← Volver a la lista de capítulos

Cómo organizar archivos CSS en Ruby on Rails

Publicado el 19 de enero de 2022

En este capítulo, escribiremos algunos CSS utilizando la metodología BEM para crear un buen sistema de diseño para nuestra aplicación.

Creando un sistema de diseño para nuestra aplicación Rails

Incluso si este es un tutorial de Turbo Rails, esto no es motivo para crear un editor de citas feo.

No sabía qué biblioteca usar para este tutorial aunque soy un gran fanático de CSS:

- ¿Debería utilizar Bootstrap y no centrarme en absoluto en el diseño?
- ¿Debería usar Tailwind y perder a todos los lectores a quienes no les gusta el marcado *feo* que es consecuencia del enfoque de utilidad primero?

Sabía que crearía un sitio web del curso para mostrar el editor de citas, así que pensé: **¿Por qué no crear un sistema de diseño para la plataforma del curso y el editor de citas y mostrar cómo lo hice** ?

Eso es lo que terminé haciendo, y por eso, en este curso, escribiremos nuestro propio CSS desde cero con la metodología BEM. Si te gusta el CSS, esta es una excelente oportunidad para aprender algunos trucos. Si no te gusta el CSS, ¡siéntete libre de copiar y pegar los fragmentos de código a medida que avanzan y pasar al siguiente capítulo!

Escribiremos la mayor parte del CSS que necesitamos en el tutorial de este capítulo y no te preocupes, ¡comenzaremos a trabajar con Turbo en el próximo capítulo!

Nuestra arquitectura CSS

CSS es un tema que suele malinterpretarse. Como cualquier otro tipo de código, necesita una arquitectura y algunas convenciones para que sea un placer trabajar con él. La mejor forma de aprender es escribir CSS desde cero para esta sencilla aplicación.

La metodología BEM

Para las convenciones de nomenclatura utilizaremos la metodología BEM . Es sencilla y la podemos resumir en **solo tres reglas** .

1. Cada componente (o *bloque*) debe tener un nombre único. Imaginemos un componente de tarjeta en nuestra aplicación. La `.card` clase CSS debe estar definida en el `card.scss` archivo y su nombre debe ser único. Esa es la letra *B* en "BEM", y significa *bloque* .
2. Cada *bloque* puede tener muchos *elementos* . Si volvemos a nuestro ejemplo de la tarjeta, cada tarjeta puede tener un título y un cuerpo. En BEM nombraremos las clases CSS correspondientes `.card__title` y `.card__body` . Nos aseguramos de que no sea posible que haya conflictos de nombres al anteponer el nombre del elemento con el nombre del bloque. Si tenemos otro bloque `.box` con un título y un cuerpo, `.box__title` y `.box__body` no entrará en conflicto con `.card__title` y, `.card__body` por lo tanto, nos aseguramos de que no haya ningún conflicto entre esas clases. Esa es la letra *E* en "BEM", y significa *elemento* .
3. Cada bloque puede tener modificadores. Si volvemos a nuestro ejemplo de cartas, tal vez puedan tener diferentes colores. Los nombres de los modificadores podrían ser `.card--primary` y `.card--secondary` . Esa es la letra *M* en "BEM", y significa *modificadores* .

Con esta sencilla metodología, nombrar se vuelve muy fácil y nos evita tener que lidiar con uno de los aspectos más frustrantes de CSS: **los conflictos de nombres** .

Organizando nuestros archivos CSS

Ahora que tenemos una convención de nombres sólida, es hora de hablar sobre la organización de archivos. Como esta aplicación es bastante simple, tendremos una arquitectura simple.

Nuestra `app/assets/stylesheets/` carpeta contendrá cuatro elementos:

- El `application.sass.scss` archivo de manifiesto para importar todos nuestros estilos.
- Una `mixins/` carpeta donde agregaremos mixins de Sass.
- Una `config/` carpeta donde agregaremos nuestras variables y estilos globales.
- Una `components/` carpeta donde agregaremos nuestros componentes.
- Una `layouts/` carpeta donde agregaremos nuestros diseños.

¿Cuál es la diferencia entre componentes y diseños?

Los componentes son piezas independientes de la página web . No deben preocuparse por cómo se presentan en la página, sino solo por su estilo. Un excelente ejemplo de componente es un botón. Los botones no saben dónde se colocarán en la página.

Un diseño, por el contrario, no añade estilo . Solo se ocupa de los márgenes, el centrado o cualquier cosa que nos ayude a posicionar los componentes entre sí. Un buen ejemplo de un diseño es un contenedor que centrará el contenido en la página. Si tienes curiosidad, el libro Every layout muestra una lista de diseños que utilizan casi todas las aplicaciones web y sus CSS asociados.

Una vez que hayamos construido nuestro sistema de diseño, ¡podremos crear nuevas páginas en poco tiempo sin necesidad de escribir CSS adicional componiendo componentes y diseños!

Nota sobre componentes y márgenes :

En teoría, como los componentes solo se preocupan por sí mismos, **no deberían tener márgenes exteriores** . Cuando diseñamos componentes individuales, no sabemos dónde terminarán ubicados en una página. Tomemos el ejemplo de un botón. No tendría sentido agregar márgenes a los botones, ya que a veces se colocarán *horizontalmente* y a veces se colocarán *verticalmente* , con más o menos espacio entre ellos.

No sabemos de antemano en qué *contexto* se utilizarán los componentes individuales. Eso es responsabilidad de los diseños. A medida que un sistema de diseño crece, será mucho más fácil reutilizar los componentes si es fácil hacer que colaboren entre sí. Será mucho más fácil si esos componentes no tienen márgenes externos.

Dicho esto, en este tutorial romperemos esta regla. Agregaremos márgenes directamente en los componentes mismos. Por ejemplo, la `.quote` clase CSS tendrá un margen inferior. En la vida real, es posible que queramos delegar esta responsabilidad a una utilidad de margen o un `.stack` diseño para facilitar la reutilización del `.quote` componente en otros *contextos* .

Como esta aplicación no evolucionará y no quería complicar demasiado las cosas, opté por romper la *regla de no dejar margen externo en los componentes* . Sin embargo, es una regla excelente para tener en cuenta la próxima vez que crees un sistema de diseño real.

Basta de teoría; ahora estamos listos para escribir un código SASS fantástico. ¡Practiquemos!

Usando nuestra arquitectura CSS en nuestro editor de cotizaciones

La carpeta mixins

La carpeta mixins es la más pequeña de todas. Solo contendrá un archivo llamado `_media.scss` en el que definiremos los puntos de interrupción de nuestras consultas de medios. En nuestra aplicación, solo tendremos un punto de interrupción al que llamaremos `tabletAndUp` :

```
// app/assets/stylesheets/mixins/_media.scss

@mixin media($query) {
  @if $query == tabletAndUp {
    @media (min-width: 50rem) { @content; }
  }
}
```

Analicemos juntos este fragmento de código. Al escribir CSS, **primero escribiremos el CSS para dispositivos móviles** :

```
.my-component {
  // The CSS for mobile
}
```

Entonces, es posible que queramos tener algunas **modificaciones para tabletas y tamaños de pantalla más grandes** . En ese caso, usar nuestra consulta de medios hace que las cosas sean muy fáciles:

```
.my-component {
  // The CSS for mobile

  @include media(tabletAndUp) {
    // The CSS for screens bigger than tablets
  }
}
```

Esto es lo que llamamos ***enfoque móvil primero*** . Primero, definimos nuestro CSS para los tamaños de pantalla más pequeños (móviles), luego agregamos anulaciones para tamaños de pantalla más grandes. Es una buena práctica usar este tipo de combinación en nuestro código Sass. Si más adelante queremos agregar más puntos de interrupción, como `laptopAndUp` o `desktopAndUp` , se vuelve muy fácil hacerlo.

También aclara a qué corresponde el punto de interrupción, ya que `tabletAndUp` es más fácil de leer que `50rem` .

También nos ayuda a mantener nuestro código DRY al evitar repetir el `50rem` número mágico en todas partes:

```
.component-1 {
  // The CSS for mobile

  @media (min-width: 50rem) {
    // The CSS for screens bigger than 50rem
  }
}

.component-2 {
  // The CSS for mobile

  @media (min-width: 50rem) {
    // The CSS for screens bigger than 50rem
  }
}
```

Imaginemos que queremos cambiar `50rem` en `55rem` algún momento. Esto sería una pesadilla de mantenimiento, ya que tendríamos que cambiar el valor en todos los componentes.

Por último, pero no menos importante, tener una lista seleccionada de puntos de interrupción en un solo lugar nos ayuda a elegir el más relevante y limitar nuestras opciones.

Este primer archivo CSS tiene mucho que decir, pero es realmente útil. Nuestro editor de citas debe ser responsivo y esta es una implementación simple pero poderosa para los puntos de interrupción.

La carpeta de configuración

Uno de los archivos más importantes es el archivo de variables para crear un sistema de diseño sólido. Aquí es donde elegiremos colores bonitos, fuentes legibles y crearemos una escala de espacios para garantizar un espaciado uniforme.

Crear un archivo de variables puede parecer un poco mágico, pero *existen reglas* como la *regla del múltiplo de 8* para los espaciados y los tamaños de fuente que nos ayudan a crear el archivo de variables. Para los tamaños de fuente y los espacios, casi siempre uso las mismas escalas.

Al crear un sitio web desde cero, a menudo elijo una "personalidad" que guiará la elección de fuentes y colores.

Ya basta de hablar de diseño; ¡comencemos a construir este archivo variable!

Primero, comencemos con todo lo que necesitamos para nuestro diseño de texto, como familias de fuentes, colores, tamaños y alturas de línea.

```
// app/assets/stylesheets/config/_variables.scss

:root {
  // Simple fonts
  --font-family-sans: 'Lato', -apple-system, BlinkMacSystemFont, 'Segoe

  // Classical line heights
  --line-height-headers: 1.1;
  --line-height-body:    1.5;

  // Classical and robust font sizes system
  --font-size-xs: 0.75rem;   // 12px
  --font-size-s: 0.875rem;   // 14px
  --font-size-m: 1rem;       // 16px
  --font-size-l: 1.125rem;   // 18px
  --font-size-xl: 1.25rem;   // 20px
  --font-size-xxl: 1.5rem;   // 24px
  --font-size-xxxl: 2rem;    // 32px
  --font-size-xxxxl: 2.5rem; // 40px

  // Three different text colors
  --color-text-header: hsl(0, 1%, 16%);
  --color-text-body:   hsl(0, 5%, 25%);
  --color-text-muted:  hsl(0, 1%, 44%);
}
```

Este primer conjunto de variables nos ayudará a garantizar que nuestro diseño de texto sea consistente en toda nuestra aplicación.

También es esencial mantener espacios uniformes para los rellenos y los márgenes en la aplicación para lograr un buen diseño. Construyamos una escala de espacios simple que podamos usar en nuestra aplicación en todas partes:

```
// app/assets/stylesheets/config/_variables.scss

:root {
  // All the previous variables

  // Classical and robust spacing system
  --space-xxxs: 0.25rem; // 4px
  --space-xxs: 0.375rem; // 6px
  --space-xs: 0.5rem;    // 8px
  --space-s: 0.75rem;    // 12px
  --space-m: 1rem;       // 16px
  --space-l: 1.5rem;     // 24px
  --space-xl: 2rem;      // 32px
  --space-xxl: 2.5rem;   // 40px
  --space-xxxl: 3rem;    // 48px
  --space-xxxxl: 4rem;   // 64px
}
```

Para realizar nuestro diseño también necesitaremos colores, por supuesto.

```
// app/assets/stylesheets/config/_variables.scss

:root {
  // All the previous variables

  // Application colors
  --color-primary:          hsl(350, 67%, 50%);
  --color-primary-rotate:   hsl(10, 73%, 54%);
  --color-primary-bg:       hsl(0, 85%, 96%);
  --color-secondary:        hsl(101, 45%, 56%);
  --color-secondary-rotate: hsl(120, 45%, 56%);
  --color-tertiary:         hsl(49, 89%, 64%);
  --color-glint:            hsl(210, 100%, 82%);

  // Neutral colors
  --color-white:      hsl(0, 0%, 100%);
  --color-background: hsl(30, 50%, 98%);
  --color-light:      hsl(0, 6%, 93%);
  --color-dark:       var(--color-text-header);
}
```

La última parte de nuestro archivo de variables contendrá varios estilos de interfaz de usuario, como radios de bordes y sombras de cuadros. Una vez más,

el objetivo es garantizar la coherencia en nuestra aplicación.

```
// app/assets/stylesheets/config/_variables.scss

:root {
  // All the previous variables

  // Border radius
  --border-radius: 0.375rem;

  // Border
  --border: solid 2px var(--color-light);

  // Shadows
  --shadow-large:  2px 4px 10px hsl(0 0% 0% / 0.1);
  --shadow-small:  1px 3px 6px hsl(0 0% 0% / 0.1);
}
```

¡Eso es todo! ¡Esas son todas las variables necesarias para diseñar nuestro editor de cotizaciones! (Utilizo las mismas variables en el sitio web que estás leyendo actualmente).

El siguiente paso en el diseño de nuestra aplicación es aplicar esas variables a estilos globales, es decir, estilos que sean todos iguales en toda la aplicación. Estos estilos deben representar valores predeterminados de estilo razonables para las etiquetas HTML. No debemos utilizar ninguna clase CSS aquí. Por ejemplo, en el archivo siguiente:

- Estilo de texto predeterminado
- Enlaces de estilo predeterminados
- Restablezca todos los márgenes y rellenos a cero como debería ser responsabilidad de las clases de diseño.

Estos cambios son globales para todas las páginas de nuestra aplicación y se pueden anular fácilmente porque solo se aplican a las etiquetas HTML y, por lo tanto, tienen poca especificidad.

```
// app/assets/stylesheets/config/_reset.scss

*,
*::before,
*::after {
```

```
  box-sizing: border-box;
}

* {
  margin: 0;
  padding: 0;
}

html {
  overflow-y: scroll;
  height: 100%;
}

body {
  display: flex;
  flex-direction: column;
  min-height: 100%;

  background-color: var(--color-background);
  color: var(--color-text-body);
  line-height: var(--line-height-body);
  font-family: var(--font-family-sans);
}

img,
picture,
svg {
  display: block;
  max-width: 100%;
}

input,
button,
textarea,
select {
  font: inherit;
}

h1,
h2,
h3,
h4,
h5,
h6 {
  color: var(--color-text-header);
```

```
  line-height: var(--line-height-headers);
}

h1 {
  font-size: var(--font-size-xxxl);
}

h2 {
  font-size: var(--font-size-xxl);
}

h3 {
  font-size: var(--font-size-xl);
}

h4 {
  font-size: var(--font-size-l);
}

a {
  color: var(--color-primary);
  text-decoration: none;
  transition: color 200ms;

  &:hover,
  &:focus,
  &:active {
    color: var(--color-primary-rotate);
  }
}
```

Ahora que tenemos nuestro estilo global establecido, podemos diseñar nuestros componentes individuales.

La carpeta de componentes

La carpeta de componentes contendrá estilos para nuestros componentes individuales. Ahora que tenemos un conjunto sólido de variables, diseñar componentes será sencillo.

Comencemos, *aunque no lo creas* , por nuestros componentes más complejos: los botones. Comenzaremos con la `.btn` clase base y luego agregaremos cuatro *modificadores* para los diferentes estilos:

```
// app/assets/stylesheets/components/_btn.scss

.btn {
  display: inline-block;
  padding: var(--space-xxs) var(--space-m);
  border-radius: var(--border-radius);
  background-origin: border-box; // Invisible borders with linear gradie
  background-color: transparent;
  border: solid 2px transparent;
  font-weight: bold;
  text-decoration: none;
  cursor: pointer;
  outline: none;
  transition: filter 400ms, color 200ms;

  &:hover,
  &:focus,
  &:focus-within,
  &:active {
    transition: filter 250ms, color 200ms;
  }

  // Modifiers will go there
}
```

La `.btn` clase es un elemento de bloque en línea al que le agregamos estilos predeterminados como padding, border-radius y transition. Tenga en cuenta que con Sass, el `&` signo corresponde al selector en el que `&` está anidado directamente. En nuestro caso `&:hover` , Sass traducirá `.btn:hover` .

Luego necesitamos agregar cuatro modificadores para crear las cuatro variantes del componente de botón que necesitaremos en nuestra aplicación:

```
// app/assets/stylesheets/components/_btn.scss

.btn {
  // All the previous code

  &--primary {
    color: var(--color-white);
    background-image: linear-gradient(to right, var(--color-primary), va

    &:hover,
```

```
    &:focus,
    &:focus-within,
    &:active {
      color: var(--color-white);
      filter: saturate(1.4) brightness(115%);
    }
  }

  &--secondary {
    color: var(--color-white);
    background-image: linear-gradient(to right, var(--color-secondary),

    &:hover,
    &:focus,
    &:focus-within,
    &:active {
      color: var(--color-white);
      filter: saturate(1.2) brightness(110%);
    }
  }

  &--light {
    color: var(--color-dark);
    background-color: var(--color-light);

    &:hover,
    &:focus,
    &:focus-within,
    &:active {
      color: var(--color-dark);
      filter: brightness(92%);
    }
  }

  &--dark {
    color: var(--color-white);
    border-color: var(--color-dark);
    background-color: var(--color-dark);

    &:hover,
    &:focus,
    &:focus-within,
    &:active {
      color: var(--color-white);
    }
```

```
  }
}
```

Aquí se aplica la misma regla: `&--primary` será traducido `.btn--primary` por el preprocesador Sass.

¡Eso fue mucho CSS para un botón simple! Los siguientes componentes serán mucho más simples. Continuemos con el `.quote` componente:

```
// app/assets/stylesheets/components/_quote.scss

.quote {
  display: flex;
  justify-content: space-between;
  align-items: center;
  gap: var(--space-s);

  background-color: var(--color-white);
  border-radius: var(--border-radius);
  box-shadow: var(--shadow-small);
  margin-bottom: var(--space-m);
  padding: var(--space-xs);

  @include media(tabletAndUp) {
    padding: var(--space-xs) var(--space-m);
  }

  &__actions {
    display: flex;
    flex: 0 0 auto;
    align-self: flex-start;
    gap: var(--space-xs);
  }
}
```

El `.quote` componente es simplemente un contenedor flexible con un `.quote__actions` elemento que contiene los botones para editar y eliminar la cita. ¡Gracias a la metodología BEM, nuestros archivos CSS están muy limpios!

Ahora diseñemos nuestros formularios en línea. En el capítulo anterior, definimos los contenedores de formularios simples que se utilizarán para los

formularios de nuestra aplicación. Necesitaremos dos componentes para diseñar nuestros formularios: `.form` y `.visually-hidden`.

```
config.wrappers :default, class: "form__group" do |b|
  b.use :html5
  b.use :placeholder
  b.use :label, class: "visually-hidden"
  b.use :input, class: "form__input", error_class: "form__input--invalid
end
```

Comencemos con el `.form` componente. Observe que tiene el mismo ancho de borde y relleno vertical que el `.btn` componente para que tenga la misma altura:

```
// app/assets/stylesheets/components/_form.scss

.form {
  display: flex;
  flex-wrap: wrap;
  gap: var(--space-xs);

  &__group {
    flex: 1;
  }

  &__input {
    display: block;
    width: 100%;
    max-width: 100%;
    padding: var(--space-xxs) var(--space-xs);
    border: var(--border);
    border-radius: var(--border-radius);
    outline: none;
    transition: box-shadow 250ms;

    &:focus {
      box-shadow: 0 0 0 2px var(--color-glint);
    }

    &--invalid {
      border-color: var(--color-primary);
    }
```

```
  }
}
```

Por último, pero no menos importante, agreguemos el `.visually-hidden` componente:

```
// app/assets/stylesheets/components/_visually_hidden.scss

// Shamelessly stolen from Bootstrap

.visually-hidden {
  position: absolute !important;
  width: 1px !important;
  height: 1px !important;
  padding: 0 !important;
  margin: -1px !important;
  overflow: hidden !important;
  clip: rect(0, 0, 0, 0) !important;
  white-space: nowrap !important;
  border: 0 !important;
}
```

Quizás te preguntes por qué necesitamos un `.visually-hidden` componente para ocultar la etiqueta de entrada y no simplemente eliminarla del DOM o usarla `display: none;`. Por motivos de accesibilidad, todas las entradas de formulario deben tener etiquetas que puedan ser interpretadas por los lectores de pantalla incluso si no están visibles en la página web. Es por eso que la mayoría de las aplicaciones usan un `.visually-hidden` componente. No tenemos que aprender el CSS de memoria, ya que podemos robar el componente del código fuente de Bootstrap .

Ningún formulario estaría completo sin una forma de mostrar mensajes de error. Agreguemos el componente simple para mostrar errores en rojo:

```
// app/assets/stylesheets/components/_error_message.scss

.error-message {
  width: 100%;
  color: var(--color-primary);
  background-color: var(--color-primary-bg);
  padding: var(--space-xs);
```

```
  border-radius: var(--border-radius);
}
```

Eso es todo. Tenemos todos los componentes que necesitamos para completar el CRUD en nuestro `Quote` modelo. Ahora solo necesitamos dos diseños y habremos terminado con el CSS.

La carpeta de diseños

El contenedor es un diseño que casi todas las aplicaciones web tienen. Se utiliza para centrar el contenido en la página y limitar su ancho máximo. Lo utilizaremos en todas las páginas de nuestro editor de citas. Esta es la implementación que utilizaremos:

```
// app/assets/stylesheets/layouts/_container.scss

.container {
  width: 100%;
  padding-right: var(--space-xs);
  padding-left: var(--space-xs);
  margin-left: auto;
  margin-right: auto;

  @include media(tabletAndUp) {
    padding-right: var(--space-m);
    padding-left: var(--space-m);
    max-width: 60rem;
  }
}
```

El encabezado es un diseño que utilizaremos dos veces en este tutorial, en la página `Quotes#index` y en la `Quotes#show` página. Contendrá el título de la página y el botón de acción principal:

```
// app/assets/stylesheets/layouts/_header.scss

.header {
  display: flex;
  flex-wrap: wrap;
  gap: var(--space-s);
  justify-content: space-between;
  margin-top: var(--space-m);
```

```
  margin-bottom: var(--space-l);

  @include media(tabletAndUp) {
    margin-bottom: var(--space-xl);
  }
}
```

Ahora que tenemos todos nuestros diseños, escribimos todos los CSS

💎 Trenes calientes

archivos en el archivo de manifiesto para que Sass los agrupe.

El archivo de manifiesto

Por último, tenemos que importar todos esos archivos CSS `application.sass.scss` para que todos nuestros archivos de diseño se agreguen a un solo archivo CSS compilado.

```
// app/assets/stylesheets/application.sass.scss

// Mixins
@import "mixins/media";

// Configuration
@import "config/variables";
@import "config/reset";

// Components
@import "components/btn";
@import "components/error_message";
@import "components/form";
@import "components/visually_hidden";
@import "components/quote";

// Layouts
@import "layouts/container";
@import "layouts/header";
```

Aquí estamos; con solo unos pocos archivos CSS, creamos un bonito diseño para nuestro CRUD en el `Quote` modelo. Agregaremos nuevos componentes en los siguientes capítulos a medida que los necesitemos, pero ya escribimos la mayor parte de nuestro CSS.

Ahora es el momento de empezar a trabajar en lo que más nos interesa: **Turbo** ! Tómate un pequeño descanso y nos vemos en el próximo capítulo, donde hablaremos sobre **Turbo Drive** !

← anterior Siguiente →