

Cómo crear temas

Guía práctica del sistema de theming: qué tokens existen, cómo se aplican, y los pasos para dar de alta un tema nuevo sin tocar ni un selector de componente.



Basada en el estado real de `libs/ui/src/lib/styles`: los cuatro temas ya presentes en el repo (Default, Alternative, Minimal, Gradient) sirven de ejemplo en todo el documento. Incluye los tokens de overlays (modal, notificación, tooltip, accordion) añadidos tras la primera versión de esta guía.

- 1 Arquitectura: shell + tokens
- 2 El contrato de tokens
- 3 Crear un tema, paso a paso
- 4 Los cuatro temas del repo
- 5 Caso de estudio: anillo de foco en degradado
- 6 Modo oscuro y temas
- 7 Checklist

1 Arquitectura: shell + tokens

Dos piezas, ninguna es un selector de componente.

Un tema en `@pi/ui` no es un solo fichero que reescriba `.pi-button`, `.pi-card` o cualquier otro selector de componente. Es, exclusivamente, un bloque que redefine **variables CSS** (custom properties) bajo una clase contenedora. Los componentes ya están escritos para leer esas variables — cambiarlas es todo lo que hace falta.

La clase contenedora: `.pi-shell`

Todo tema se aplica añadiendo dos clases a un elemento raíz (normalmente `<body>`, porque los overlays de Material — diálogos, menús, snackbars — cuelgan de ahí y también deben heredar el tema):

```
<body class="pi-shell pi-shell__<nombre-del-tema>">
```

`.pi-shell` por sí sola ya aplica el contrato completo de tokens con sus valores por defecto (ver `themes/_default.scss`). `.pi-shell__<nombre>` se aplica *a la vez*, no en su lugar, y sobrescribe solo los tokens que ese tema quiere cambiar. Como ambas clases tienen la misma especificidad, gana la que se declaró más tarde en el CSS compilado — por eso el orden de `@forward` en `themes/index.scss` importa (ver §3).

El helper `shellTheming()` (en `libs/ui/src/lib/styles/configs.ts`) construye ese string por ti: `shellTheming('minimal')` devuelve `'pi-shell pi-shell__minimal'`. `ShellClass` ya expone uno por tema — no hace falta escribir el string a mano.

Las dos familias de variables

PREFIJO	QUÉ CONTROLA	EJEMPLO
<code>--pi-color-*</code>	Paleta de marca: primary, accent, highlighted, warn, success, grey, cada uno con escalones (−10 a −900).	<code>--pi-color-primary,</code> <code>--pi-color-primary-900</code>
<code>--pi-theme-*</code>	Forma y comportamiento: radios, sombras, grosor de borde, peso/transformación de texto, rellenos de botón.	<code>--pi-theme-button-radius,</code> <code>--pi-theme-card-shadow</code>

Un tema puede tocar una familia, la otra, o ambas. `_alternative.scss` cambia las dos (repaleta primary/accent/highlighted y la forma). `_minimal.scss` y `_gradient.scss` solo tocan `--pi-theme-*` — la paleta de marca se queda igual a propósito.

El puente con Angular Material

Los componentes nativos de Material (slider, checkbox, radio...) no leen `--pi-color-*`: pintan con sus propios roles `--mat-sys-*`, que genera `mat.theme()`. El puente se declara **una sola vez**, en `_default.scss`, por indirección:

```
--mat-sys-primary: var(--pi-color-primary);
--mat-sys-primary-container: var(--pi-color-primary-100);
/* ... */
```

Como es una indirección, cualquier tema que redefina `--pi-color-primary` arrastra automáticamente a Material sin tener que repetir el bloque del puente. Si tu tema no toca la paleta (como Minimal o Gradient), no necesitas tocar esto en absoluto.

Límite conocido: el puente cubre color, no forma. El radio/sombra de checkbox, radio, slider o mat-select vienen de tokens `--mdc-*` que Angular Material fija en tiempo de compilación (no son variables de runtime reasignables desde un tema). Aplanarlos también requeriría overrides por componente — fuera del alcance de este contrato.

2 El contrato de tokens

Todo lo que un tema puede redefinir, con su valor en el tema por defecto y dónde se consume.

Botones

TOKEN	POR DEFECTO	SE USA EN
<code>--pi-theme-button-height</code>	<code>var(--pi-spacing-xxx-large)</code> (2.5rem)	<code>pi-button/_pi-button-theme.scss</code>
<code>--pi-theme-button-radius</code>	<code>var(--pi-border-radius-medium)</code>	<code>pi-button/_pi-button.scss</code> , <code>dynamic-menu.component.scss</code>
<code>--pi-theme-button-font-weight</code>	<code>600</code>	<code>pi-button-theme.scss</code>
<code>--pi-theme-button-letter-spacing</code>	<code>normal</code>	
<code>--pi-theme-button-text-transform</code>	<code>none</code>	
<code>--pi-theme-button-shadow</code>	<code>none</code>	
<code>--pi-theme-button-transition</code>	<code>background-color .16s ease-in</code>	<code>pi-button-theme.scss</code> , <code>stepper.component.scss</code> , <code>file-upload.component.scss</code> , <code>dynamic-menu.component.scss</code>
<code>--pi-theme-button-fill-{primary,accent,warn,success,highlighted}</code>	<code>none</code> (manda el <code>background-color</code> sólido)	<code>_pi-button-theme.scss</code>

Campos de formulario

TOKEN	POR DEFECTO	SE USA EN
<code>--pi-theme-field-radius</code>	<code>0.875rem</code>	<code>vendors/_mat-form-field.scss</code> , <code>file-upload.component.scss</code> ,
<code>--pi-theme-field-border-width</code>	<code>1px</code>	<code>_mat-datepicker.scss</code> , <code>_mat-select.scss</code>
<code>--pi-theme-field-border-color</code>	<code>var(--pi-color-grey-200)</code>	
<code>--pi-theme-field-border-color-hover</code>	<code>var(--pi-color-grey-300)</code>	
<code>--pi-theme-field-border-color-focus</code>	<code>var(--pi-color-primary)</code>	
<code>--pi-theme-field-surface</code>	<code>var(--pi-color-white)</code>	
<code>--pi-theme-field-label-color</code>	<code>var(--pi-color-grey-300)</code>	
<code>--pi-theme-field-label-transform</code> / <code>-letter-spacing</code> / <code>-weight</code>	sin declarar → hereda el valor original del componente	<code>_mat-form-field.scss</code>
<code>--pi-theme-field-focus-gradient</code>	sin declarar → <code>none</code> (opt-in, ver §5)	<code>_mat-form-field.scss</code>

Superficies

TOKEN	POR DEFECTO	SE USA EN
<code>--pi-theme-card-radius</code>	<code>var(--pi-border-radius-large)</code>	<code>components/_pi-card.scss</code> , <code>skeleton.component.scss</code>
<code>--pi-theme-card-shadow</code>	<code>var(--pi-box-shadow-panel)</code>	<code>_pi-card.scss</code>
<code>--pi-theme-panel-radius</code>	<code>var(--pi-border-radius-large)</code>	<code>panel.component.scss</code> , <code>_mat-select.scss</code> , <code>_mat-datepicker.scss</code> ,

TOKEN	POR DEFECTO	SE USA EN
		dynamic-menu.component.scss
--pi-theme-chip-radius	sin declarar → radio original del chip	chip/styles/_size.scss (x3)

Fíjate en el patrón `var(--pi-theme-x, <valor-original>)` que usan varios consumidores (chip, panel, card, overlays). Eso significa que **no es obligatorio declarar todos los tokens** en un tema nuevo: los que omitas caen al valor que ya tenía el componente antes de que existiera el sistema de theming. Solo declara lo que quieras cambiar.

Overlays (modal, notificación, tooltip, accordion)

Grupo añadido después de la primera versión de este contrato. Ningún tema salvo **Minimal** los redefine todavía: Default, Alternative y Gradient usan siempre el segundo argumento de `var()`, que reproduce exactamente la forma que tenía cada overlay antes de tokenizar.

TOKEN	POR DEFECTO (FALLBACK)	SE USA EN
<code>--pi-theme-modal-radius</code>	sin declarar → <code>var(--pi-border-radius-medium)</code> (<code>var(--pi-border-radius-large)</code> en <code>.pi-modal__panel--full</code>)	<code>vendors/_mat-dialog.scss</code> (×2)
<code>--pi-theme-notification-radius</code>	sin declarar → <code>var(--pi-border-radius-large)</code>	<code>snack-bar.component.scss</code> , <code>_mat-snackbar.scss</code> (×2)
<code>--pi-theme-notification-shadow</code>	sin declarar → <code>var(--pi-box-shadow)</code>	<code>_mat-snackbar.scss</code>
<code>--pi-theme-tooltip-radius</code>	sin declarar → <code>var(--pi-border-radius-medium)</code>	<code>_mat-tooltip.scss</code>
<code>--pi-theme-tooltip-shadow</code>	sin declarar → <code>var(--pi-box-shadow-panel-double)</code>	<code>_mat-tooltip.scss</code>
<code>--pi-theme-accordion-radius</code>	sin declarar → <code>var(--pi-border-radius-large)</code>	<code>accordion.component.scss</code> , <code>_horizontal-accordion.scss</code> , <code>_vertical-accordion.scss</code> (×6, incluye un <code>calc()</code>)
<code>--pi-theme-overlay-divider-color</code>	sin declarar → <code>var(--pi-color-grey-100)</code>	<code>drawer.component.scss</code> , <code>hub.component.scss</code>
<code>--pi-theme-hub-grabber-color</code>	sin declarar → <code>var(--pi-color-grey-100)</code>	<code>hub.component.scss</code>

Minimal declara los seis primeros en su mixin (`_minimal.scss`): aplanar todos los radios a `--pi-border-radius-xx-small` y apaga las dos sombras (`none`) — mismo criterio que ya aplicaba a botón, campo y card, extendido al resto de superficies flotantes. Los dos últimos (`drawer/hub`) son más recientes y todavía ningún tema los redefine.

3 Crear un tema, paso a paso

Cuatro ficheros, ninguno toca un componente. Ejemplo con un tema ficticio `brutalist`.

Los cuatro pasos de esta página los da un generador: `nx g @pi/ui:theme brutalist`. Crea el fichero del paso 1 (en blanco, con los grupos de tokens más habituales comentados), hace los pasos 2 y 3, y además extiende `PiThemeService` (paso 4) para que el tema quede seleccionable sin ningún paso manual más. Lo que sigue es para quien prefiera hacerlo a mano o quiera entender qué toca cada paso.

1 Crea `libs/ui/src/lib/styles/themes/_brutalist.scss`

Un mixin con los tokens que cambian, más una clase que lo aplica:

```
@mixin brutalistThemeTokens {
  --pi-theme-button-radius: 0;
  --pi-theme-button-shadow: 4px 4px 0 var(--pi-color-black);
  --pi-theme-card-radius: 0;
  --pi-theme-card-shadow: none;
}

.pi-shell__brutalist {
  @include brutalistThemeTokens;
}
```

2 Regístralo en `themes/index.scss`

Añade el `@forward/@use` al final de la lista — el orden es el orden de prioridad en el CSS compilado:

```

@forward 'default';
@forward 'alternative';
@forward 'minimal';
@forward 'gradient';
@forward 'brutalist'; // ← nuevo, al final

@use 'default' as *;
@use 'alternative' as *;
@use 'minimal' as *;
@use 'gradient' as *;
@use 'brutalist' as *; // ← nuevo, al final

```

3 Añade la entrada en ShellClass (libs/ui/src/lib/styles/configs.ts)

```

export const ShellClass = {
  Default: shellTheming('default'),
  Alternative: shellTheming('alternative'),
  Minimal: shellTheming('minimal'),
  Gradient: shellTheming('gradient'),
  /* Bordes duros, sombra dura, sin medias tintas. */
  Brutalist: shellTheming('brutalist'),
};

```

4 Aplícalo en la app consumidora

Con PiThemeService (ver apps/playground/src/app/app.ts) basta con:

```

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

const themeService = inject(PiThemeService);
themeService.setTheme('brutalist');

```

Si el generador ya extendió ThemeVariant/shellClassByVariant (paso automático 4), no hace falta tocar nada más que este setTheme. Sin el servicio, el mecanismo de bajo nivel sigue funcionando igual:

```

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

document.body.className = ShellClass.Brutalist;

```

No hay quinto paso. No se toca ningún .component.scss, ningún selector, ningún componente TypeScript. Si necesitas cambiar algo que **no** está en el contrato de tokens (§2), esa es la señal de que hace falta añadir un token nuevo al contrato — y entonces sí tocas el .scss del componente afectado, pero solo una vez, para que quede disponible para todos los temas futuros (así se añadió --pi-theme-field-focus-gradient, ver §5).

4 Los cuatro temas del repo

Cuatro implementaciones reales del mismo contrato, en `libs/ui/src/lib/styles/themes/`.

● Default

Rellenos planos y sólidos, controles compactos. Declara el contrato completo — el resto de temas parten de aquí.

● Alternative

Aspecto de marketing: repaleta `primary`→violeta, `accent`→turquesa, botones en pastilla con degradado y sombra, campos con superficie tintada.

● Minimal

Esquinas casi rectas, sin sombra (separa por borde fino en vez de elevación), sin degradados ni versalitas. Ahora también aplana los overlays — modal, notificación, tooltip y accordion comparten el mismo radio recto. Misma paleta que Default.

● Gradient

Igual que Default en forma, pero con degradado en el relleno de los botones y en el anillo de foco de los campos.

Los cuatro comparten *exactamente* los mismos selectores de componente — la prueba de que el contrato de tokens funciona es que ninguno necesitó tocar un `.component.ts` ni un `.component.html`.

5 Caso de estudio: anillo de foco en degradado

Qué hacer cuando el efecto que quieres **no** lo cubre ningún token existente.

El tema *Gradient* quería un borde en degradado al enfocar un campo. El problema: el borde de un `<mat-form-field appearance="outline">` no es un border normal — es tres `<div>` (leading, notch, trailing) cada uno con su propio borde parcial, para dejar hueco a la etiqueta flotante. Un background en degradado ahí se vería cortado en cada unión.

La solución: un anillo aparte, no el borde del notch

En vez de pelear con los tres fragmentos, se añade un `::after` sobre `.mat-mdc-text-field-wrapper` (la caja completa, sin partir) que pinta *solo el borde* gracias a `mask-composite`: se enmascara la copia `padding-box` (el interior) fuera de la copia `border-box` (el rectángulo completo), dejando visible únicamente el anillo:

libs/ui/src/lib/styles/common/vendors/_mat-form-field.scss

```
.mat-mdc-text-field-wrapper { position: relative; }

&.mat-focused .mat-mdc-text-field-wrapper::after {
  content: '';
  position: absolute;
  inset: 0;
  z-index: 1;
  border-radius: var(--pi-theme-field-radius);
  padding: var(--pi-theme-field-border-width, 1px);
  background: var(--pi-theme-field-focus-gradient, none);
  -webkit-mask:
    linear-gradient(#fff 0 0) content-box,
    linear-gradient(#fff 0 0);
  -webkit-mask-composite: xor;
  mask-composite: exclude;
  pointer-events: none;
}
```

El token nuevo, `--pi-theme-field-focus-gradient`, se añadió al contrato compartido (consumido con fallback `none`) — así cualquier tema puede optar por él sin que los demás se vean afectados. *Gradient* además pone `--pi-theme-field-border-color-focus: transparent` para apagar el borde sólido del notch; si no, se vería el borde plano por debajo del anillo.



Sin degradado
`focus-gradient: none`



Con degradado
(anillo vía `mask-composite`)

Regla general: cuando el efecto que buscas no cabe en una simple reasignación de variable, la solución sigue siendo "un token opt-in con fallback inerte" + "una técnica CSS aislada en el fichero compartido del componente" — nunca un selector nuevo dentro del propio fichero del tema. El fichero de tema solo declara *valores*.

6 Modo oscuro y temas

El modo oscuro del playground es ortogonal a los temas — conviven, no se solapan.

@pi/ui no trae un tema oscuro propio. El playground activa uno "best-effort" añadiendo la clase `dark-mode` al `<html>` (no al `<body>`, que es donde vive `.pi-shell`) — hoy vía `PiThemeService.setDarkMode()/toggleDarkMode()`, que hace ese toggle por ti (persistido en `localStorage`) en vez de que cada app repita el mismo `effect()` — y ahí reasigna solo los **neutros** (`--pi-color-white`, `--pi-color-grey-*`, `--pi-color-black`) — las paletas de marca (`primary`, `accent`...) se dejan igual:

`apps/playground/src/app/styles.scss`

```
html.dark-mode {
  --pi-color-white: #1c1f26;
  --pi-color-grey-100: #2a2e38;
  --pi-color-black: #f1f2f4;
  /* ... solo neutros, la marca no cambia ... */
}
```

Como es una clase distinta en un elemento distinto, cualquier combinación (tema × modo oscuro) funciona sin código extra — son dos cascadas independientes que solo coinciden en que ambas reasignan variables CSS. La única excepción real es cuando un tema usa un escalón de marca *como si fuera* superficie neutra (por ejemplo, `--pi-color-primary-25` como fondo de campo en `Alternative`): ese caso puntual sí necesita un ajuste en modo oscuro, ya presente en `styles.scss` bajo `html.dark-mode .pi-shell__alternative`.

7 Checklist

Antes de dar el tema por terminado

- ✓ Se creó con `nx g @pi/ui:theme <nombre>` — hace los tres puntos siguientes por ti.
- ✓ El fichero nuevo vive en `themes/`, exporta un mixin y una clase `.pi-shell__<nombre>`.
- ✓ Está en `themes/index.scss`, al final de ambos bloques (`@forward` y `@use`).
- ✓ Tiene entrada en `ShellClass` (`configs.ts`) y en `PiThemeService` (`ThemeVariant/shellClassByVariant`), con un comentario de una línea que resuma el aspecto.
- ✓ Solo declara los tokens que cambia — el resto hereda de `_default.scss` sin repetirlo.
- ✓ Si toca `--pi-color-primary/accent/highlighted`, revisado que el puente `--mat-sys-*` (declarado una vez en `default`) sigue coherente.
- ✓ Si toca `overlays` (`modal/notificación/tooltip/accordion`), revisado con las tres páginas de catálogo, no solo con un botón suelto.
- ✓ Si al tokenizar una superficie nueva tocaste un `.component.scss`, `nx run ui:lint-styles` no marca ese fichero (detecta `--pi-color-*` usado fuera de un fallback `--pi-theme-*`).
- ✓ Probado en el playground con las tres páginas de catálogo, no solo en un botón suelto.

Evita

- ✗ Añadir un selector de componente (`.pi-button.mi-tema`) dentro del fichero del tema — si hace falta, el token que falta va al contrato compartido, no aquí.
- ✗ Redefinir tokens que no cambian "por si acaso" — genera ruido y dificulta ver qué es realmente distinto en ese tema.
- ✗ Tocar `warn / success` por estética — son colores semánticos; cambiarlos perjudica la lectura del estado (`error/éxito`).
- ✗ Asumir que `--pi-theme-*` alcanza a los controles nativos de Material (`checkbox`, `radio`, `slider`): esos usan tokens `--mdc-*` fijados en compilación, fuera de este contrato (ver §1).