



MÓDULO DIW · DAW — UNIDAD DIDÁCTICA 5

Arquitectura CSS (BEM) y Utility-First con Tailwind CSS

RA2 · Crea interfaces web homogéneas definiendo y aplicando estilos

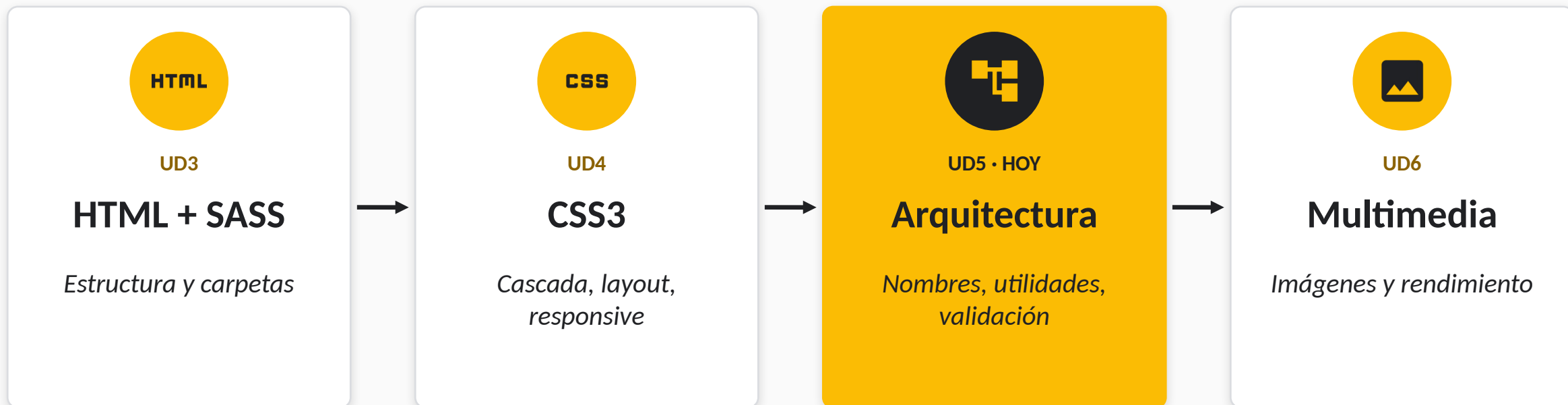
- Crear clases de estilo con una convención que entienda todo el equipo (BEM)
- Componer interfaces con clases de utilidad (Tailwind CSS)
- Validar el CSS con herramientas y mantener viva la guía de estilo





DÓNDE ESTAMOS

AURIA ya tiene piel: hoy ordenamos el armario



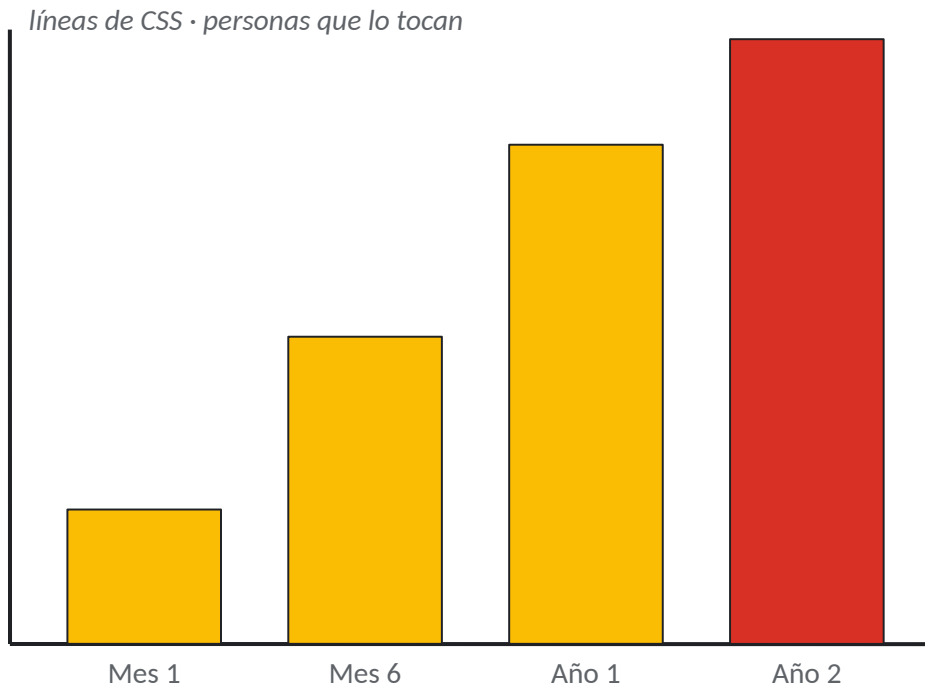
La pregunta de la unidad: ¿cómo escribimos CSS que otra persona pueda tocar sin miedo dentro de seis meses?

Con esta unidad se cierra el RA2. En la UD6 empieza el RA3: recursos multimedia.



El CSS se escribe una vez y se lee cien

Un proyecto real con el tiempo



Varias personas, un mismo CSS

Lo que tú escribes hoy lo modifica otra persona mañana



Se lee más de lo que se escribe

Un nombre claro ahorra horas de buscar «qué hace esta clase»



Miedo a borrar

Sin convención nadie sabe si una regla se usa: el CSS solo crece



Revisión automática

En las empresas el CSS pasa un linter antes de entrar al proyecto



Diseño y código sincronizados

La guía de estilo es el contrato entre diseño y desarrollo



DATO CLAVE — El problema del CSS no es escribirlo: es mantenerlo cuando crece y cuando lo toca más de una persona.



Cuatro bloques, de la convención a la guía de estilo



1 • El problema

Nombres sin sentido,
especificidad que sube, miedo a
borrar



2 • BEM

Bloque, elemento y modificador:
nombres que explican la
estructura



3 • Tailwind CSS

Utility-first: componer con clases
pequeñas y tokens en @theme



4 • Validar y documentar

W3C, Stylelint, Prettier y la guía de
estilo (RA2.h, RA2.i)



Laboratorio: la tarjeta de producto de AURIA, refactorizada en BEM con SASS, la misma tarjeta en Tailwind, Stylelint vigilando los nombres y la guía de estilo al día.



CLAVE — BEM y Tailwind no son rivales que haya que elegir de por vida: son dos respuestas al mismo problema.



BLOQUE 1

El problema: CSS que nadie toca

Por qué un proyecto sin convención acaba siendo imposible de mantener



Tres síntomas de un CSS sin convención

styles.css (año 2)

```
.card { ... }  
.card2 { ... }  
.btn-2 { ... }  
.rojo { color: red; }  
.nuevo-btn-ok { ... }  
  
#main .card .btn:hover { ... }  
div.productos ul li a span { ... }  
  
.card .title { ... }  
.modal .title { ... } /* ¿cuál gana? */
```



Nombres sin significado

.card2, .btn-2, .rojo: ¿qué son? ¿dónde se usan? ¿y si .rojo pasa a ser azul?



Especificidad que sube

Selectores largos y anidados que solo se pueden vencer con otro más largo... o con !important (UD4)



Estilos que se pisan

.card .title y .modal .title dependen del sitio: mueves el HTML y cambia el aspecto



RESULTADO — Nadie se atreve a borrar nada: se añade otra regla «por si acaso» y el fichero solo crece.



Recordad la UD4: la especificidad es una escalada

`.title`



$(0, 1, 0)$

`.card .title`



$(0, 2, 0)$

`#main .card .title`



$(1, 2, 0)$

`#main .card .title !important`



!important

La idea de toda la unidad: si TODOS los selectores pesan lo mismo, $(0, 1, 0)$, nadie tiene que pelear: decide el orden y el nombre.



CLAVE — La especificidad baja y plana no es una manía: es lo que permite cambiar una pieza sin romper otra.



Dos respuestas al mismo problema

CSS sin convención

BEM

Semántica y estructura

Nombras cada pieza según lo que ES dentro de su componente y escribes su CSS una vez.

`.card__title`

Utility-first (Tailwind)

Composición atómica

Cada clase hace UNA cosa. Montas el aspecto combinándolas en el HTML.

`text-xl font-semibold`



OTRAS — Existen más arquitecturas (OOCSS, SMACSS, ITCSS...). BEM y Tailwind son las más extendidas y las que veremos.



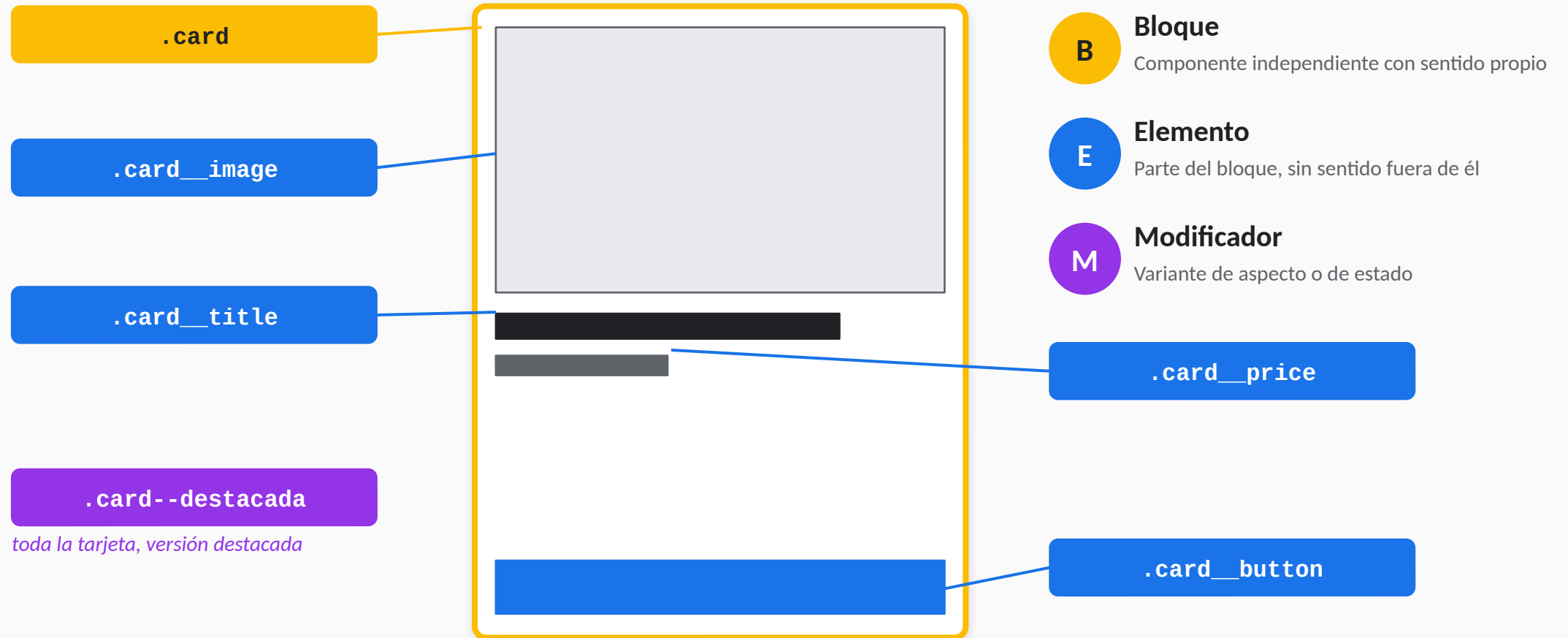
BLOQUE 2

BEM: Bloque, Elemento, Modificador

Una convención de nombres que explica la estructura del componente



Tres piezas: bloque, elemento y modificador



ORIGEN — BEM lo creó Yandex (el buscador ruso) hacia 2009 para que equipos grandes compartieran el mismo CSS.



La sintaxis: dos guiones bajos y dos guiones



Se lee: «el botón de la tarjeta, en su variante principal»

Patrón	Ejemplo	Significado
.bloque	.card	El componente
.bloque__elemento	.card__title	Una parte del componente
.bloque--modificador	.card--destacada	Variante del componente entero
.bloque__elemento--modificador	.card__button--primary	Variante de una parte



NOMBRES — En minúsculas y con guion simple dentro de cada palabra compuesta: .product-card__add-button--disabled



La tarjeta de AURIA, con nombres BEM

index.html

```
<article class="card card--destacada">
  
  <h3 class="card__title">AURIA Pro</h3>
  <p class="card__price">199 €</p>
  <button class="card__button
    card__button--primary">
    Añadir al carrito
  </button>
</article>
```

1

Modificador + base

class="card card--destacada": el modificador SIEMPRE acompaña a la clase base, no la sustituye

2

Se lee en el HTML

Sin abrir el CSS ya sabes qué pieza es cada cosa y a qué componente pertenece

3

El HTML semántico sigue

BEM pone nombres a las clases; las etiquetas (article, h3, button) siguen siendo las de la UD3



OJO — BEM no cambia qué etiqueta usas: un botón sigue siendo <button>. Solo ordena los nombres de las clases.



BEM y SASS encajan: el & de la UD3

components/_card.scss

```
.card {  
  display: flex;  
  flex-direction: column;  
  gap: $space-md;  
  
  &--destacada {  
    border: 2px solid $color-brand-primary;  
  }  
  &__title { font-size: 1.25rem; }  
  &__button--primary {  
    background: $color-brand-primary;  
  }  
}
```

compila



css/main.css

```
.card { display: flex; ... }  
.card--destacada { border: ... }  
.card__title { font-size: 1.25rem; }  
.card__button--primary {  
  background: #1e88e5;  
}
```

Todo el CSS de la tarjeta en un fichero, y en el resultado todas las reglas son de UNA sola clase.



OJO — &__title genera .card__title (una clase), no .card .title (dos). Por eso BEM + SASS no aumenta la especificidad.



Especificidad plana: todas las reglas pesan (0,1,0)

Sin convención

`#main .card .title`



`.card .title`



`.modal div .title`



`.title`



Un id o un selector anidado pesan más que cualquier clase sola: para cambiarlos hay que escribir algo aún más pesado.

Con BEM

`.card__title`



`.card__button`



`.modal__title`



`.card--destacada`



Todas iguales: si dos reglas chocan, decide el orden. Cambiar una pieza no obliga a pelear con otra.



CLAVE — Esta es la alternativa al `!important` de la UD4: no ganar subiendo el peso, sino que nadie tenga que ganar.



Cuatro reglas que evitan el 90 % de los errores

Sin nietos en el nombre

❌ `.card__body__title`

✅ `.card__title`

El nombre refleja el bloque, no el árbol del HTML

El modificador no va solo

❌ `class="card--destacada"`

✅ `class="card card--destacada"`

El modificador solo trae lo que cambia

Sin selectores de etiqueta

❌ `.card h3 { ... }`

✅ `.card__title { ... }`

Si cambias h3 por h2, el estilo no se pierde

Bloque dentro de bloque

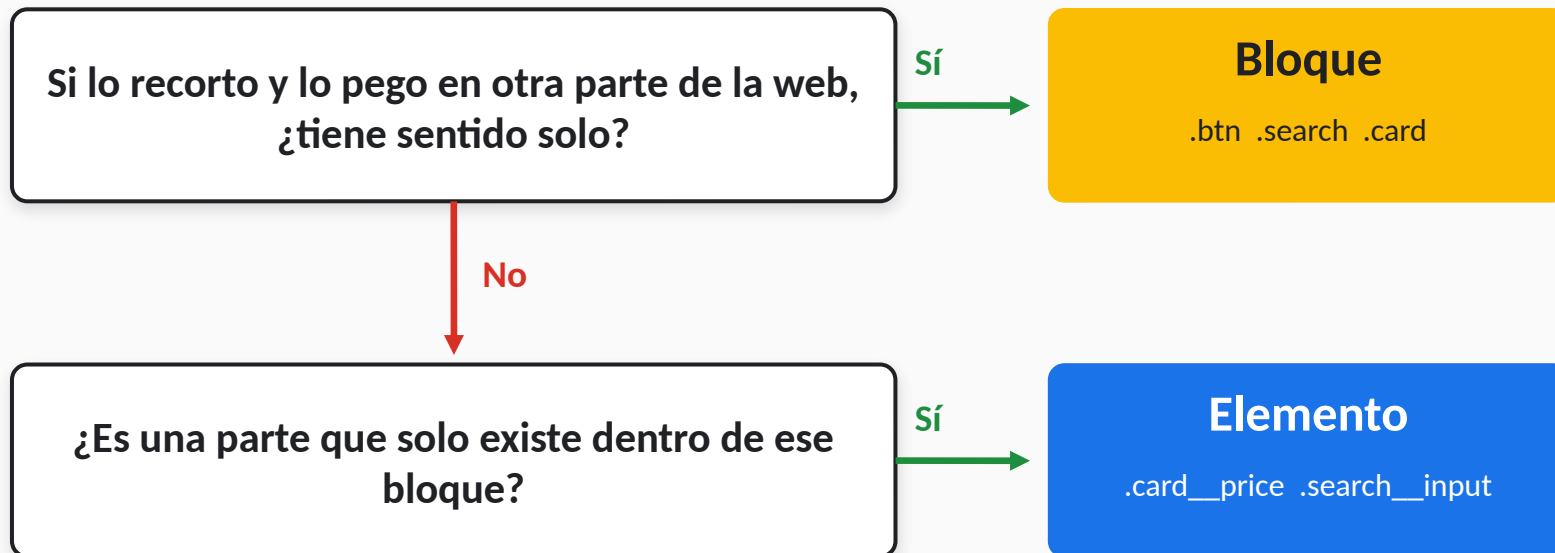
❌ `.card .btn { ... }`

✅ `class="card__button btn btn--primary"`

Mezcla: el botón es pieza de la tarjeta Y un bloque propio



¿Bloque o elemento? Una pregunta lo decide



Ejemplos de AURIA

Botón «Añadir»: bloque `.btn` (se usa en toda la web)

Precio: elemento `.card__price`

Valoración con estrellas: bloque `.rating` (aparece también en reseñas)

Etiqueta «Oferta»: elemento `.card__badge`, o bloque `.badge` si se reutiliza



CLAVE — No hay una respuesta única: lo importante es que el equipo lo decida y lo escriba en la guía de estilo.



Estados: modificador, pseudoclase o atributo ARIA

Pseudoclases

Estados que el navegador ya conoce

`&:hover`

`&:focus-visible`

`&:disabled`

El ratón encima, el foco del teclado, el botón deshabilitado

Modificadores

Variantes que decides tú

`.card--destacada`

`.btn--secondary`

`.alert--error`

Cambian el aspecto de forma permanente o según el contenido

Atributos ARIA

Estados que también debe oír el lector

`[aria-expanded="true"]`

`[aria-current="page"]`

`[aria-invalid="true"]`

El CSS y la accesibilidad leen el mismo dato (UD9)



REGLA — Si el estado ya existe en HTML o ARIA, úsalo para el CSS en vez de inventar una clase: así no se desincronizan.



Ventajas y límites de BEM



Ventajas

- Nombres que se explican solos en el HTML
- Especificidad plana y predecible
- Un componente = un fichero SASS
- Se puede borrar un bloque entero sin miedo
- Encaja con los componentes de la UD7 (Atomic Design)



Límites

- Nombres largos: `.card__button--primary`
- Exige disciplina: un solo `.card .title` y se rompe
- Hay que decidir qué es bloque y qué es elemento
- No guarda los valores: necesita los tokens de la UD2



CLAVE — Los nombres largos son el precio de la claridad: se escriben una vez y se leen cien.



BLOQUE 3

Utility-first con Tailwind CSS

Componer el aspecto con clases pequeñas que hacen una sola cosa



Cambio de paradigma: del nombre a la composición

BEM

1. Pones un nombre: `.card__button`
2. Escribes su CSS en `_card.scss`
3. Lo usas en el HTML

HTML limpio • la lógica visual está en el CSS

Utility-first

1. No escribes CSS nuevo
2. Combinas clases ya hechas:
`bg-blue-600 text-white px-4 py-2`
3. El aspecto se lee en el HTML

CSS mínimo • la lógica visual está en el HTML

Una clase de utilidad = UNA declaración: `flex` → `display: flex` • `p-4` → `padding: 1rem`



HISTORIA — Tailwind CSS nació en 2017 (Adam Wathan). La versión actual, la 4, se configura desde el propio CSS.



La misma tarjeta de AURIA, en Tailwind

```
index.html

<article class="flex flex-col gap-2 rounded-lg
              border-2 border-brand-primary p-4">
  
  <h3 class="text-xl font-semibold">AURIA Pro</h3>
  <p class="text-gray-600">199 €</p>
  <button class="rounded-md bg-brand-primary px-4 py-2
                text-white hover:bg-blue-700
                focus-visible:outline-2">
    Añadir al carrito
  </button>
</article>
```



Mismo resultado visual que la versión BEM



OBSERVA — No hay ni una línea de CSS propio: todo el aspecto está en el atributo class.



Las clases son CSS con nombre corto

CSS	Tailwind
display: flex	flex
flex-direction: column	flex-col
justify-content: space-between	justify-between
gap: 0.5rem (8px)	gap-2
padding: 1rem (16px)	p-4
padding-inline: 1rem	px-4
border-radius: 0.5rem	rounded-lg
font-weight: 600	font-semibold
display: grid; grid-template-columns: repeat(3, ...)	grid grid-cols-3

La escala de espaciado

Cada número son 0,25rem (4px):

p-1	 4px
p-2	 8px
p-4	 16px
p-6	 24px
p-8	 32px



TRUCO — No hay que memorizarlas: la extensión «Tailwind CSS IntelliSense» de VS Code autocompleta y enseña el CSS de cada clase.



Variantes: estados y responsive con un prefijo

Prefijo	Ejemplo	Equivale a
hover:	<code>hover:bg-blue-700</code>	<code>&:hover</code> de SASS
focus-visible:	<code>focus-visible:outline-2</code>	<code>&:focus-visible</code>
md:	<code>md:flex-row</code>	<code>@media (min-width: 48rem)</code>
lg:	<code>lg:grid-cols-3</code>	<code>@media (min-width: 64rem)</code>
dark:	<code>dark:bg-gray-900</code>	<code>@media (prefers-color-scheme: dark)</code>

Mobile-first, como en la UD4

```
class="flex-col md:flex-row"
```

- Sin prefijo: se aplica SIEMPRE (la base, el móvil)
- md: añade a partir de 768 px
- Se combinan: md:hover:bg-blue-700

sm 640 · md 768 · lg 1024 · xl 1280 · 2xl 1536



CLAVE — md: significa «a partir de md», no «solo en tablet». Es exactamente el min-width de la UD4.



Instalar Tailwind CSS v4 con su CLI

terminal

```
npm install tailwindcss @tailwindcss/cli
```

src/input.css

```
@import "tailwindcss";
```

terminal

```
npx @tailwindcss/cli -i ./src/input.css \
-o ./src/output.css --watch
```

1

Instalar

Como Dart Sass en la UD3: con npm, dentro del proyecto

2

Importar

Una sola línea en tu CSS de entrada trae todo Tailwind

3

Compilar con --watch

Lee tu HTML, busca las clases que usas y genera output.css

4

Enlazar

`<link rel="stylesheet" href="output.css">` en el HTML



PROTOTIPOS — Para probar sin instalar existe un `<script>` CDN (`@tailwindcss/browser`), pero no se usa en producción.



Los tokens de la UD2 entran por @theme

```
src/input.css

@import "tailwindcss";

@theme {
  --color-brand-primary: #1e88e5;
  --color-brand-dark: #1565c0;
  --spacing-md: 1rem;
  --font-display: "Inter", sans-serif;
}
```



Variable en @theme	Clases que crea
--color-brand-primary	bg-brand-primary · text-brand-primary · border-brand-primary
--spacing-md	p-md · m-md · gap-md
--font-display	font-display

Tailwind no sustituye a los tokens: los implementa. El nombre semántico de la UD2 (color-brand-primary) pasa a ser una clase. Si mañana cambia el azul de AURIA, se cambia en un solo sitio. Y además queda disponible como variable CSS: `var(--color-brand-primary)`.



VERSIONES — En Tailwind 3 esto se hacía en `tailwind.config.js`. Lo veréis en muchos tutoriales: la idea es la misma.



Solo genera las clases que usas



Tailwind tiene miles de clases posibles, pero tu CSS final solo trae las que encuentra en tu código: suele quedar en pocos KB.

La trampa

Tailwind busca texto, no ejecuta tu código. Una clase montada a trozos no existe para él:

```
`bg-${color}-600`  x
```

Escribe siempre el nombre completo:

```
color === 'rojo' ? 'bg-red-600'
```

```
: 'bg-blue-600'  ✓
```



CLAVE — Si una clase «no hace nada», lo primero: ¿aparece escrita entera en algún fichero que Tailwind lee?



Salidas de emergencia: valores arbitrarios y @apply

Valores arbitrarios

```
w-[327px] top-[13px] bg-[#ff5a1f]
```

Escriben un valor exacto entre corchetes. Útiles para un caso puntual... y peligrosos: son los valores mágicos de la UD2 otra vez.

Si lo repites, crea un token en @theme

@apply

```
.btn {  
  @apply rounded-md bg-brand-primary  
  px-4 py-2 text-white;  
}
```

Junta varias utilidades en una clase propia. Sirve cuando no tienes componentes (HTML repetido a mano).

Con moderación: si todo es @apply, vuelves a escribir CSS



CLAVE — Si en tu HTML hay muchos corchetes, no estás usando el sistema de diseño: lo estás saltando.



¿BEM o Tailwind? Depende del proyecto

	BEM + SASS	Tailwind CSS
Dónde está el aspecto	En el .scss	En el HTML
HTML	Limpio, nombres con significado	Largo, muchas clases
Velocidad para maquetar	Media	Alta
Tamaño del CSS	Crece con cada componente	Pequeño y estable
Tokens	Variables SASS / custom properties	@theme
Encaja mejor con	Webs con HTML escrito a mano, CMS	Proyectos con componentes (React, Vue...)

Y se pueden mezclar

Muchos equipos usan BEM (o componentes) para la estructura y unas pocas utilidades para ajustes puntuales: márgenes entre piezas, ocultar algo en móvil...

Lo que NO funciona es no decidir: medio proyecto con cada sistema y sin norma.



CLAVE — No hay un ganador: hay una decisión de equipo, escrita en la guía de estilo y vigilada por las herramientas.



BLOQUE 4

Validar y documentar

Herramientas que revisan el CSS por ti y una guía de estilo viva (RA2.h, RA2.i)



Tres herramientas, tres preguntas distintas



Validador W3C

¿Es CSS válido según el estándar?

Propiedades que no existen, valores mal escritos, llaves sin cerrar

jigsaw.w3.org/css-validator • manual



Stylelint

¿Cumple las normas del equipo?

Nombres BEM, colores repetidos, selectores demasiado específicos, duplicados

En el editor y antes de cada commit



Prettier

¿Está bien formateado?

Sangría, espacios, comillas, una declaración por línea

Automático al guardar



OJO — El validador W3C entiende CSS, no SCSS: se le pasa el main.css compilado. Stylelint sí puede revisar el .scss.



Stylelint: instalar y enseñarle BEM

terminal

```
npm install --save-dev stylelint \
stylelint-config-standard-scss
```

.stylelintrc.json

```
{
  "extends": "stylelint-config-standard-scss",
  "rules": {
    "selector-class-pattern": [
      "^[a-z][a-z0-9-]*(__[a-z0-9-]+)?(--[a-z0-9-]+)?$",
      { "resolveNestedSelectors": true,
        "message": "Usa BEM: bloque__elemento--mod" }
    ]
  }
}
```

extends

Parte de un conjunto de reglas ya hecho (el estándar, versión SCSS)

selector-class-pattern

Expresión regular que deben cumplir todos los nombres de clase

La regex, en castellano

minúsculas, y opcionalmente __elemento y --modificador.
resolveNestedSelectors: revisa también los &__ de SASS

terminal

```
npx stylelint "scss/**/*.scss"
```



CLAVE — Ya no depende de que alguien se acuerde de la convención: la máquina la revisa en cada fichero.



Leer lo que dice Stylelint

```
npx stylelint "scss/**/*.scss"
```

```
scss/components/_card.scss
```

```
12:3 ✖ Expected ".cardTitle" to match pattern
```

```
selector-class-pattern
```

```
18:5 ✖ Unexpected duplicate selector ".card__price"
```

```
no-duplicate-selectors
```

```
24:14 ✖ Expected "#ffffff" to be "#fff"
```

```
color-hex-length
```

```
✖ 3 problems (3 errors, 0 warnings)
```

12:3

Línea y columna: dónde está el problema

✖ **mensaje**

Qué ha encontrado, en inglés

selector-class-pattern

El nombre de la regla: búscalo en stylelint.io para saber por qué existe

```
npx stylelint "scss/**/*.scss" --fix
```

arregla lo automático (por ejemplo, #ffffff → #fff); los nombres los cambias tú



TRUCO — Con la extensión de Stylelint en VS Code los errores aparecen subrayados mientras escribes, como las faltas de ortografía.



El circuito: que la máquina revise antes que las personas



Por qué importa: en una revisión de código nadie debería perder tiempo discutiendo sangrías o nombres mal escritos. Las herramientas se encargan de lo mecánico y las personas revisan lo que importa: si el componente está bien pensado.



EN CLASE — Nos quedamos en el editor (Prettier + Stylelint). Commit y CI con git los veréis en los proyectos.



La guía de estilo: de Penpot al código

Elemento	En el diseño (UD2)	En el código (UD4-UD5)
Colores	Tokens de color en Penpot	<code>\$color-brand-primary</code> • <code>--color-brand-primary</code> en <code>@theme</code>
Tipografía	Escala de estilos de texto	Variables de tamaño • <code>clamp()</code> (UD4) • <code>text-xl</code>
Espaciado	Tokens de espaciado	<code>\$space-md</code> • <code>--spacing-md</code>
Componentes	Componentes de la librería	Bloques BEM (<code>_card.scss</code>) • catálogo en Storybook (UD7)
Convenciones	—	BEM, Stylelint y cuándo usar utilidades

RA2.i pide UTILIZAR y ACTUALIZAR la guía: si el código cambia y la guía no, la guía miente.



CLAVE — Un token cambiado en Penpot que no llega al código (o al revés) es la divergencia que hay que evitar.



Qué contiene una guía de estilo de código



Tokens

Colores, tipografía, espaciado y radios, con su nombre en código



Componentes

Cada bloque: para qué sirve, sus elementos y sus modificadores



Convención de nombres

BEM, prefijos, idioma (inglés o castellano, pero uno)



Hacer / no hacer

Ejemplos correctos e incorrectos, como en Material o GOV.UK



Herramientas

Cómo compilar, qué linter se usa y con qué reglas



Cambios

Qué se ha modificado y cuándo, para que nadie se pierda



EN AURIA — Basta un README.md en el proyecto con estas secciones. En la UD7 se convierte en un catálogo navegable con Storybook.



BLOQUE 5

Manos a la obra

Laboratorio AURIA, caso real y errores comunes



La tarjeta de AURIA, con dos arquitecturas



Se parte de la tarjeta de la UD4. La versión Tailwind va en una carpeta aparte (auria-tailwind/) para comparar las dos sin mezclarlas.



ÉXITO — Las dos tarjetas se ven igual, Stylelint no da errores y cambiar el azul de marca es tocar una sola línea en cada versión.



CASO REAL

Arquitecturas CSS en producción



GOV.UK Frontend

El sistema de diseño del gobierno británico usa BEM con prefijo: .govuk-button--secondary



Tailwind en componentes

Librerías como shadcn/ui o Tailwind Plus reparten componentes hechos solo con utilidades



Linter obligatorio

Los proyectos serios rechazan un cambio si no pasa Stylelint en la integración continua



Tokens como fuente única

Los mismos tokens alimentan Penpot/Figma, el CSS y las apps móviles



Guías públicas

Material Design, GOV.UK o Atlassian publican sus guías con ejemplos de hacer / no hacer



Borrar sin miedo

Con componentes aislados se puede eliminar código muerto sabiendo que no rompe nada



PRUEBA — Abre gov.uk, F12 y mira las clases de cualquier botón: es BEM de libro.



Trampas de BEM

Nietos en el nombre

Copiar el árbol del HTML en la clase

✗ `.card__body__footer__button`

✓ `.card__button`

Modificador sin base

Se pierden los estilos comunes

✗ `<div class="card--destacada">`

✓ `<div class="card card--destacada">`

Volver a anidar

La especificidad sube otra vez

✗ `.card .card__title { ... }`

✓ `.card__title { ... }`

Estilar por etiqueta

Cambias h3 por h2 y se rompe

✗ `.card h3 { ... }`

✓ `.card__title { ... }`



Trampas de Tailwind y herramientas

Clases montadas a trozos

Tailwind no las encuentra y no las genera

✗ ``text-${tono}-600``

✓ `tono === 'ok' ? 'text-green-600' : ...`

Corchetes por todas partes

Valores mágicos otra vez

✗ `p-[13px] text-[#1e88e5]`

✓ `p-md text-brand-primary (@theme)`

Validar el .scss en el W3C

El validador no entiende SASS

✗ Pegar `_card.scss` en el validador

✓ Validar `css/main.css` (el compilado)

Desactivar la regla que molesta

El linter deja de proteger

✗ `"selector-class-pattern": null`

✓ Renombrar la clase según BEM



Preguntas de control y debate

- 1 ¿Qué es bloque, qué es elemento y qué es modificador en `.card__button--primary`?
- 2 ¿Por qué con BEM no hace falta `!important`? Explícalo con la especificidad de la UD4.
- 3 ¿Qué diferencia hay entre escribir `.card__title` con `&` en SASS y escribir `.card .title`?
- 4 En Tailwind, ¿qué significa `md:flex-row` y a qué media query de la UD4 equivale?
- 5 ¿Qué revisa el validador W3C, qué revisa Stylelint y qué hace Prettier?



MINI-RETO PRÁCTICO

Reto express: 20 minutos, en parejas

1

Detectives

Encontrad 3 webs: una BEM, una Tailwind y una sin convención

2

BEM

Renombrad en BEM un componente de la que no tiene

3

Tailwind

Rehaced ese componente solo con utilidades

4

Stylelint

Pasad vuestro BEM por Stylelint: 0 errores

5

Veredicto

¿Cuál elegiríais para ese proyecto y por qué?

Pistas para reconocerlas

F12 → Elements → mirad los atributos class



Pareja A



Pareja B

- Dobles guiones bajos y dobles guiones → BEM
- flex px-4 md:... → Tailwind
- Clases tipo .box2 o ids con estilo → sin convención



ÉXITO — La otra pareja entiende vuestro componente BEM sin abrir el CSS y vuestra elección está justificada.



Arquitectura CSS y BEM

arquitectura CSS

Normas para organizar y nombrar el CSS de un proyecto

BEM

Block, Element, Modifier: convención de nombres de clases

bloque

Componente independiente con sentido propio: `.card`

elemento

Parte de un bloque, sin sentido fuera de él: `.card__title`

modificador

Variante de un bloque o elemento: `.card--destacada`

mezcla (mix)

Un nodo con clases de dos bloques: `card__button btn`

especificidad plana

Todas las reglas con una sola clase: (0,1,0)

&

En SASS, el selector actual: `&__title` → `.card__title`

OOCSS • SMACSS • ITCSS

Otras arquitecturas CSS conocidas

código muerto

CSS que ya no se usa y nadie se atreve a borrar

guía de estilo

Documento vivo con tokens, componentes y normas

convención

Acuerdo de equipo sobre cómo se escribe el código



Tailwind, validación y formato

utility-first

Componer el aspecto con clases de una sola función

clase de utilidad

Clase que aplica una sola declaración: p-4

Tailwind CSS

Framework utility-first; v4 se configura desde el CSS

variante

Prefijo que condiciona una utilidad: hover: md: dark:

@theme

Bloque de Tailwind v4 donde se definen los tokens

valor arbitrario

Valor exacto entre corchetes: w-[327px]

@apply

Junta utilidades dentro de una clase propia

linter

Herramienta que revisa reglas y errores del código

Stylelint

Linter de CSS y SCSS configurable por reglas

Prettier

Formateador automático de código

Validador W3C

Comprueba que el CSS cumple el estándar

CI

Integración continua: revisión automática en el servidor



Dentro de seis meses, ¿quién va a leer tu CSS?

1

Nombres

BEM o utilidades: que cada clase diga lo que es

2

Tokens

Los valores salen de la guía de estilo, no de la cabeza

3

Herramientas

Stylelint y Prettier revisan lo mecánico

Probablemente tú.

Y no te vas a acordar de por qué pusiste `.btn-2`. Escribe el CSS para esa persona.

Igual que con los tokens: el ahorro no se ve el día que lo escribes, se ve el día que hay que cambiarlo.



DEBATE — ¿Qué haría falta para que un compañero tocara vuestro CSS de AURIA sin preguntaros nada?

CHECKPOINT

Resumen de puntos clave



Sin convención, el CSS crece, la especificidad sube y nadie se atreve a borrar



BEM: .bloque__elemento--modificador, todo con una sola clase (0,1,0)



En SASS, &__ y &-- generan nombres BEM sin anidar selectores



Tailwind compone con utilidades; los tokens entran por @theme



W3C valida el estándar, Stylelint las normas del equipo, Prettier el formato



La guía de estilo se usa y se actualiza: si el código cambia y ella no, miente

SIGUIENTE • UD6

Multimedia y rendimiento

- Licencias y derechos de autor
- WebP, AVIF y SVG
- picture, srcset y sizes
- Audio, vídeo y carga diferida

Con la UD5 se cierra el RA2. En la UD6 empezamos el RA3: imágenes, vídeo y audio que pesen poco y carguen rápido.