

Tokens de @pi/ui

Todo el aspecto visual de la librería (color, forma, sombra, tipografía de controles) se resuelve en **custom properties de CSS**, nunca en valores hardcoded dentro de un componente. Hay dos familias con propósitos distintos, más un puñado de primitivas base:

Familia	Dónde vive	Qué representa
--pi-color-*	:root (_pi-color-variables.scss)	Paleta de marca (primary/accent/highlighted/warn/success/grey/white/black/background)
--pi-theme-*	.pi-shell (themes/*.scss)	Forma y comportamiento por superficie: botón, campo, card, panel, chip, overlay
Primitivas base	:root (_pi-global-variables.scss, _pi-font-sizes-variables.scss, _pi-public-variables.scss)	Espaciado, radios, sombras, tamaños de fuente, animaciones — sin tematizar

Los componentes de `common/` (botón, campo, card, panel, chip, modal, snackbar, tooltip, accordion...) **no leen la paleta ni la forma directamente**: consumen `--pi-theme-*`, y son los ficheros de `styles/themes/` los que deciden qué valor tiene cada token. Así, un tema nuevo no toca un solo selector de componente — solo redefine tokens (ver `_alternative.scss`, `_minimal.scss`, `_gradient.scss` como ejemplo).

Cómo se aplica un tema

La forma recomendada es `PiThemeService` — inyectable, con persistencia en `localStorage` y sin tener que tocar el DOM a mano:

```
import { inject } from '@angular/core';
import { PiThemeService } from '@pi/ui';

const themeService = inject(PiThemeService);

themeService.setTheme('alternative'); // pinta pi-shell pi-shell__alternative en <body>
themeService.toggleDarkMode(); // pinta/quita dark-mode en <html>
```

`theme` / `darkMode` son señales normales: también se pueden mutar con un `[(value)]` / `[(checked)]` dos vías directo sobre la señal (ver el uso real en `apps/playground/src/app/app.ts` / `app.html`). Por debajo, cada cambio dispara un `effect()` que aplica la clase y persiste — cualquier camino de escritura queda cubierto por igual, no solo los setters.

Si no quieres el servicio (p. ej. fuera de un contexto de inyección), el mecanismo de bajo nivel sigue disponible:

```
import { ShellClass } from '@pi/ui';

document.body.className = ShellClass.Alternative; // 'pi-shell pi-shell__alternative'
```

ShellClass expone Default, Alternative, Minimal y Gradient (libs/ui/src/lib/styles/configs.ts). Cada uno corresponde a un mixin en libs/ui/src/lib/styles/themes/_<nombre>.scss, incluido desde themes/index.scss. Dentro de un shell se puede forzar el aspecto contrario en un subárbol con .is-private / .is-public (p. ej. el input context de pi-card).

El patrón de fallback

Un token puede consumirse de dos formas, según si el tema por defecto lo declara o no:

```
// El tema Default declara este token → todos los temas lo tienen.
border-radius: var(--pi-theme-card-radius);

// El tema Default NO lo declara → el segundo argumento de var() reproduce
// el valor original hardcodeado. Solo Minimal (u otro tema futuro) lo pisa.
border-radius: var(--pi-theme-chip-radius, #{$chip-border-radius});
```

Esto es intencional: los tokens que **todos** los temas existentes necesitan personalizar (radio/altura/tipografía de botón, forma de campo) se declaran en _default.scss. Los que solo un tema concreto toca (radio de overlays en Minimal, degradado de foco en Gradient) se dejan sin declarar en Default —así su valor por defecto es exactamente el que tenía el componente antes de tokenizar, byte a byte, y ningún tema existente cambia de aspecto sin querer.

--pi-theme-* : tabla completa

D = Default · A = Alternative · M = Minimal · G = Gradient. Una celda vacía significa "no lo redefine, hereda el valor de la fila D (o el fallback de var()) si D tampoco lo declara, marcado como (fallback)".

Botón (pi-button/_pi-button.scss , _pi-button-theme.scss , menu/dynamic-menu)

Token	D	A	M	G
--pi-theme-button-height	var(--pi-spacing-xxx-large) (2.5rem)	2.875rem	2.5rem	—
--pi-theme-button-radius	var(--pi-border-radius-medium)	999px (píldora)	var(--pi-border-radius-xx-small)	—

--pi-theme-button-font-weight	600	700	500	—
--pi-theme-button-letter-spacing	normal	0.04em	normal	—
--pi-theme-button-text-transform	none	uppercase	none	—
--pi-theme-button-shadow	none	0 6px 16px -6px rgba(106,63,212,.55)	none	—
--pi-theme-button-transition	background-color .16s ease-in	all .2s ease-out	background-color .12s ease-out, border-color .12s ease-out	—
--pi-theme-button-fill-primary	none (relleno plano)	degradado primary-900→primary-400	none	degradado primary-400→primary-900
--pi-theme-button-fill-accent	none	degradado accent-900→accent-400	none	degradado accent-400→accent-900
--pi-theme-button-fill-warn	none	—	none	—
--pi-theme-button-fill-success	none	—	none	—
--pi-theme-button-fill-highlighted	none	degradado highlighted→highlighted-300	none	degradado highlighted-300→highlighted

`--pi-theme-button-fill-#{$color}` se interpola por nombre de color en `_pi-button-theme.scss:47` (`background-image: var(--pi-theme-button-fill-#{$color}, none)`), así que un tema nuevo que quiera degradado en un color que no sea primary/accent/highlighted solo tiene que declarar `--pi-theme-button-fill-<color>` .

Campos de formulario (`_mat-form-field.scss` , `file-upload` , `_mat-datepicker.scss` , `_mat-select.scss`)

Token	D	A	M	G
-------	---	---	---	---

--pi-theme-field-radius	0.875rem	999px	var(--pi-border-radius-xx-small)	—
--pi-theme-field-border-width	1px	2px	1px	—
--pi-theme-field-border-color	var(--pi-color-grey-200)	var(--pi-color-primary-100)	var(--pi-color-grey-200)	—
--pi-theme-field-border-color-hover	var(--pi-color-grey-300)	var(--pi-color-primary-300)	var(--pi-color-grey-400)	—
--pi-theme-field-border-color-focus	var(--pi-color-primary)	var(--pi-color-primary)	var(--pi-color-black)	transparent (deja ver el degradado)
--pi-theme-field-surface	var(--pi-color-white)	var(--pi-color-primary-25)	var(--pi-color-white)	—
--pi-theme-field-label-color	var(--pi-color-grey-300)	var(--pi-color-primary-900)	var(--pi-color-grey-400)	—
--pi-theme-field-label-transform (<i>fallback: none</i>)	—	uppercase	none	—
--pi-theme-field-label-letter-spacing (<i>fallback: normal</i>)	—	0.08em	normal	—
--pi-theme-field-label-weight (<i>fallback: inherit</i>)	—	700	500	—
--pi-theme-field-focus-gradient (<i>fallback: none</i>)	—	—	—	linear-gradient(120deg, primary, accent, highlighted)

Superficies: card / panel / chip

Token	D	A	M	G	Consumidores
--pi-theme-card-radius	var(--pi-border-radius-large)	1.5rem	var(--pi-border-radius-xx-small)	—	_pi-card.scss, skeleton.component.scss

--pi-theme-card-shadow	var(--pi-box-shadow-panel)	0 10px 30px -12px rgba(59,13,107,.28)	none	—	_pi-card.scss
--pi-theme-panel-radius	var(--pi-border-radius-large)	1.5rem	var(--pi-border-radius-xx-small)	—	panel.component.scss, _mat-datepicker.scss, _mat-select.scss, dynamic-menu
--pi-theme-chip-radius (<i>fallback: radio original por tamaño</i>)	—	0.25rem	0	—	chip/styles/_size.scss (x3, por tamaño y por affix)

Overlays: modal / notification / tooltip / accordion

Ningún tema salvo Minimal los redefine — Default/Alternative/Gradient usan siempre el *fallback* de `var()` (el valor que tenían antes de tokenizar).

Token	Fallback (D/A/G)	M	Consumidores
--pi-theme-modal-radius	var(--pi-border-radius-medium) (panel completo) / var(--pi-border-radius-large) (.pi-modal__panel--full)	var(--pi-border-radius-xx-small)	_mat-dialog.scss (x2)
--pi-theme-notification-radius	var(--pi-border-radius-large)	var(--pi-border-radius-xx-small)	snack-bar.component.scss, _mat-snackbar.scss (x2)
--pi-theme-notification-shadow	var(--pi-box-shadow)	none	_mat-snackbar.scss
--pi-theme-tooltip-radius	var(--pi-border-radius-medium)	var(--pi-border-radius-xx-small)	_mat-tooltip.scss
--pi-theme-tooltip-shadow	var(--pi-box-shadow-panel-double)	none	_mat-tooltip.scss
--pi-theme-accordion-radius	var(--pi-border-radius-large)	var(--pi-border-radius-xx-small)	accordion.component.scss, _horizontal-accordion.scss, _vertical-accordion.scss (x6, incluye un calc(... - 0.0625rem))

--pi-theme-overlay-divider-color	var(--pi-color-grey-100)	—	drawer.component.scss, hub.component.scss (borde bajo la cabecera)
--pi-theme-hub-grabber-color	var(--pi-color-grey-100)	—	hub.component.scss (el asa superior)

Puente con Angular Material (--mat-sys-*)

Los componentes de Material (slider, checkbox, radio, progress...) no leen --pi-color-* : pintan con sus propios roles --mat-sys-* , generados por mat.theme() a partir de una paleta separada.

_default.scss reengancha una cosa con otra **una sola vez**, por indirección:

```
--mat-sys-primary: var(--pi-color-primary);
--mat-sys-on-primary: var(--pi-color-white);
--mat-sys-primary-container: var(--pi-color-primary-100);
--mat-sys-on-primary-container: var(--pi-color-primary-900);
--mat-sys-secondary: var(--pi-color-accent);
--mat-sys-on-secondary: var(--pi-color-white);
--mat-sys-secondary-container: var(--pi-color-accent-100);
--mat-sys-on-secondary-container: var(--pi-color-accent-900);
--mat-sys-tertiary: var(--pi-color-highlighted);
--mat-sys-on-tertiary: var(--pi-color-white);
--mat-sys-error: var(--pi-color-warn);
--mat-sys-on-error: var(--pi-color-white);
```

Como es indirección (var(--pi-color-*) , no un valor literal), un tema que redefine --pi-color-primary etc. (como Alternative) tiñe también los controles de Material sin tener que repetir este bloque.

--pi-color-* : paleta de marca

Declarada una vez en :root (_pi-color-variables.scss) con los valores Default, y redeclarada completa dentro de .pi-shell__alternative (Minimal y Gradient **no** tocan la paleta — ver comentario en _minimal.scss : "minimalista aquí es una cuestión de forma, no de color"). Se consume **directamente por nombre** (no hay indirección --pi-theme-* de por medio) en ~100 ficheros de la librería: bordes, fondos, texto, iconos, estados de badge/chip/status, etc.

Cada paleta (primary , accent , highlighted , warn , success) sigue la misma escala de 10 pasos + variante rgba del tono 200:

10 → 25 → 50 → 75 → 100 → 200 → 200-rgba → 300 → 400 → (base) → 900

Escala	primary (D)	accent (D)	highlighted (D)	warn (D)
10	#fdfdff	#fffd9	#f9bfff	#fff9fa
25	#fafbff	#fffaed	#f2f5ff	#ffedf0
50	#f0f4ff	#fdf5e5	#e9eeff	#fcebeb
75	#eaefff	#fae5bf	#dae2ff	#ffe7ea

100	#d7e2fb	#fce0ad	#bccdfa	#f7cdcd
200	#c3d5ff	#f6d495	#6482d7	#f2acac
200- rgba	rgba(196,213,255,.15)	rgba(252,224,173,.15)	rgba(168,193,255,.15)	rgba(237,129,129,
300	#a3c0ff	#f2c26a	#436ac1	#ec8b8b
400	#8cacf2	#f6bc63	#1d3f8a	#e87272
base	#799ef0	#eda82a	#0d2a6b	#e45959
900	#5772d4	#d78f0f	#030f3c	#c13737

Neutros y utilidad, sin escala de tema (no cambian entre Default/Alternative):

Token	Valor	Uso
--pi-color-white	#ffffff	—
--pi-color-white-900	#f4f4f4	—
--pi-color-white-rgba-03/05/06/09	blanco a 33/50/66/88% opacidad	overlays, disabled
--pi-color-black-200	#323748	—
--pi-color-black	#1a1a1a	texto, foco (Minimal)
--pi-color-grey-10...900	escala de 10 pasos, #fcfcfd → #3d3d3d	bordes, texto secundario
--pi-color-background-300/400/500	#aeaeb6 / #fcfcfc / #f7f7f7	fondos de página

En `_alternative.scss` la paleta completa (`primary` , `accent` , `highlighted`) se redeclara con una gama azul→violeta / turquesa / violeta oscuro, respectivamente. `warn` y `success` se dejan intactos a propósito en todos los temas: son colores semánticos y cambiarlos perjudicaría la lectura del estado.

Nota: `_pi-color-variables.scss` declara `--pi-color-grey-200` dos veces (líneas 328-329) y `--pi-color-white` / `--pi-color-background-400` con el mismo patrón en otro punto del fichero — inofensivo (la segunda declaración gana y tiene el mismo valor) pero es limpieza pendiente si se retoca el fichero.

Primitivas base (sin tematizar)

Estas variables no cambian entre temas — son la escala de espaciado/forma/sombra de toda la librería, y los tokens `--pi-theme-*` casi siempre se expresan en función de ellas (`var(--pi-border-radius-large)` , `var(--pi-spacing-xxx-large)` ...).

Familia	Tokens	Rango
---------	--------	-------

Espaciado	<code>--pi-spacing-xxxx-small ... --pi-spacing-xxxx-large</code>	0.0625rem (1px) ... 3rem (48px), 11 pasos
Radio	<code>--pi-border-radius-xx-small ... --pi-border-radius-x-large</code>	0.125rem (2px) ... 1.375rem (22px), 5 pasos
Sombra	<code>--pi-box-shadow, -hover, -double, -panel, -panel-double, -panel-disabled, -top, -inset</code>	valores rgba fijos, ver <code>_pi-global-variables.scss</code>
Animación	<code>--pi-animation, -fast, -slow</code>	all .16s/.25s/.4s ease-out
Tamaño de fuente	<code>--pi-font-size-xxx-small ... -xxx-large</code>	0.5rem (8px) ... 2.875rem (46px), 9 pasos
Layout (header/footer/navbar/aside/card)	<code>--pi-height-*, --pi-width-*</code>	ver <code>_pi-global-variables.scss</code>
Landing/marketing	<code>--pi-public-font-size-*, --pi-public-height-*, --pi-public-width-*</code>	ver <code>_pi-public-variables.scss</code>

`_pi-gradient-variables.scss` (`$pi-gradient-primary-light` , etc.) son variables **Sass**, no custom properties de CSS — se resuelven en tiempo de compilación y no forman parte del contrato de theming en runtime.

Cómo tokenizar una superficie nueva

1. En el consumidor, cambia el valor hardcoded por `var(--pi-theme-<superficie>-<propiedad>, <valor-original>)` . El fallback garantiza que ningún tema existente cambia de aspecto hasta que decida redefinir el token.
2. Si **todos** los temas van a necesitar tocarlo pronto, decláralo también en `defaultThemeTokens` (`_default.scss`) con el valor original — así queda documentado como parte del contrato en vez de vivir solo como fallback disperso en el CSS.
3. Añade el override en el/los temas que lo necesiten (`_alternative.scss` , `_minimal.scss` , `_gradient.scss`).
4. Actualiza este documento.

Cómo añadir un tema nuevo

La forma recomendada es el generador:

```
nx g @pi/ui:theme mi-marca
```

Crea `libs/ui/src/lib/styles/themes/_mi-marca.scss` (en blanco, con los grupos de tokens más habituales comentados a modo de chuleta), lo registra en `themes/index.scss` , añade la entrada a `ShellClass` en `styles/configs.ts` y extiende `ThemeVariant` / `PiThemeService` — el tema queda seleccionable con `themeService.setTheme('mi-marca')` sin ningún paso manual más. Solo falta descomentar y ajustar los tokens que definen el aspecto, y (si tu app expone un selector de tema) añadirlo a esa lista.

Estos son los pasos que da por ti, si prefieres hacerlo a mano o entender qué toca:

1. Crea `libs/ui/src/lib/styles/themes/_<nombre>.scss` con un mixin `<nombre>ThemeTokens` que redefina solo los tokens `--pi-theme-*` (y, si aplica, `--pi-color-*`) que cambian respecto a Default.
2. Aplícalo a `.pi-shell__<nombre>`.
3. Impórtalo desde `themes/index.scss`.
4. Añade la entrada a `ShellClass` en `styles/configs.ts`.
5. Añade `'<nombre>'` a `ThemeVariant` y una entrada a `shellClassByVariant` en `util/theme/theme.service.ts`.

No hace falta tocar ningún componente: si consumen los tokens en vez de valores literales, el tema nuevo se propaga solo.

Lint de tokens: evitar que un color de marca se filtre sin querer

El bug que motivó tokenizar `accordion/tab/select-option/drawer/hub` esta misma sesión (un `var(--pi-color-*)` a pelo en el componente, así que ningún tema ni el modo oscuro puede redefinir esa superficie) tiene ahora una regla de stylelint que lo detecta automáticamente: `pi/no-raw-color-token` (`tools/stylelint-plugin-pi-theme/`).

```
nx run ui:lint-styles
```

Marca cualquier `--pi-color-*` que aparezca fuera del *fallback* de un `var(--pi-theme-*, ...)` — es decir, exactamente el patrón contrario al de ["El patrón de fallback"](#). Están exentos `styles/themes/**` (ahí `--pi-color-*` es legítimo: son las definiciones) y `styles/common/abstracts/**` (donde se declaran las primitivas de la paleta).

Estado actual: es un target aparte, no está enganchado a `nx lint` ni a CI. Al escribirse, señala ~500 usos preexistentes repartidos por componentes que esta sesión no llegó a tokenizar — adoptarlo como gate exige ir migrándolos primero (mismo patrón que en "Cómo tokenizar una superficie nueva"). Úsalo para no introducir *más* mientras se migra el resto.