Informe Técnico: Metodologías de Organización de Estilos en SmartBudget

Para: Equipo Técnico
De: [Tu Nombre]
Fecha: 6 de Febrero, 2026

---

1. ¿Qué problemas resuelve este proyecto?

Al organizar un sitio web sin una metodología clara, aparecen varios obstáculos:

1. **Conflictos de estilos:** Cuando las clases CSS no están bien organizadas, es fácil que un estilo sobrescriba a otro accidentalmente. Esto hace que tengamos que usar selectores muy específicos solo para "ganar" la batalla.
2. **Código difícil de mantener:** Si todos los estilos están en un solo archivo gigante, encontrar dónde hacer un cambio se vuelve complicado y lento.
3. **Reutilización limitada:** Sin componentes bien definidos, terminas escribiendo el mismo código una y otra vez para botones, tarjetas o menús.

Con la organización correcta, el código se vuelve predecible y escalable.

---

2. ¿Qué solución elegí? (Metodología BEM)

Para este proyecto utilic é **BEM** (Block Element Modifier), una metodología que organiza el CSS en bloques independientes.

¿Por qué BEM?

- **Independencia:** Cada componente es autónomo. Si creo un botón con la clase `.btn--primary`, puedo usarlo en cualquier parte del sitio y siempre se verá igual.
- **Nombres descriptivos:** En lugar de clases genéricas como `.rojo` o `.grande`, usamos nombres que dicen exactamente qué son. Por ejemplo, `.navbar__link--active` deja claro que es un link activo dentro del navbar.
- **Escalabilidad:** Si el proyecto crece, BEM permite que varios desarrolladores trabajen sin pisar el código del otro, porque cada uno trabaja en su propio bloque.

---

3. ¿Cómo lo organicé? (SASS con patrón 7-1)

Para aprovechar al máximo BEM, utilicé **SASS**, un preprocesador CSS que permite escribir código más organizado con variables y mixins.

Apliqué el **patrón 7-1**, que divide el proyecto en carpetas lógicas:

abstracts/: Variables y mixins reutilizables. Aquí definí la paleta de colores, tipografía y tamaños. Si quisiera cambiar el color principal del sitio, solo lo modifico en `_variables.scss` y se actualiza automáticamente en todo el código.

base/: Estilos base y reset. Aquí puse las reglas generales para que todo se vea consistente en cualquier navegador.

components/: Componentes reutilizables como botones y cards. Cada componente vive en su propio archivo (por ejemplo, `_buttons.scss`), lo que hace muy fácil encontrar y modificar estilos específicos.

layout/: Estructura principal como header y footer.

pages/: Estilos específicos de cada página (en este caso, la landing page).

---

4. Integración con Bootstrap 4

Además de mi código personalizado con BEM, integré **Bootstrap 4** para aprovechar componentes pre-diseñados como:

- Navbar responsive con menú hamburguesa
- Grid system (container, row, col)
- Modales funcionales
- Utilidades de espaciado

Bootstrap me permitió acelerar el desarrollo sin sacrificar la personalización, ya que aplicé mis propios estilos BEM encima de los componentes de Bootstrap para mantener la identidad visual del proyecto.

---

5. Reflexión final

Aplicar BEM junto con el patrón 7-1 de SASS hizo que el proyecto fuera mucho más organizado que escribir todo en un solo archivo CSS.

Lo más valioso fue la **modularidad**: cuando terminé el componente de botones, sabía que podía usarlo en cualquier parte sin romper otros estilos. En equipos de trabajo, esto es fundamental porque varios programadores pueden trabajar en componentes diferentes al mismo tiempo sin generar conflictos.

La curva de aprendizaje existe (acostumbrarse a la nomenclatura de BEM toma tiempo), pero los beneficios en mantenibilidad y escalabilidad son enormes.