos a lo largo del proyecto.

En nuestro caso utilizaremos sólo dos herramientas para realizar este proyecto. La primera serán las variables y la segunda serán los mixin los cuáles nos permitirán a crear algunas reglas CSS que podremos reutilizar dentro de nuestras hojas de estilo agregando algunos argumentos en su interior..

. Estas funciones de Sass las revisaremos más adelante, así que por ahora crearemos estos parciales dentro de la carpeta abstracts.

¿Cómo integrarlos en el proyecto?

Para hacerlo volveremos al directorio sass, posteriormente entraremos a la carpeta abstracts y finalmente crearemos los parciales `_variables.scss` y `_mixins.scss`.

```
cd ..
cd abstracts
touch _variables.scss mixins.scss
```

¿Qué hacer con las carpetas restantes?

Las carpetas restantes como pages y themes las dejaremos como están, ya que por el tamaño del proyecto no son necesarias. En cuanto al directorio vendor más adelante la utilizaremos para agregar algunas dependencias que nos serán útiles a medida que avancemos con el proyecto.

Importar parciales al manifiesto

El último paso a realizar es el más importante de toda la serie, ya que importaremos todos los parciales que incluimos dentro de los directorios del patrón 7-1 hacia el manifiesto.

Es importante recordar que el orden de los parciales en este archivo es importante para el buen funcionamiento de todos nuestros estilos a lo largo del proyecto.

El orden que deberían tener los @imports es el siguiente:

```
// main.scss # Manifest File

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

// Vendors

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

// Layout
@import 'layout/header';
@import 'layout/main-section';
@import 'layout/forms';

// Components
@import 'components/buttons';
@import 'components/inputs';

// Pages

// Theme
```

Usemos como plantilla este orden para hacerlo en nuestro proyecto.

Comencemos agregando los parciales de abstracts.

```
// abstracts
@import 'abstracts/variables';
@import 'abstracts/mixins';
```

Vendor por ahora no contendrá nada.

```
// Vendors
```

Sigamos con base.

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

Sólo falta layout.

```
// Layout
@import 'layout/header';
@import 'layout/main-section';
@import 'layout/forms';
```

Y finalmente components.

```
// Components
@import 'components/buttons';
@import 'components/inputs';
```

Los directorios pages y themes no los usaremos en este proyecto, pero es una buena idea mantener comentado el lugar que deberían ir estos dentro del manifiesto.

Probemos si no existen errores en nuestra implementación corriendo nuevamente en la terminal el comando `sass --watch`, con el nuevo input y output.

Volvamos al directorio raíz del proyecto y luego corramos el comando.

```
cd ../..
sass --watch assets/sass/main.scss:assets/css/main.css
Sass is watching for changes. Press Ctrl-C to stop.
```

Bien, con esto hemos terminado de crear y configurar nuestro proyecto con Sass.

BEM

Ahora que hemos terminado la base de nuestro proyecto es momento de comenzar a crear la estructura HTML de la maqueta, pero antes de hacerlo conoceremos un concepto que nos ayudará a comprender a cómo nombrar nuestras clases de CSS de manera más clara y consistente.

¿Qué son las clases semánticas?

La tarea de crear clases es, aunque parezca aburrida, una práctica que debería ser tan importante como crear los estilos de la interfaz, ya que el hacerlo de una manera correcta nos ayudará a hacer más fácil la tarea de encontrar un elemento usando CSS o eventualmente el DOM en Javascript.

Nicolás Gallagher, creador de Normalize.css menciona en un artículo acerca de este tema una interesante definición acerca de las clases semánticas:

Se recomienda a los autores que utilicen valores de [atributo de clase] que describan la naturaleza del contenido, en lugar de valores que describan la presentación deseada del contenido.

En otras palabras, nosotros como maquetadores deberemos escribir clases que describan qué es objetivamente el contenido y no lo que pensamos qué es.

Otro punto interesante de este artículo de Gallagher las recomendaciones que menciona acerca de las clases.

A saber:

- La semántica de los contenidos ya se encuentra dada por la naturaleza de los elementos HTML o por otros atributos, por lo tanto podremos aprovecharlos de esos elementos para crear nuestras clases.
- Las clases nos entregan información relevante para los desarrolladores y no así para las máquinas, de modo que estas deberán ser legibles para humanos.
- El propósito de las clases es ser el ancla de CSS y Javascript. Crear clases con cuáles semánticas es clave para el correcto uso de estos selectores en CSS y JS.
- Las clases comunican información útil a desarrolladores. Escribir de buena manera clases hará que nuestros código sea escalable y fácil de entender por parte de otros desarrolladores.

Teniendo en cuenta los puntos vistos anteriormente, la forma en la que podremos llevar a la práctica la semántica de clases es creando nuestro propio sistema de clases o utilizar alguna metodología para nombrar clases.

La primera opción es algo útil si somos organizados y sabemos como presentar este sistema a otros desarrolladores, por que como sabemos nosotros no trabajaremos solos. Si por el contrario deseamos usar un sistema probado y ampliamente utilizado nos podremos decantar por el uso de metodologías de arquitectura CSS.

¿Qué son las metodologías de Arquitectura CSS?

Estas metodologías son unas guías de estilo que nos ayudarán a organizar y estructurar nuestro CSS de manera que sea fácil escalarlos y mantenerlos por otros desarrolladores.

Las metodologías que podremos usar para esta tarea son:

- **OOCSS**
 - Esta es la primera metodología creada para este fin el cual separa la estructura y el contenido de la interfaz en objetos CSS que pueden ser modificados de manera independiente.
- **SMACSS**
 - Es una guía de estilos con la que podremos escribir código CSS basados en categorías.
- **Atomic Design**
 - Es una metodología que divide los elementos de una interfaz en átomos, que luego pueden ser unidos en moléculas y organismos.
- **SUITCSS**
 - Es una guía de estilo que presenta las mejores prácticas a usar en la creación de clases y estructura CSS.
- **BEM**
 - Es una metodología que basada en componentes que nos ayudará a separar los elementos que constituyen nuestra interfaz en bloques independientes.

Nosotros, dentro de esta gran cantidad de opciones usaremos BEM para construir nuestras clases en CSS.

¿Por qué elegir BEM?

Utilizaremos esta metodología por sobre otras por tres razones:

La primera tiene que ver con la propuesta que ofrece para escribir clases, que evitan la repetición de estilos y previenen conflictos asociados a la cascada CSS.

La segunda razón es ralacionada a su popularidad y aceptación por parte de los desarrolladores. El implementarlo en nuestros proyectos nos dará un marco de trabajo que será entendido no sólo por nosotros, sino que también por otros desarrolladores que revisen nuestro código.

La última tiene que ver con la facilidad de aprender esta metodología, de modo que la curva de aprendizaje que conlleva sus bases será baja.

Teniendo clara las razones de su uso es momento de conocer las bases de BEM.

¿Qué es BEM?

Bem significa bloque, elemento y modificador y como sabemos es una metodología que nos ayudará a separar los elementos de nuestra interfaz en bloques.

Para entender como funciona BEM imaginemos que nuestra interfaz tiene un header que contiene en su interior una barra de navegación.

```
<div> <!-- Header -->

<div><!-- Navegación -->
  <ul>
    <li>Inicio</li>
    <li>Acerca de</li>
    <li>Blog</li>
  </ul>
</div><!-- fin navegación -->

</div> <!-- fin header -->
```

El `<div>` que representa a header cumple una función específica dentro de una interfaz la cual es contener todos los elementos de la cabecera del sitio. El `<div>` que contiene el listado desordenado cumple otra que es contener los elementos de la navegación.

Estos contenedores dentro de BEM son parte de una estructura conocida como bloques.

Conociendo BEM

Bloques:

Los bloques son entidades o funcionalidades independientes de un componente. Estos en HTML son representados con el atributo de clase.

Algo importante a saber sobre estos bloques es que ellos deben describir un propósito, o sea, identificar que es el elemento, asimismo este no debe ser confundido con un estado del elemento, o sea si este grande, pequeño, o tiene un color en específico.

En el ejemplo anterior, la mejor forma de nombrar a estos bloques es identificando el tipo de elemento que son.

Como vimos el primer bloque es la cabecera del sitio y su objetivo es contener a los elementos que componen a un header, por lo tanto el nombre de clase más lógico para este bloque es `class="header"`.

El segundo bloque es la navegación y cumple con el objetivo de contener los elementos que componen a una barra de navegación. En este caso el nombre clase más adecuado para este bloque es `class="navigation"`.

Algo importante a saber sobre los bloques es que estos pueden ser anidados unos con otros, de modo que podremos integrarlos sin problemas la cantidad veces que creamos necesaria.

Como resultado, la cabecera y la navegación integrado con la nomenclatura de BEM se verá de la siguiente manera:

```
<div class="header">

  <div class="navigation">
    <ul>
      <li>Inicio</li>
      <li>Acerca de</li>
      <li>Blog</li>
    </ul>
  </div>

</div>
```

Elementos:

Ahora que ya identificamos los bloques que contienen a los elementos, veamos con mayor detalle el bloque de navegación.

El bloque de navegación contiene en su interior una lista desordenada, que a su vez tiene tres ítems que componen la estructura de una barra de navegación. Estos elementos que se encuentran al interior del bloque `.navigation` forman parte de los elementos del bloque.

Los elementos son entidades que están ligadas a un bloque y que usadas por si sola no tendrán una connotación semántica. Esto quiere decir que si usamos de forma independiente estos elementos perderán su contexto semántico y por ende no formarán parte del bloque.

Para reconocerlos dentro de nuestra interfaz deberemos identificar el objetivo del bloque y no su estado. Además, para identificarlos como elementos de un bloque deberemos nombrar a la clase con el `nombre-bloque__nombre-elemento`.

En el contexto de nuestro ejemplo, los elementos dentro del bloque tendrán la siguiente nomenclatura:

```

<div class="header">

    <div class="navigation">
        <ul class="navigation_list">
            <li class="navigation_item">Inicio</li>
            <li class="navigation_item">Acerca de</li>
            <li class="navigation_item">Blog</li>
        </ul>
    </div>

</div>

```

Algo importante a saber es que los elementos que encuentren anidados dentro de otros elementos deberán mantener la nomenclatura `nombre-bloque__nombre-elemento` y no `nombre-bloque__nombre-elemento__nombre-elemento`.

Modificador:

El último elemento a tener en cuenta sobre este BEM son lo modificadores los cuales cumplen una función importante ya que definen la apariencia, estado o comportamiento de un elemento o bloque.

La sintaxis usada para estos elementos es:

Para bloques:

```

<div class="footer footer_dark-theme">
    <small class="footer__copyright">2018. Todos los derechos reservados</small>
</div>

```

Para elementos:

```

<form class="form">
    <div class="form__container">
        <input type="text" class="form__input form__input_focused">
    </div>
</form>

```

Revisemos un ejemplo:

Imaginemos que tenemos la siguiente estructura HTML con una implementación de BEM:

```

<form class="form">
    <div class="form__container">
        <input type="text" class="form__input">
        <input type="submit" value="Enviar" class="form__button">
    </div>
</form>

```

Si quisieramos cambiar el tamaño del botón por defecto por uno más pequeño deberíamos, en primer lugar, identificar donde se encuentra el elemento, que en este caso sería el `<input>` del tipo `type="submit"`. Aquí deberíamos agregar este modificador de la siguiente manera:

```

<input type="submit" value="Enviar" class="form__button nombre-
bloque__nombre-elemento-clave_valor">

```

En este contexto, como deseamos que el tamaño del bloque sea de tamaño pequeño la implementación de este nuevo tamaño podría ser:

```
<input type="submit" value="Enviar" class="form__button form__button_size-s">
```

Como vemos implementar BEM en nuestros proyectos es fácil y no requiere de mayor aprendizaje del que hemos visto hasta el momento.

Creando HTML de proyecto usando BEM

Ahora que conocemos los conceptos claves de BEM es momento de aplicarlos en nuestro proyecto.

Para hacerlo debemos ingresar a nuestro HTML(`index.html`) y comenzar a crear la estructura de contenido de la maqueta.

Primero que nada creamos la base del proyecto HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>

</body>
</html>
```

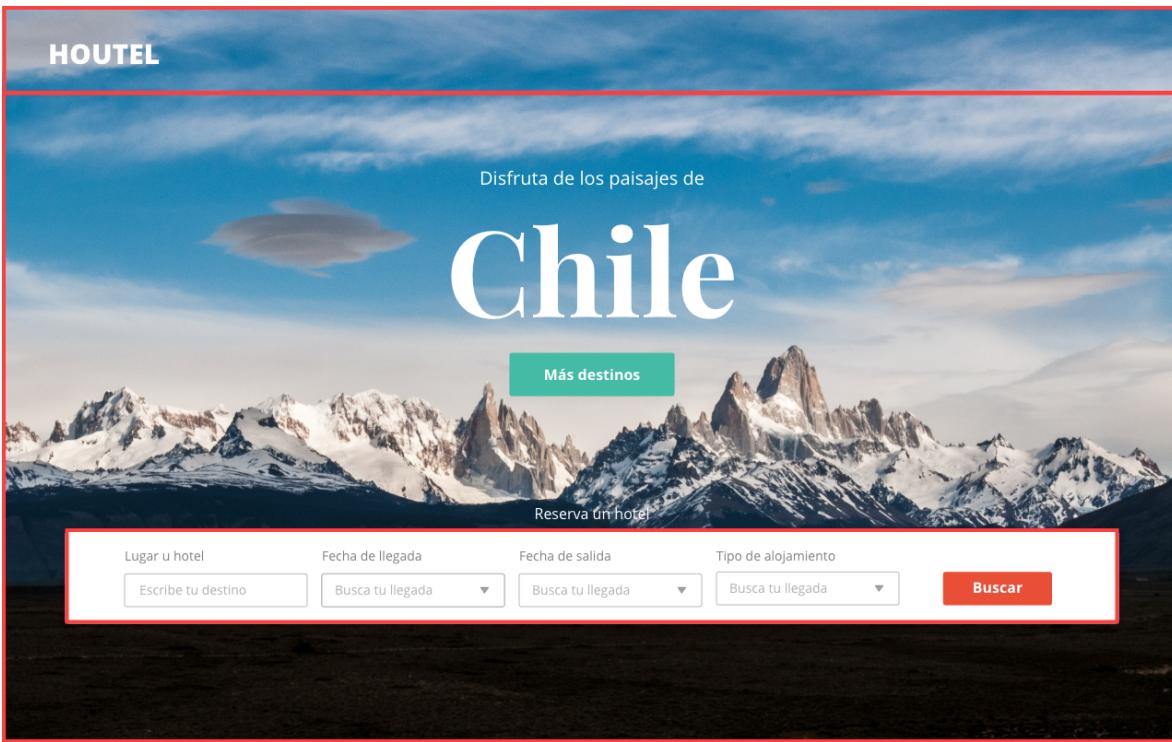
Luego agreguemos el archivo CSS compilado a nuestro HTML usando la etiqueta `<link>`.

```
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <link rel="stylesheet" href="assets/css/main.css">
  <title>Document</title>
</head>
<body>

</body>
</html>
```

Ahora que ya tenemos lista la base de nuestro HTML, comencemos a definir el layout.

Revisemos el mockup que nos entrega Houstel:



Este esta constituido por tres bloques principales. El primero es la `cabecera` del sitio, le sigue `contenido principal` y finalmente el `formulario de búsqueda`.

Estos elementos los definiremos en la hoja HTML usando algunas etiquetas conocidas como "etiquetas semánticas".

¿Qué son las etiquetas semánticas?

Las etiquetas semánticas son elementos HTML que describen el significado semántico de un elemento en específico.

Por ejemplo, es más fácil identificar una etiqueta `<header>` que una etiqueta `<div>` dentro del contexto de cabecera.

Las etiquetas semánticas más utilizadas son:

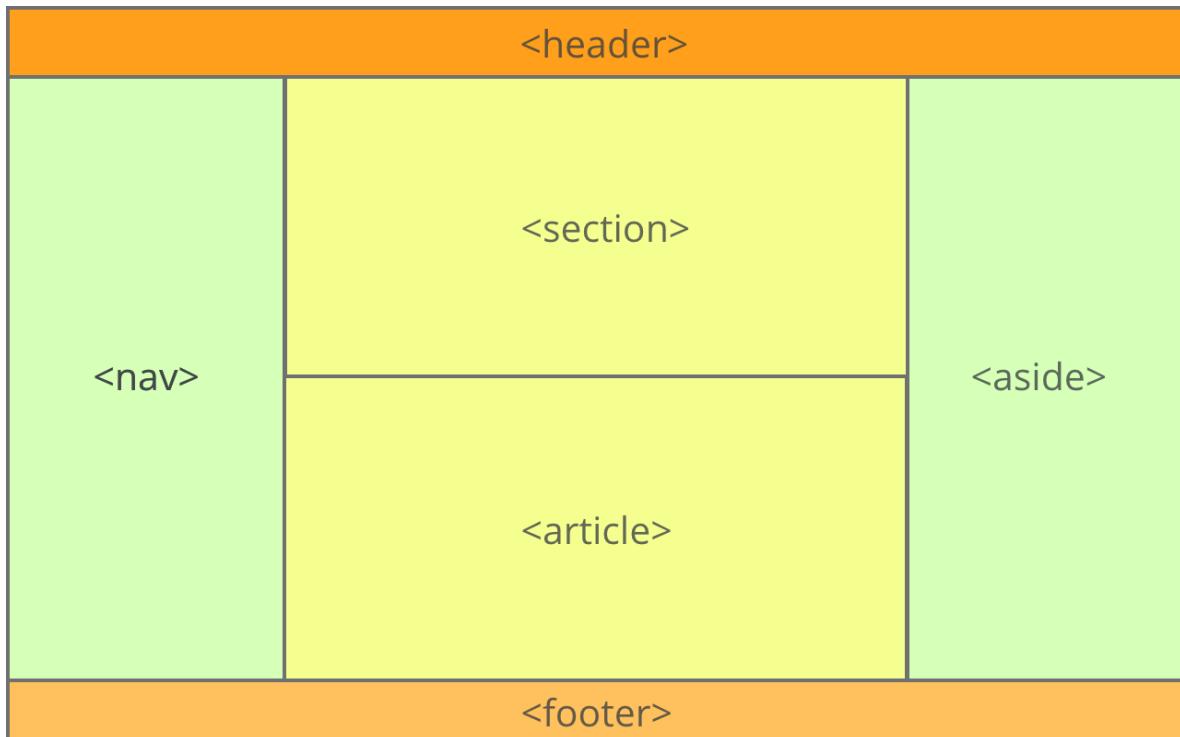


Imagen: Santo grial del diseño web.

`<header>` : Etiqueta que define la cabecera.

`<nav>` : Etiqueta que define la navegación de un sitio.

`<main>` : Etiqueta que define el contenido principal de nuestro sitio.

`<section>` : Etiqueta que define a una sección.

`<article>` : Etiqueta que define el contenido de un artículo.

`<aside>` : Etiqueta que define la página lateral de un contenido.

`<footer>` : Etiqueta que define el pie de página del sitio o de una sección.

Usando las etiquetas semánticas en el proyecto

Ahora que conocemos que son las etiquetas semánticas es momento de aplicarlas dentro de nuestro proyecto.

Header:

Comencemos agregando la etiqueta `<header>` al interior del `<body>`. Como ya sabemos el contexto que representa este elemento agregaremos la clase llamada `class="header"`.

```
<body>
  <header class="header">
    </header>
  </body>
```

Main:

El siguiente paso es hacer lo mismo pero con el contenido principal. Para este caso la etiqueta más adecuada es `<main>` ya que en el se encuentra todo el contenido central del sitio. En cuanto a la clase, como sabemos que el contenido es el principal usaremos el mismo nombre semántico como nombre la clase.

```
<main class="main">  
  
</main>
```

Form:

El último paso es crear la estructura del formulario. Para los formularios existe una etiqueta de HTML especial para este tipo de contenido llamado `<form>`. Siguiendo el mismo patrón la clase del bloque que crearemos se llamará `class="form"`.

```
<form class="form">  
  
</form>
```

Aplicando elementos dentro del proyecto

Ahora que ya tenemos la estructura HTML lista es momento de integrar los elementos que componen a los bloques.

Header:

Revisemos el contenido del header.



Como podemos ver el único elemento que integra al bloque header es el logo.

Para integrarlo usaremos un `<div>` que contendrá el logo y una etiqueta `` para agregar nuestro logo.

```
<header class="header">  
  <div>  
      
  </div>  
</header>
```

En cuanto a la nomenclatura que usaremos para el contenedor y la imagen, el logo será una entidad a parte la cuál podremos reutilizar a lo largo de la página, de modo que el contenedor del logo será el bloque y la imagen será el elemento que compone al logo.

Esta lógica al principio puede sonar extraña, pero si vemos el objetivo que ejerce un logo dentro de un diseño y los lugares en los que podremos aplicarlo, veremos que entenderlo como bloque es una forma fácil de reutilizar este elemento.

Por lo tanto, lo lógico para este caso es utilizar la clase `class="logo"` como bloque y el elemento `` como `class="logo__image"`.

```
<header class="header">
  <div class="logo">
    
  </div>
</header>
```

Algo que no podemos olvidar del logo es el hecho de agregarlo. Copiemos y peguemos este en la carpeta destinada para las imágenes.

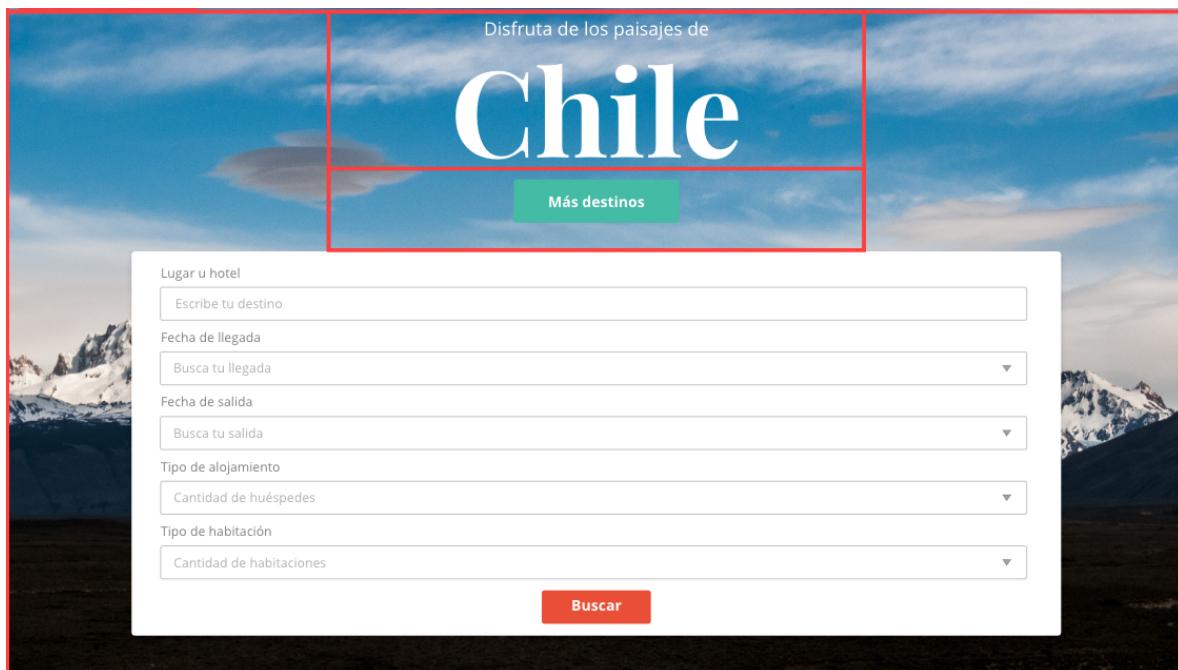
Ahora agreguemos su ruta:

```
<header class="header">
  <div class="logo">
    
  </div>
</header>
```

Main:

Luego de finalizar la estructura de header seguiremos con los elementos que componen main.

Los elementos que componen main se pueden dividir en:



- Un contenedor de los elementos de main.
 - Para este contenedor usaremos en etiqueta `<div>`.
- Un título que contiene la información principal del sitio. Algo a tener en cuenta de este título es que existe una palabra en su interior que tiene un mayor énfasis visual que otras partes del título, sin embargo el carácter como contenido sigue siendo el mismo.
 - En este caso usaremos un `<h1>` para el título y para dar el énfasis visual a la palabra chile agregaremos un ``.
- Y finalmente nos encontraremos con un botón.
 - Este tendrá una etiqueta `<button>` al que daremos un carácter de botón utilizando el atributo `type="button"`.

```

<main>
  <div>
    <h1>Disfruta de los paisajes del
      <span>
        Chile
      </span>
    </h1>
    <button type="button">Más destinos</button>
  </div>
</main>

```

Ahora agreguemos las clases que definirán semánticamente a nuestros elementos.

- El primer elemento como elemento contenedor tendrá una nomenclatura `class="main_container"`.
- En cuanto al título, el elemento `<h1>` tendrá una clase `class="main_title"` y el `` el cual le daremos énfasis tendrá dos clases al interior, una relacionada al elemento y otra que será el modificador de este. El elemento como tal tendrá clase `class="main_title-emphasis"` y el modificador que le dará el tamaño se llamará `class="main_title-emphasis main_title-emphasis_size-xxl"`.
- Finalmente el botón que invita al usuario a conocer más destinos le daremos una identidad de bloque, debido a que lo reutilizaremos dentro de la creación de nuestra maqueta. Asimismo crearemos un contenedor afuera de el a modo de posicionar el botón dentro de contenedor main.
 - Si vemos la guía de estilos podremos ver que botón tiene dos estados, uno llamado default de color rojo y positive de color verde. El color del botón es un estado que debemos definir como modificador.
 - Teniendo claro los dos contextos del botón podemos inferir que la clase que define el bloque será `class="button"` y el modificador que da el estado del botón siendo verde será llamado `class="button button_positive"`.
 - En cuanto al contenedor, este será un elemento del bloque main de modo que su nombre de clase será `class="main_button"`

```

<main class="main">
  <div class="main_container">
    <h1 class="main_title">Disfruta de los paisajes de
      <span class="main_title-emphasis main_title-emphasis_size-xxl">
        Chile
      </span>
    </h1>
    <div class="main_button">
      <button type="button" class="button button_positive">Más
      destinos</button>
    </div>
  </div>
</main>

```

Form:

La última tarea que debemos hacer en nuestra hoja HTML es identificar los elementos del formulario.

Si lo vemos de cerca, podremos identificar los siguiente elementos:

- El formulario está constituido por el bloque form.
- Este bloque tiene a su vez unos contenedores que separan el contexto de cada input del formulario.
- Dentro de estos contenedores podremos encontrar diversos elementos.
 - En el primer contenedor hay un label con un input de texto.
 - El segundo es otro label con un input tipo date.
 - El tercero tiene la misma estructura.
 - Es importante acotar que los input tipo date que usaremos en este caso práctico no son soportados por todos los navegadores. De hecho este input en navegadores como Safari e Internet Explorer se verán como inputs de texto.
 - El cuarto tiene un label y una etiqueta <select> que en su interior contiene opciones.
 - Estos opciones podremos definirlos como deseemos en este caso práctico.
 - Por ejemplo podríamos usar como opción un hotel, una hostal y finalmente un departamento.
 - El quinto tiene la misma estructura que el cuarto de modo que crearemos la misma estructura.
 - Los option que agregaremos serán: 1, 2, 3, + de 4.
- El último elemento del formulario es un botón.

```
<form class="form">
  <div>
    <label>Lugar u hotel</label>
    <input type="text" placeholder="Escribe tu destino">
  </div>
  <div>
    <label>Fecha de llegada</label>
    <input type="date">
  </div>
  <div>
    <label>Fecha de llegada</label>
    <input type="date">
  </div>
  <div>
    <label>Huéspedes</label>
    <select>
      <option value="1">Hotel</option>
      <option value="2">Hostel</option>
      <option value="3">Departamento</option>
    </select>
  </div>
</form>
```

```

        </select>
    </div>
    <div>
        <label>Habitaciones</label>
        <select>
            <option value="1">1 habitación</option>
            <option value="2">2 habitaciones</option>
            <option value="3">3 habitaciones</option>
            <option value="4">+ de 4</option>
        </select>
    </div>
    <button type="submit">Buscar</button>
</form>

```

Ahora sólo nos falta darles nombres a las clases de los elementos del formulario:

- Comencemos con los contenedores.
 - Siguiendo la lógica usada de BEM estos deberían tener una clase `class="form__container"`.
- Los elementos label tendrán una clase `class="form__label"`.
- Sigamos con los inputs. Lo inputs al igual que los demás elementos forman parte de bloque form, de modo que estos solamente deberán reflejar el objetivo que cumplen.
 - En el primer caso la clase de este elemento podría ser `class="form__input-text"`.
 - El segundo caso como `class="form__input-date"`.
 - En el tercero a modo de reutilizar los estilos del input date usaremos la misma clase `class="form__input-date"`.
- Para los dos elementos select usaremos la clase `form__select`.
- Finalmente, para el botón usaremos la misma clase usada en el botón de arriba (`class="button"`) y además incluiremos un estado específico a ese botón llamado default, de modo que este deberá llamarse `class="button_default"`.

```

<form class="form">
    <div class="form__container">
        <label class="form__label">Lugar u hotel</label>
        <input class="form__input-text" type="text" placeholder="Escribe tu destino">
    </div>
    <div class="form__container">
        <label class="form__label">Fecha de llegada</label>
        <input class="form__input-date" type="date" class="">
    </div>
    <div class="form__container">
        <label class="form__label">Fecha de llegada</label>
        <input class="form__input-date" type="date" class="">
    </div>
    <div class="form__container">
        <label class="form__label">Huéspedes</label>
        <select class="form__select" name="kjlkd">
            <option value="1">Hotel</option>
            <option value="2">Hostel</option>
            <option value="3">Departamento</option>
        </select>
    </div>
    <div class="form__container">

```

```
<label class="form__label">Habitaciones</label>
<select class="form__select" name="">
  <option value="1">1 habitación</option>
  <option value="2">2 habitaciones</option>
  <option value="3">3 habitaciones</option>
  <option value="4">+ de 4</option>
</select>
</div>
<button type="submit" class="button button_default">Buscar</button>
</form>
```

Con esto hemos terminado de estructurar e implementar BEM dentro de nuestra hoja HTML.

Usando variables de Sass en un proyecto

Ahora que hemos terminado la estructura de nuestro HTML es momento de traspasar la información de la guía de estilos hacia nuestro proyecto.

En Sass tendremos una herramienta muy útil para realizar esta tarea llamada variable.

¿Qué son las variables de Sass?

Las variables de Sass nos ayudarán a almacenar información relevante que deseemos reutilizar en nuestras hojas de estilo. En las variables podremos almacenar colores, fuentes, tamaños o cualquier valor de CSS. A su vez, para que una variable pueda ser una variable en Sass está deberá comenzar con un signo de dolar (\$) y deberá tener un nombre que represente el contenido de la variable.

Para usarlas deberemos escribir primero el signo dolar, seguido por el nombre de la variable y el valor que queramos asignar.

```
$nombre-variable: valor;
```

Usando variables en proyecto

Teniendo esto claro esto es momento de traspasar toda la información contenida dentro de la guía de estilos a nuestro proyecto.

Primero ingresemos a la carpeta Sass y luego a "abstracts". Aquí encontraremos el archivo que creamos hace un rato llamado `variables.scss`. En este archivo incluiremos toda la información de los estilos que podamos reutilizar a lo largo de la creación de nuestra maqueta.

Revisemos la guía de estilos para encontrar información útil acerca de los estilos de la interfaz.

Comencemos revisando el esquema de colores:

Creando variables de color

Colors Scheme

White #FFF	Carmine pink #E94F37	Black olive #393E41	Verdigris #44BBA4
Platinum #E8E8E8	Salmon #EF7F6D	Granite gray #5D6163	Medium aquamarine #66C7B4
Gray #BABABA	Spanish pink #F7FBF6	Old silver #818486	Perl Aqua #88D3C5

Este esquema está definido por 4 colores principales. Cada uno de estos colores tiene dos tipos difuminaciones distintas. Así mismo dentro de los colores podremos ver el nombre que representa a su color y el valor hexadecimal que tiene.

Esta información nos será útil para organizar los esquemas de colores, definir el nombre y valor de las variables a usar.

Ingresemos nuevamente a `variables.scss` y agreguemos el primer esquema de colores.

Primero agreguemos un comentario indicando que aquí irán los colores. Luego creemos otro enunciando al primer esquema.

```
// Color Scheme  
  
// First Color Scheme
```

La primera variable a crear sera el blanco. Esta deberá comenzar con el simbolo dolar, seguido por el nombre del color que en este caso es `white` y el valor hexadecimal.

```
$white: #FFF;
```

Sigamos con las difuminaciones del color principal:

La variable del primer color es `$platinum` y el valor es `#E8E8E8`.

```
$platinum: #E8E8E8;
```

La segunda variable es `$gray` y el valor es `#bababa`.

```
$gray: #BABABA;
```

Al terminar veremos el primer esquema de colores.

```
// Color Scheme

// First Color Scheme
$white: #FFF;
$platinum: #E8E8E8;
$gray: #BABABA;
```

Para los siguientes esquemas deberemos seguir los mismos pasos.

Cuando terminemos los esquemas de colores se verán así:

```
// Color Scheme

// First Color Scheme
$white: #FFF;
$platinum: #E8E8E8;
$gray: #BABABA;

// Second Color Scheme
$carmine-pink: #E94F37;
$salmon: #EF7F6D;
$spanish-pink: #F7BFB6;

// Third Color Scheme
$black-olive: #393E41;
$granite-gray: #5D6163;
$old-silver: #818486;

// Fourth Scheme
$verdigris: #44BBA4;
$medium-aquamarine: #66C7B4;
$perl-aqua: #88D3C5;
```

Creando variables para familia De Fuentes

Con esto listo sigamos traspasando la infomación de la guía de estilos.

Typography

Typefaces

Playfair Display Bold

Open Sans Bold
Open Sans Semi Bold
Open Sans Regular

En la sección "typography" dentro de "Typefaces" encontraremos las tipografías usadas en el mockup y el peso que tiene cada una de ellas.

la tipografías usadas podremos transformarlas en variables a modo usarlas en diferentes lugares.

Antes de hacerlo es una buena idea buscar las tipografías utilizadas en la representación visual.

Las fuentes Playfair Display y Open Sans forman parte del repositorio de fuentes de Google Fonts, de modo que debemos ingresar a este sitio para usarlas.

Ingresemos a Google fonts. Luego busquemos la primera tipografía llamada "Playfair Display".

Cuando la encontramos presionemos en el botón  para agregarla.

Ahora busquemos "Open Sans" y hagamos el mismo procedimiento.

Cuando terminemos debemos presionar en el recuadro que dice "2 familias seleccionadas". Ya en su interior presionaremos en el apartado "customize", en donde elegiremos los pesos de las fuentes.

Este peso hace referencia al grosor que tiene una fuente específica.

Seleccionemos:

- **Open sans:** Regular 400, semi-bold 600, bold 700.
- **Playfair Display:** Bold 700.

Volvamos a la pestaña "embed" y copiemos el elemento link y luego peguemos este en `index.html`, al interior de `<head>`.

```
<link href="https://fonts.googleapis.com/css  
family=Open+Sans:400,600,700|Playfair+Display:700" rel="stylesheet">
```

Ahora que ya está listo este paso agreguemos los nombres de tipografías descritas en Google Font.

La primera variable que crearemos se llamará `$playfair` y tendrá el valor `'Playfair Display', serif;`

La segunda variable será `$open-sans`. Esta tendrá un valor de `'Open Sans', sans-serif;`.

```
//Font Family  
$playfair: 'Playfair Display', serif;  
$open-sans: 'Open Sans', sans-serif;
```

Creando variables para peso de tipografía

Ahora es turno de agregar los pesos de las tipografías. Volvamos a nuestra hoja de estilos e ingremos en ella las variables de peso.

Según la guía de estilos la primer peso es regular. Este tipo de grosor por convención tiene un número 400.

```
// Tipography  
  
// Font Weight  
$regular: 400;
```

El segundo peso mencionado en la guía de estilos es el semi bold. Este tiene un valor de 600.

```
$semi-bold: 600;
```

Finalmente, el último peso a agregar es bold. Este tiene un valor de 700.

```
$bold: 700;
```

Al terminar tendremos todos los valores de peso de tipografías.

```
// Tipography  
  
// Font Weight  
$regular: 400;  
$semi-bold: 600;  
$bold: 700;
```

Creando variables para tamaño de fuentes

Los última información que agregaremos desde la guía de estilo hacia `variables.scss` son los tamaños de las tipografías.

Font Sizes

My slave

130px — xxxx-large

My slave human

100px — xxx-large

My slave human didn't give me any food

30px — xx-large

My slave human didn't give me any food

20px — x-large

My slave human didn't give me any food

18px — large

My slave human didn't give me any food

16px — medium

My slave human didn't give me any food

14px — small

My slave human didn't give me any food

12px — x-small

Estos tamaños se pueden ver ejemplificados en el texto de ejemplo usando en la guía de estilo. Más abajo se encuentra una leyenda con el tamaño en píxeles de la fuente y un nombre de referencia para el tamaño.

Aplicar este tamaño de fuente como variable es bastante sencillo, pues deberemos hacer el mismo procedimiento realizado con las otras variables escritas anteriormente:

- La primera variable se llamará xxxx-large y tendrá un tamaño de `130px`.
- La segunda es xxx-large, con valor de `100px`.
- Despues viene xx-large, con valor de `30px`.
- Sigue x-large, con `20px`.
- Luego large, con `18px`.
- Sigue medium, con `16px`.
- Small, con `14px`.
- Y finalmente x-small, con un valor de `12px`.

```
// Font Sizes
$xxxx-large: 130px;
$xxx-large: 100px;
$xx-large: 30px;
$x-large: 20px;
$large: 18px;
$medium: 16px;
$small: 14px;
$x-small: 12px;
```

Con esto hemos terminado de traspasar las variables que usaremos para trabajar en nuestro proyecto.

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.

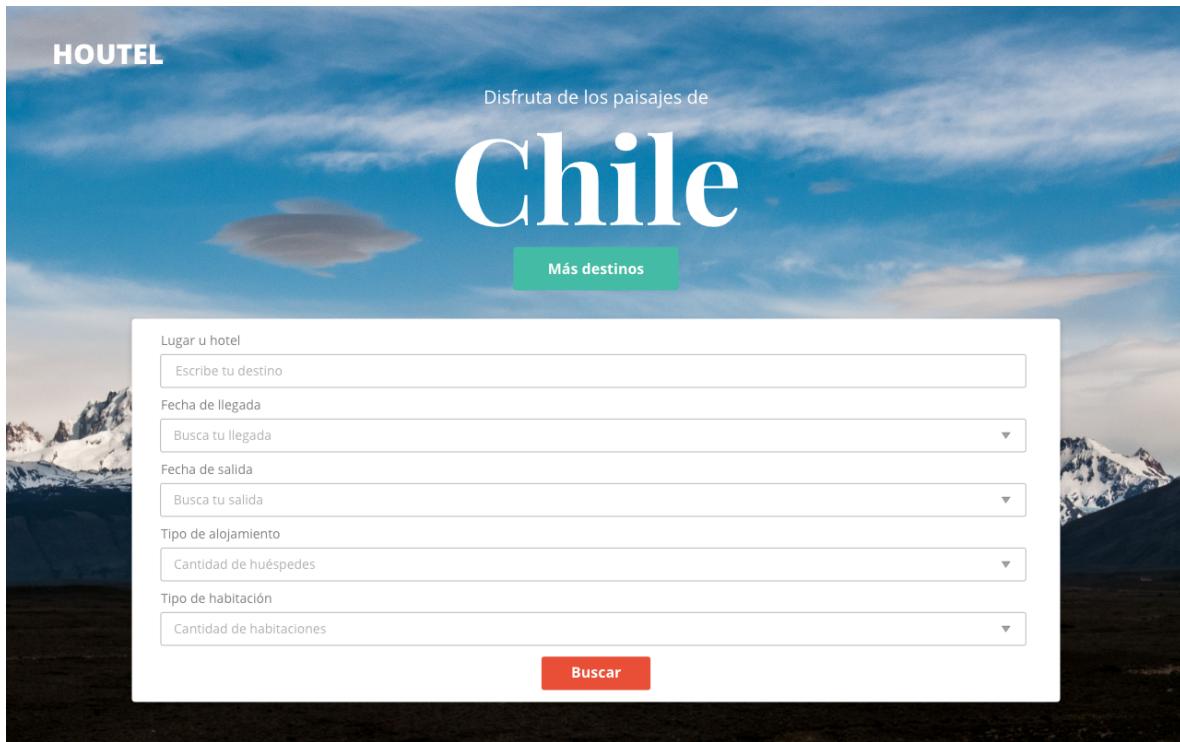
Creando estilos del layout del proyecto

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:



- El sector del header tiene espacios a los lados que limita al logo y hace que este no tope con 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`

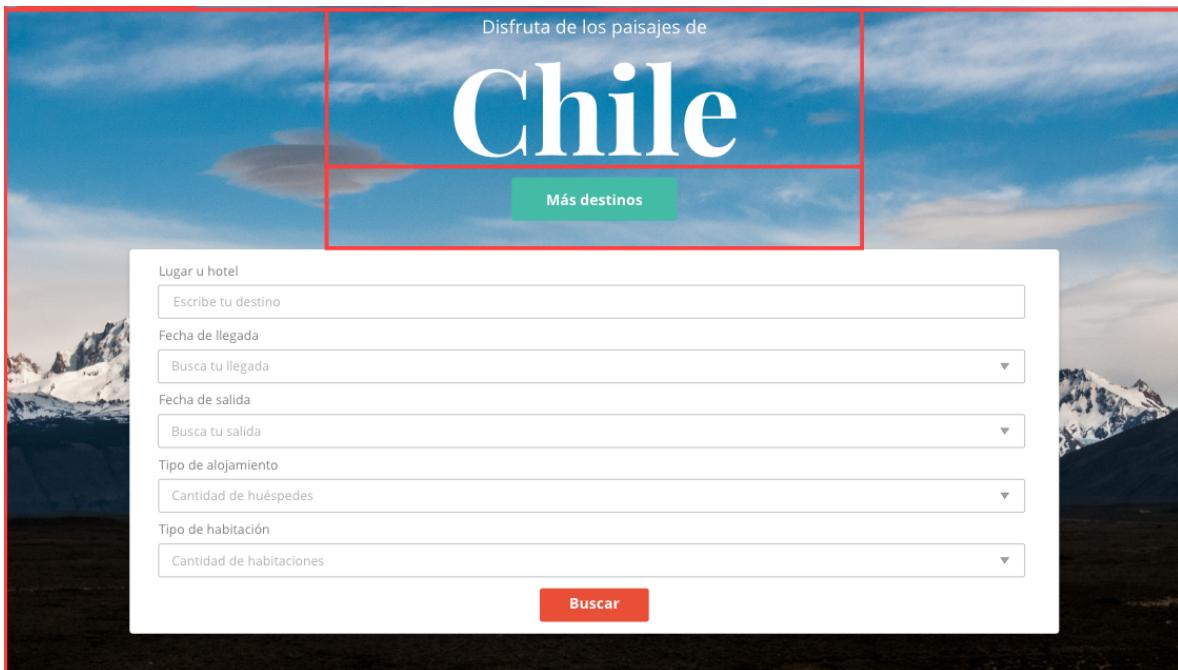
```
/* Header Layout */
.header {
    padding-top: 2rem;
    padding-left: 2rem;
}

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

Recarguemos la página para ver las nuevos estilos.

Creando los estilos de main

Ahora sigamos con los estilos del contenido principal.



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.

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;
}

```

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



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 ampersand 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 `$open-sans`. Demos un tamaño mínimo de letra (`$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`.

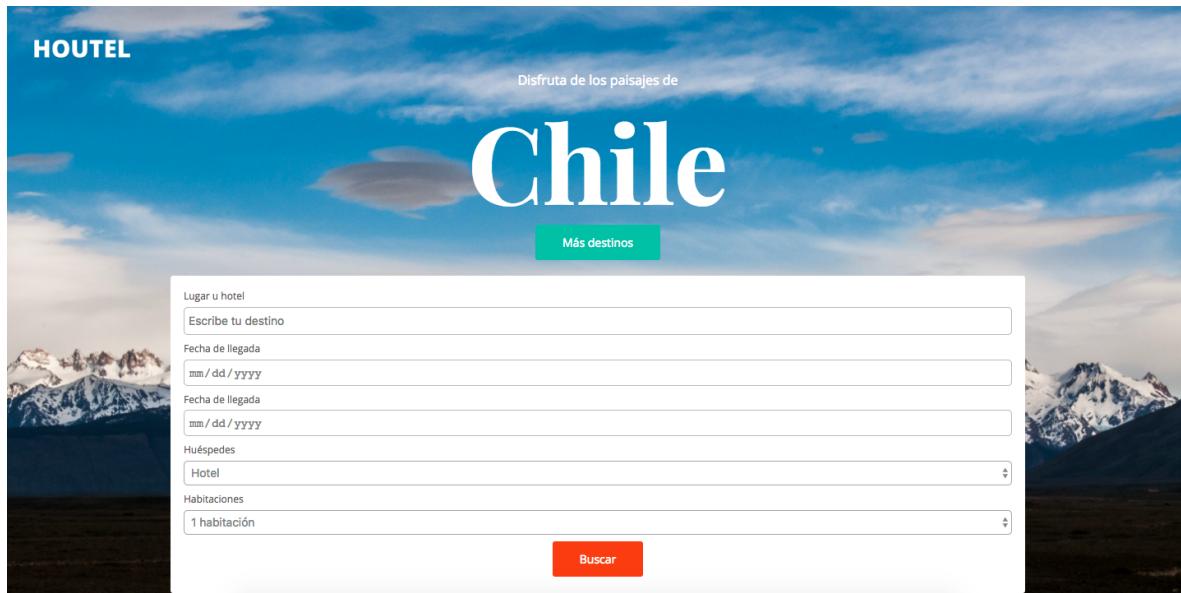
Redondeemos los inputs con un `border-radius: 5px;` y luego ajustemos el borde con un `border: 1px solid $gray;`

```
.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:



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

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 incluídos se verán a todo color en la página.



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.



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 ya visto.