

can we continue this chat here:
<https://chatgpt.com/c/6892d673-ac80-8323-a047-4f6476f2fe44>

I can't move or join another chat directly, but you can paste anything from that conversation here and we can pick up right where you left off.

Do you want to copy the last part of that chat so we can continue?



because the other caht is fully lengh alreayd if i cpy and past won work

Got it — when a chat hits its limit, copying everything over here in one go won't work because it'll still be too long.

Here's what I suggest:

- Bring over just the **key parts or last relevant section** from that chat (what we need to continue from).
- I can **summarize the earlier context** in a compact way so we don't lose any important details.
- Then we can keep going here without running into the limit.

Do you want me to create a **short but precise recap** of the old chat so we can continue smoothly?



we are buildin a design token for comp0oentn make the most complete possible going compoennt by compoennt
progress bar

Perfect — let's build the **Progress Bar** design tokens fully and systematically so it's scalable, theme-friendly, and covers all use cases.

Here's a complete breakdown:

Progress Bar – Design Tokens

🎨 Color Tokens

- `progressBar.track.background` – Background color of the track (empty area)
 - `progressBar.track.border` – Border color of the track (if applicable)
 - `progressBar.fill.background` – Fill/indicator color for progress
 - `progressBar.fill.gradient` – Gradient or pattern applied to the fill (optional)
 - `progressBar.fill.success` – Fill color when progress completes successfully
 - `progressBar.fill.error` – Fill color when progress indicates failure/error
 - `progressBar.fill.warning` – Fill color when progress is in a warning state
 - `progressBar.text.color` – Color of progress label (percentage or status)
-

📏 Size & Dimension Tokens

- `progressBar.height.sm` – Small height (compact)
 - `progressBar.height.md` – Default height
 - `progressBar.height.lg` – Large height (accessible/focus)
 - `progressBar.width.default` – Default width (if fixed)
 - `progressBar.border.radius` – Border radius of the progress track & fill
-

🔤 Typography Tokens

- `progressBar.label.fontFamily` – Font for percentage text or label
 - `progressBar.label.fontSize.sm` – Font size for small progress bar
 - `progressBar.label.fontSize.md` – Font size for default progress bar
 - `progressBar.label.fontWeight` – Font weight for percentage text
 - `progressBar.label.textAlignment` – Alignment of label (inside, outside, or centered)
-

⚡ Animation Tokens

- `progressBar.animation.duration` – Duration of fill transition
 - `progressBar.animation.easing` – Easing function for fill movement
 - `progressBar.animation.delay` – Delay before animation starts (optional)
-

State Tokens

- `progressBar.state.loading` – Default progress animation behavior
 - `progressBar.state.ineterminate.background` – Color or pattern for indeterminate mode
 - `progressBar.state.paused.background` – Color for paused state
 - `progressBar.state.disabled.opacity` – Opacity when disabled
-

Structure Tokens

- `progressBar.gap` – Space between label and bar (if external label)
 - `progressBar.padding` – Inner padding if the bar has an outline style
 - `progressBar.icon.size` – Size of any icon shown alongside progress label
-

Example token naming in JSON style:

json

 Copy code

```
{  
  "progressBar": {  
    "track": {  
      "background": "{colors.neutral.200}",  
      "border": "{colors.neutral.300}"  
    },  
    "fill": {  
      "background": "{colors.primary.500}",  
      "gradient": "linear-gradient(90deg, {colors.primary.500}, {colors.pr  
      "success": "{colors.success.500}",  
      "error": "{colors.error.500}",  
      "warning": "{colors.warning.500}"  
    },  
    "label": {  
      "fontFamily": "{typography.fontFamily.base}",  
      "fontSize": {  
        "sm": "12px",  
        "lg": "14px"  
      }  
    }  
  }  
}
```

```

        "md": "14px"
    },
    "fontWeight": "{typography.fontWeight.medium}",
    "color": "{colors.neutral.900}"
},
"height": {
    "sm": "4px",
    "md": "8px",
    "lg": "12px"
},
"border": {
    "radius": "{radius.sm}"
},
"animation": {
    "duration": "300ms",
    "easing": "ease-in-out"
},
"state": {
    "disabled": {
        "opacity": 0.4
    }
}
}
}

```

👉 Question: Do you want me to **include circular/radial progress tokens here as well** (so Progress Bar covers both linear and circular styles), or keep them separate? This will decide how we group naming (`progressBar.linear` & `progressBar.circular`).



no json need

I also have this container that I wrappe thing inside

Got it — no JSON, just clear, structured token naming.

Also, since you have a **container wrapper** around the progress bar, we'll add tokens for

that too.

Progress Bar – Tokens (No JSON)

Color Tokens

- **Track**
 - `progressBar.track.background`
 - `progressBar.track.border`
 - **Fill**
 - `progressBar.fill.background`
 - `progressBar.fill.gradient`
 - `progressBar.fill.success`
 - `progressBar.fill.warning`
 - `progressBar.fill.error`
 - **Label/Text**
 - `progressBar.label.color`
-

Size & Spacing Tokens

- **Height**
 - `progressBar.height.sm`
 - `progressBar.height.md`
 - `progressBar.height.lg`
 - **Border Radius**
 - `progressBar.border.radius`
 - **Container Padding & Gap**
 - `progressBar.container.padding`
 - `progressBar.container.gap` (space between label and bar)
-

Typography Tokens

- `progressBar.label.font.family`
- `progressBar.label.font.size.sm`
- `progressBar.label.font.size.md`
- `progressBar.label.font.weight`
- `progressBar.label.alignment` (inside, outside, centered)

⚡ Animation Tokens

- `progressBar.animation.duration`
 - `progressBar.animation.easing`
 - `progressBar.animation.delay`
-

🛠 State Tokens

- `progressBar.state.loading` (default motion)
 - `progressBar.state.indeterminate.background`
 - `progressBar.state.paused.background`
 - `progressBar.state.disabled.opacity`
-

✳️ Container Tokens (Wrapper)

If your container is a styled wrapper around the progress bar (for background, elevation, framing, etc.):

- `progressBar.container.background`
 - `progressBar.container.border`
 - `progressBar.container.border.radius`
 - `progressBar.container.shadow`
 - `progressBar.container.width`
-

👉 Question — is your **container** just a neutral holder for spacing/alignment, or does it have **its own background, border, or shadow** (like a card/section)? That will decide how many tokens we add for it.



I need a container component where I use to put content inside

Ah, I see now — you're talking about a **generic Container component** (used as a layout wrapper for any content, not just the progress bar).

Let's define **complete design tokens** for a versatile Container that you can use across the system.

Container – Tokens

Color Tokens

- `container.background` – Default background color
 - `container.background.surface` – Variant for surfaces (light/dark modes)
 - `container.background.elevated` – Elevated state background (e.g., over another surface)
 - `container.border.color` – Border color (if applied)
-

Size & Spacing Tokens

- **Padding** (internal spacing)
 - `container.padding.sm`
 - `container.padding.md`
 - `container.padding.lg`
 - **Margin** (external spacing if standardized)
 - `container.margin.sm`
 - `container.margin.md`
 - `container.margin.lg`
 - **Width**
 - `container.width.full` (100%)
 - `container.width.max` (max content width)
 - **Border Radius**
 - `container.border.radius.sm`
 - `container.border.radius.md`
 - `container.border.radius.lg`
-

Shadow & Elevation Tokens

- `container.shadow.none`

- container.shadow.sm
 - container.shadow.md
 - container.shadow.lg
-

AB
CD

Typography Tokens (if container has titles or headings)

- container.header.font.family
 - container.header.font.size
 - container.header.font.weight
 - container.header.color
-



State Tokens

- container.state.default – Normal background/border/shadow
 - container.state.hover – Hover background/border/shadow
 - container.state.focus – Focus outline or shadow
 - container.state.disabled.opacity
-



This makes the container **modular enough to be:**

- Page section wrapper
 - Card-like component
 - Content holder for modals, forms, etc.
-



Question: Do you want this **Container** to also support **variants** like:

- container.variant.card (elevated)
- container.variant.surface (flat)
- container.variant.outlined (border only)

That will change how we structure the tokens (variants vs. base). Do you want to go that route?



icon

Perfect — now let's define the **Icon component** tokens (so it works across the system, covers sizing, colors, states, and alignment).

Icon – Tokens

Color Tokens

- `icon.color.default` – Standard icon color
 - `icon.color.primary` – Primary brand color icon
 - `icon.color.secondary` – Secondary brand color icon
 - `icon.color.muted` – Lower emphasis / disabled tone
 - `icon.color.success` – Success state
 - `icon.color.warning` – Warning state
 - `icon.color.error` – Error state
 - `icon.color.inverted` – For use over dark backgrounds
-

Size Tokens

- `icon.size.xs` – Very small icons (decorative UI elements)
 - `icon.size.sm` – Small icons (buttons, form fields)
 - `icon.size.md` – Default size
 - `icon.size/lg` – Large icons (prominent display)
 - `icon.size.xl` – Extra-large icons (illustrative use)
-

State Tokens

- `icon.state.default.opacity`
 - `icon.state.hover.color`
 - `icon.state.active.color`
 - `icon.state.disabled.opacity`
-

Stroke & Shape Tokens

- `icon.stroke.width.sm` – Thin stroke icons
 - `icon.stroke.width.md` – Default stroke weight
-

- `icon.stroke.width.lg` – Bold stroke weight
 - `icon.corner.radius` – Rounding for icons with a background shape
-

💡 Layout Tokens

- `icon.spacing.inline` – Space between icon and adjacent text
 - `icon.alignment.vertical` – Align middle/top/bottom in line with text
 - `icon.container.padding` – Padding when icon is inside a shaped background
 - `icon.container.background` – Background color for icon containers (optional)
-

👉 Quick check:

Do you want to treat **Icon Buttons** (circle, square, filled, outlined) as a **separate component** with its own tokens, or keep them under `icon` with variants like `icon.variant.button.circle` and `icon.variant.button.square`?

That will decide if we split the token sets or keep them unified.



i need icons but not buttons, but with square or circle just more about visual design than actions

Ah, perfect — so we're not talking about **interactive icon buttons**, but **icons as visual elements** that can live inside a square or circle frame purely for design (status, decoration, emphasis).

Let's set up tokens for **Icons (standalone + framed styles)**:

Icon – Tokens (Visual Only)

🎨 Color Tokens

- **Base Icon Colors**
 - `icon.color.default` – Neutral icon color
 - `icon.color.primary` – Brand primary

- `icon.color.secondary` – Brand secondary
 - `icon.color.success` – Success
 - `icon.color.warning` – Warning
 - `icon.color.error` – Error
 - `icon.color.inverted` – For dark backgrounds
 - **Background (Framed Icons)**
 - `icon.background.circle` – Circle background color
 - `icon.background.square` – Square background color
 - `icon.background.transparent` – No background
-

Size Tokens

- **Icon Sizes**
 - `icon.size.xs` – 12px
 - `icon.size.sm` – 16px
 - `icon.size.md` – 20px
 - `icon.size.lg` – 24px
 - `icon.size.xl` – 32px
 - **Background Sizes (Circle/Square)**
 - `icon.container.size.sm` – Small frame (e.g. 24px)
 - `icon.container.size.md` – Medium frame (e.g. 32px)
 - `icon.container.size.lg` – Large frame (e.g. 40px)
-

Shape & Radius Tokens

- `icon.container.shape.circle` – Fully rounded (50%)
 - `icon.container.shape.square` – Square corners (radius token can apply)
 - `icon.container.border.radius.sm` – Small rounding for square
 - `icon.container.border.radius.md` – Medium rounding for square
-

Visual Style Tokens

- `icon.container.border.width` – Border thickness (if framed)
- `icon.container.border.color` – Border color for framed icons
- `icon.container.shadow` – Drop shadow for elevated look
- `icon.container.padding` – Space between icon and container edge

💡 This makes it flexible enough for:

- Status icons (circle or square backgrounds for emphasis)
 - Decorative icons (brand accents)
 - Informational callouts (with framed styling)
-

👉 Quick check for naming — do you want to **separate the framed icons** as their own type (e.g. `iconFramed.circle.primary`) or keep everything under `icon.container` like above?

This will impact how cleanly we can swap between **plain icons** and **decorative framed icons** in your theme.



but is the icon + a square?

Exactly — in your case it sounds like it's:

👉 **Icon (the glyph) + a square or circle background (purely decorative, not a button)**

So the tokens need to treat it as **two layers**:

1. **Icon glyph styling** (size, color, weight)
 2. **Container/frame styling** (shape, background, border, padding)
-

📌 Tokens for “Icon + Shape”

1 Icon (Glyph)

- `icon.color.default`
- `icon.color.primary`
- `icon.size.xs / sm / md / lg / xl`
- `icon.stroke.width.sm / md / lg` (for outline-style icons)

2 Container (Square or Circle Background)

• Shape & Size

- `icon.container.shape.circle` (`border-radius: 50%`)

- `icon.container.shape.square` (`border-radius: token`)
 - `icon.container.size.sm / md / lg` (overall container size)
 - **Background & Border**
 - `icon.container.background.default`
 - `icon.container.background.primary`
 - `icon.container.border.width`
 - `icon.container.border.color`
 - **Spacing**
 - `icon.container.padding.sm / md / lg` (space between glyph and container edge)
 - **Shadow (if used for elevation)**
 - `icon.container.shadow.sm / md / lg`
-

 **Example in use:**

- A **checkmark icon** in a **green circle** for success
 - A **bell icon** in a **neutral square** for notifications
 - A **warning icon** in a **yellow circle**
-

 Question — do you want these to be **theme-driven decorative assets** (so colors adapt automatically between light/dark themes), or do you want to **manually define background + icon colors** per use case?

That will decide whether we create `icon.container.background.surface` & `icon.color.onSurface` type tokens.



make it more advanced and complete

Perfect — let's make this **as advanced and complete as possible** so your "Icon + Shape" system works for **every visual case** (status, accent, decoration, data visual emphasis) and is **future-proof for theming**.

We'll treat it as **two layered components**:

1. **Icon glyph** (vector symbol itself)
 2. **Container frame** (square, circle, or custom shape)
-

ICON + SHAPE – Advanced Design Tokens

1 ICON (Glyph Layer)

Color Tokens (Adaptive & State-Based)

- `icon.color.default` – Standard neutral color
 - `icon.color.brand.primary` – Primary brand color
 - `icon.color.brand.secondary` – Secondary brand color
 - `icon.color.surface` – For use on surfaces (e.g., inverted for dark mode)
 - `icon.color.onPrimary` – Icon color when on a primary background
 - `icon.color.onSecondary` – Icon color when on a secondary background
 - `icon.color.success` – Success indication
 - `icon.color.warning` – Warning indication
 - `icon.color.error` – Error indication
 - `icon.color.info` – Informational indication
 - `icon.color.muted` – Low emphasis, subdued icons
-

Size Tokens

- `icon.size.xs` – 12px (micro-icons)
 - `icon.size.sm` – 16px
 - `icon.size.md` – 20px (default UI icon)
 - `icon.size/lg` – 24px
 - `icon.size.xl` – 32px (feature emphasis)
 - `icon.size.xxl` – 40px (hero/illustrative icon)
-

Stroke & Style Tokens

- `icon.stroke.width.thin` – 1px
- `icon.stroke.width.medium` – 1.5px
- `icon.stroke.width.bold` – 2px
- `icon.fill.opacity` – Fill transparency level for filled icons

- `icon.stroke.opacity` – Stroke transparency level for outline icons
-

2 CONTAINER (Shape Layer)

Shape & Background Colors

- `icon.container.shape.circle` – Circular frame
- `icon.container.shape.square` – Square frame
- `icon.container.shape.rounded` – Rounded rectangle frame
- `icon.container.shape.pill` – Capsule frame

Background Colors (Theme Adaptive)

- `icon.container.background.default` – Neutral
- `icon.container.background.primary` – Brand primary background
- `icon.container.background.secondary` – Brand secondary background
- `icon.container.background.surface` – Matches surface background tone
- `icon.container.background.elevated` – For raised elements (slight elevation)
- `icon.container.background.transparent` – No fill background

Borders

- `icon.container.border.width.none` – No border
 - `icon.container.border.width.thin` – 1px
 - `icon.container.border.width.medium` – 2px
 - `icon.container.border.color.default` – Neutral border
 - `icon.container.border.color.primary` – Brand border
 - `icon.container.border.color.onBackground` – Border color matching the background
-

Size & Spacing

- `icon.container.size.sm` – Small frame (24px)
 - `icon.container.size.md` – Medium frame (32px)
 - `icon.container.size.lg` – Large frame (40px)
 - `icon.container.size.xl` – Extra large frame (48px+)
 - `icon.container.padding.sm` – Minimal space between glyph and frame
 - `icon.container.padding.md` – Standard space between glyph and frame
 - `icon.container.padding/lg` – Larger padding for emphasis
-

Shadow & Depth

- `icon.container.shadow.none` – Flat
 - `icon.container.shadow.sm` – Subtle shadow for soft elevation
 - `icon.container.shadow.md` – Moderate elevation
 - `icon.container.shadow.lg` – High elevation (e.g., floating elements)
-

3 COMBINED BEHAVIORS (Icon + Shape System)

Variant

- `icon.variant.filledCircle` – Circle background filled, glyph on top
- `icon.variant.outlinedCircle` – Circle border only
- `icon.variant.filledSquare` – Square background filled
- `icon.variant.outlinedSquare` – Square border only
- `icon.variant.transparent` – No background, glyph only

Theme Adaptation

- `icon.color.onLight` – Adjusted color for light backgrounds
- `icon.color.onDark` – Adjusted color for dark backgrounds
- `icon.container.background.light` – Background optimized for light theme
- `icon.container.background.dark` – Background optimized for dark theme

Alignment & Layout

- `icon.alignment.inline` – Align inline with text
 - `icon.alignment.center` – Centered in container
 - `icon.alignment.block` – Standalone (block-level alignment)
-

This setup lets you handle:

- Status indicators (success, warning, error)
 - Decorative accents (brand icons inside shapes)
 - Informational callouts (emphasized icons in UI)
 - Fully theme-adaptive icons (light/dark mode ready)
-

👉 I have one question before I finalize this set into a **clean hierarchy**:

Do you want to **treat the shapes (circle, square, pill) as a separate "iconFrame"**

component for clarity (`iconFrame.circle` , `iconFrame.square`), or keep everything nested under `icon.container.shape` like I wrote above?

This decision will make your token structure **much cleaner for theming**.



this is the list of component I already built. I need now make thee semantic tier 2, make all possibilites possible:

Below is a naming blueprint for each of the components in your list. It follows the three-tier approach—primitives, semantic and component-level tokens—and uses clear, descriptive names as recommended in design-system best practices

intodesignsystems.com

thedesignsystem.guide

. Adjust the colours, spacing values and prefixes to match your brand.

Buttons

Structural tokens

`button-border-radius`: general radius for all buttons.

`button-padding-x` / `button-padding-y`: horizontal and vertical padding.

`button-height`: default button height.

Style tokens

`button-primary-background`, `button-primary-text`,
`button-primary-border-color`.

`button-secondary-background`, `button-secondary-text`.

`button-destructive-background`, `button-destructive-text`.

Add state variations like `button-primary-background`

hover, button-primary-background-disabled.

Payment navigation button

Tokens mirror those for a primary button but reflect its context:

button-payment-nav-background, button-payment-nav-text, button-payment-nav-icon-color.

States: button-payment-nav-background-hover, button-payment-nav-background-disabled.

Action button

Tokens emphasise an affordance or call-to-action:

button-action-background, button-action-text, button-action-icon-color.

States: button-action-background-hover, button-action-background-disabled.

Checkboxes

checkbox-size, checkbox-border-radius.

checkbox-unchecked-background, checkbox-unchecked-border-color.

checkbox-checked-background, checkbox-checked-border-color.

checkbox-checkmark-color.

checkbox-label-color (for text).

Forms

Forms are composed of inputs, labels and groupings; you can use:

form-background, form-border-radius, form-padding.

form-section-spacing: spacing between stacked fields.

Use input tokens (below) for each field type within a

form.

Headers

header-background, header-height, header-padding-x, header-padding-y.

header-title-color, header-title-font-size, header-title-font-weight.

header-subtitle-color (if needed).

header-border-color: for bottom border or divider.

Inputs (text fields)

input-background, input-border-color, input-border-radius.

input-text-color, input-placeholder-color, input-label-color.

input-padding-x, input-padding-y.

States: input-border-color-focus, input-background-disabled.

List item

list-item-background, list-item-hover-background, list-item-selected-background.

list-item-text-color, list-item-subtext-color.

list-item-padding-x, list-item-padding-y.

list-item-border-bottom-color: for separators.

Modal

modal-overlay-color: semi-transparent background behind the dialog.

modal-background, modal-border-radius, modal-shadow.

modal-title-color, modal-title-font-size, modal-content-

padding.

modal-close-icon-color.

Radio button

radio-size, radio-border-radius (often circle).

radio-unchecked-background, radio-unchecked-border-color.

radio-checked-background, radio-checked-border-color.

radio-dot-color (the inner filled circle).

radio-label-color.

Radio button group

radio-group-spacing: horizontal or vertical spacing between options.

radio-group-alignment: could be semantic (horizontal / vertical).

Group label tokens: radio-group-label-color, radio-group-label-font-size.

Toggle switches

toggle-switch-width, toggle-switch-height.

toggle-switch-border-radius.

toggle-switch-off-background, toggle-switch-on-background.

toggle-switch-thumb-color.

Optional state tokens: toggle-switch-off-border-color, toggle-switch-on-border-color.

Toggle segmentation (segmented control)

segmented-control-background, segmented-control-border-radius.

segmented-control-option-spacing: space between segments.

segmented-control-option-background, segmented-control-option-text-color.

segmented-control-option-selected-background, segmented-control-option-selected-text-color.

segmented-control-divider-color (if separators between segments).

File uploader

file-uploader-background, file-uploader-border-color, file-uploader-border-radius.

file-uploader-icon-color, file-uploader-text-color.

Hover states: file-uploader-background-hover, file-uploader-border-color-hover.

file-uploader-error-text-color for error messages.

Toggle chips (chip buttons)

chip-height, chip-border-radius, chip-padding-x, chip-padding-y.

chip-background, chip-text-color, chip-border-color.

chip-selected-background, chip-selected-text-color, chip-selected-border-color.

Segmented controls (same as toggle segmentation)

Use the tokens listed under toggle segmentation.

Kebab menu (3-dot context menu)

kebab-menu-icon-color.

kebab-menu-background, kebab-menu-border-radius, kebab-menu-shadow.

kebab-menu-item-background, kebab-menu-item-hover-background.

kebab-menu-item-text-color.

Range slider

range-slider-track-color, range-slider-track-height.

range-slider-progress-color (filled portion).

range-slider-thumb-color, range-slider-thumb-size,
range-slider-thumb-border-radius.

Hover or active tokens: range-slider-thumb-color-active,
range-slider-track-color-hover.

Drop down

dropdown-background, dropdown-border-color,
dropdown-border-radius.

dropdown-item-background, dropdown-item-hover-
background, dropdown-item-selected-background.

dropdown-item-text-color, dropdown-item-selected-
text-color.

dropdown-divider-color for separators.

Slider (linear slider for numeric values)

Same tokens as range slider; you might prefer:

slider-track-color, slider-active-track-color.

slider-thumb-color, slider-thumb-size.

slider-label-color (if labels or ticks are displayed).

Tables

table-border-color, table-border-radius.

table-header-background, table-header-text-color,
table-header-font-weight.

table-row-background, table-row-hover-background,
table-row-selected-background.

table-row-text-color, table-row-alt-background (for zebra striping).

table-cell-padding-x, table-cell-padding-y.

Tooltip

tooltip-background, tooltip-text-color, tooltip-border-radius.

tooltip-shadow.

tooltip-padding-x, tooltip-padding-y.

tooltip-arrow-size if arrows are used.

Accordion

accordion-header-background, accordion-header-text-color.

accordion-header-icon-color (for the chevron).

accordion-header-padding-x, accordion-header-padding-y.

accordion-body-background, accordion-body-text-color.

accordion-border-color, accordion-border-radius.

Badges

badge-background, badge-text-color, badge-border-color.

badge-padding-x, badge-padding-y, badge-border-radius.

Variants like badge-success-background, badge-error-background, etc.

Cards

card-background, card-border-color, card-border-radius, card-shadow.

card-padding-x, card-padding-y.

card-title-color, card-subtitle-color, card-body-color.

card-image-height if images are part of the pattern.

Content separator

separator-color, separator-thickness, separator-margin-y.

Empty state

empty-state-background, empty-state-border-radius.

empty-state-icon-color, empty-state-title-color, empty-state-description-color.

empty-state-spacing: vertical spacing between icon/title/description.

empty-state-action-button-... if actions are included.

Inline message

inline-message-background, inline-message-border-color, inline-message-border-radius.

inline-message-icon-color, inline-message-text-color, inline-message-link-color.

Variants: inline-message-success-background, inline-message-error-background, etc.

Loading indicators

loading-spinner-size, loading-spinner-color.

loading-overlay-background: if there's a backdrop.

loading-text-color (for messages like "Loading...").

Page layout

page-background, page-max-width, page-margin-x, page-padding-y.

page-section-spacing: vertical spacing between

sections.

page-sidebar-width if you have side navigation.

Pagination

pagination-background, pagination-border-color.

pagination-button-background, pagination-button-text-color.

pagination-button-hover-background, pagination-button-selected-background, pagination-button-selected-text-color.

pagination-spacing: space between page numbers.

Progress bar

progress-bar-background, progress-bar-height, progress-bar-border-radius.

progress-bar-track-color, progress-bar-filled-color.

progress-bar-indeterminate-color (for animated stripes).

Progress tracker

progress-tracker-step-size (diameter of step indicators).

progress-tracker-line-color.

progress-tracker-completed-color, progress-tracker-active-color, progress-tracker-upcoming-color.

progress-tracker-label-color, progress-tracker-label-font-size.

Tabs

tabs-background, tabs-border-color (bottom border).

tab-padding-x, tab-padding-y.

tab-text-color, tab-text-color-hover, tab-text-color-active.

tab-background-hover, tab-background-active.

tab-indicator-color, tab-indicator-height.

Tags

tag-background, tag-border-color, tag-border-radius.

tag-text-color.

Variants: tag-primary-background, tag-secondary-background, etc.

tag-padding-x, tag-padding-y.

Toggletip (tooltip triggered by a toggle)

toggletip-background, toggletip-text-color, toggletip-border-radius.

toggletip-shadow.

toggletip-padding-x, toggletip-padding-y.

toggletip-icon-color if there's an icon to trigger it.

Snackbars and toasts

snackbar-background, snackbar-text-color, snackbar-border-radius, snackbar-shadow.

snackbar-icon-color.

snackbar-padding-x, snackbar-padding-y.

snackbar-spacing for distance from other UI elements.

If you have success/error variants: snackbar-success-background, snackbar-error-background, etc.

Great — I see exactly what you're after.

You've already defined **Tier 1** (structural & style tokens at the component level). Now for **Tier 2 semantic tokens**, the idea is to move from **component-specific raw values** to

semantic meanings (Success, Error, Primary, Info, Neutral, etc.), so any component can inherit these without redefining them.

This unlocks:

- **Theming** (light/dark/brand)
 - **Consistency** (same Success background across components)
 - **Scalability** (update once, propagate everywhere)
-

📦 Semantic Tier 2 – Cross-Component Tokens

(Shared across *all* components; components just map to these)

🎨 Color Semantics

- **Primary Actions**
 - color.action.primary.background
 - color.action.primary.background.hover
 - color.action.primary.background.active
 - color.action.primary.text
 - color.action.primary.border
- **Secondary Actions**
 - color.action.secondary.background
 - color.action.secondary.background.hover
 - color.action.secondary.text
 - color.action.secondary.border
- **Status Colors**
 - color.status.success.background
 - color.status.success.text
 - color.status.success.border
 - color.status.warning.background
 - color.status.warning.text
 - color.status.warning.border
 - color.status.error.background
 - color.status.error.text
 - color.status.error.border
 - color.status.info.background

- color.status.info.text
 - color.status.info.border
 - **Neutral Surfaces**
 - color.surface.default
 - color.surface.hover
 - color.surface.active
 - color.surface.disabled
 - color.surface.inverse
 - **Text**
 - color.text.primary
 - color.text.secondary
 - color.text.muted
 - color.text.inverse
 - **Borders**
 - color.border.default
 - color.border.subtle
 - color.border.focus
 - color.border.disabled
-

Spacing Semantics

- **Padding**
 - spacing.padding.sm
 - spacing.padding.md
 - spacing.padding.lg
 - **Margin**
 - spacing.margin.sm
 - spacing.margin.md
 - spacing.margin.lg
 - **Gap (between items)**
 - spacing.gap.xs
 - spacing.gap.sm
 - spacing.gap.md
 - spacing.gap.lg
-

Typography Semantics

- **Headings**
 - typography.heading.sm.fontSize
 - typography.heading.md.fontSize
 - typography.heading.lg.fontSize
 - typography.heading.fontWeight
 - **Body**
 - typography.body.sm.fontSize
 - typography.body.md.fontSize
 - typography.body.lg.fontSize
 - typography.body.fontWeight
 - **Labels**
 - typography.label.fontSize
 - typography.label.fontWeight
 - typography.label.color
-

Shape Semantics

- radius.sm
 - radius.md
 - radius.lg
 - radius.round (full circle)
-

Shadow Semantics

- shadow.none
 - shadow.sm
 - shadow.md
 - shadow.lg
-

Mapping Components to Tier 2

Instead of `button-primary-background`, it becomes:

- **Primary Button** → `color.action.primary.background`
- **Success Alert** → `color.status.success.background`
- **Error Snackbar** → `color.status.error.background`

Example:

Component	Maps to Semantic Token
Button (Primary)	color.action.primary.background
Checkbox Checked	color.action.primary.background
Inline Message (Success)	color.status.success.background
Modal Background	color.surface.default
Tooltip Background	color.surface.inverse
Tab Active Border	color.action.primary.border
Progress Bar Fill	color.action.primary.background

 This means:

- Tier 1 = raw values at component level (your existing list)
- Tier 2 = semantic aliases for theme consistency
- Tier 3 = raw primitives (color hex, spacing px, font rem)

 Question — do you want me to go through each of your components one by one (Buttons, Checkboxes, Forms, etc.) and map all possible Tier 2 semantic tokens for them (so we cover every state/variant)? That would make it “all possibilities possible” like you asked. I can start with **Buttons** and build down the list. Would you like me to proceed this way?



lets go more advanced and do all possible options lets do only color for now

Perfect — we'll go **ultra-advanced** and focus **only on Tier 2 semantic color tokens** for now.

Goal: Create a **master semantic color system** that can map to *any* component and *any* state, in a way that supports:

- **Variants** (primary, secondary, tertiary, neutral)
 - **States** (default, hover, active, disabled, focus, pressed)
 - **Status colors** (success, warning, error, info)
 - **Surface levels** (backgrounds, overlays, inverse)
 - **Text layers** (primary, secondary, on-color, muted)
 - **Border colors** (default, focus, subtle)
-

Tier 2 Semantic Color Tokens – Advanced Master Set

1 Action Colors (Primary, Secondary, Tertiary, Neutral)

Primary Action

- `color.action.primary.background.default`
- `color.action.primary.background.hover`
- `color.action.primary.background.active`
- `color.action.primary.background.disabled`
- `color.action.primary.text.default`
- `color.action.primary.text.hover`
- `color.action.primary.text.disabled`
- `color.action.primary.border.default`
- `color.action.primary.border.focus`

Secondary Action

- `color.action.secondary.background.default`
- `color.action.secondary.background.hover`
- `color.action.secondary.background.active`
- `color.action.secondary.background.disabled`
- `color.action.secondary.text.default`
- `color.action.secondary.text.hover`
- `color.action.secondary.text.disabled`
- `color.action.secondary.border.default`
- `color.action.secondary.border.focus`

Tertiary Action

- color.action.tertiary.background.default
- color.action.tertiary.background.hover
- color.action.tertiary.background.active
- color.action.tertiary.text.default
- color.action.tertiary.text.hover
- color.action.tertiary.text.disabled
- color.action.tertiary.border.default
- color.action.tertiary.border.focus

Neutral Action

- color.action.neutral.background.default
 - color.action.neutral.background.hover
 - color.action.neutral.background.active
 - color.action.neutral.background.disabled
 - color.action.neutral.text.default
 - color.action.neutral.text.hover
 - color.action.neutral.text.disabled
 - color.action.neutral.border.default
 - color.action.neutral.border.focus
-

2 Status Colors (Success, Warning, Error, Info)

Success

- color.status.success.background.default
- color.status.success.background.hover
- color.status.success.background.active
- color.status.success.text.default
- color.status.success.text.inverse (text on success background)
- color.status.success.border.default

Warning

- color.status.warning.background.default
- color.status.warning.background.hover
- color.status.warning.background.active
- color.status.warning.text.default
- color.status.warning.text.inverse
- color.status.warning.border.default

Error

- `color.status.error.background.default`
- `color.status.error.background.hover`
- `color.status.error.background.active`
- `color.status.error.text.default`
- `color.status.error.text.inverse`
- `color.status.error.border.default`

Info

- `color.status.info.background.default`
 - `color.status.info.background.hover`
 - `color.status.info.background.active`
 - `color.status.info.text.default`
 - `color.status.info.text.inverse`
 - `color.status.info.border.default`
-

3 Surface & Background Colors

Surfaces

- `color.surface.level.0` – Base background
 - `color.surface.level.1` – Elevated background (e.g., card)
 - `color.surface.level.2` – More elevated (e.g., modal)
 - `color.surface.level.inverse` – Inverse background for dark mode elements
 - `color.surface.overlay.light` – Light overlay (e.g., hover)
 - `color.surface.overlay.dark` – Dark overlay (e.g., modal backdrop)
-

4 Text Colors

Default

- `color.text.primary` – Main readable text
- `color.text.secondary` – Lower emphasis
- `color.text.muted` – Disabled or subtle text
- `color.text.inverse` – Text on dark backgrounds

On Color (Dynamic for Buttons, Alerts, etc.)

- `color.text.onPrimary` – Text on primary backgrounds
- `color.text.onSecondary` – Text on secondary backgrounds

- `color.text.onSuccess` – Text on success backgrounds
 - `color.text.onWarning` – Text on warning backgrounds
 - `color.text.onError` – Text on error backgrounds
 - `color.text.onInfo` – Text on info backgrounds
-

5 Border Colors

- `color.border.default` – Neutral
 - `color.border.subtle` – Low contrast dividers
 - `color.border.focus` – Focus outline
 - `color.border.disabled` – Disabled border
 - `color.border.inverse` – Border on dark backgrounds
-

6 Special Overlays & Highlights

- `color.overlay.backdrop` – Modal/Drawer overlay
 - `color.overlay.hover` – Hover highlight
 - `color.overlay.focus` – Focus highlight glow
 - `color.overlay.selected` – Selected item background highlight
-

💡 Why this works:

Every component (Button, Checkbox, Modal, etc.) just maps its colors to these semantic tokens.

Example:

- `button-primary-background-hover` →
`color.action.primary.background.hover`
 - `inline-message-success-background` →
`color.status.success.background.default`
 - `progress-bar-filled-color` → `color.action.primary.background.default`
-

👉 Next step:

Do you want me to **go through your entire component list** (Buttons → Progress Tracker → Snackbars, etc.) and explicitly **map each component's colors to these semantic**

tokens (so we make sure *all possibilities* are covered)? That's where it gets *really advanced*.



lets do Spacing Semantics

Perfect — now we'll make the **Tier 2 Semantic Spacing Tokens** as complete as possible.

Just like with colors, these will be **semantic** (not tied to a single component), so any component (Button, Card, Modal, List Item, etc.) can map to them.

We'll build it in **three layers**:

1. **Global spacing scale** (base units that everything maps to)
 2. **Component spacing semantics** (padding, gap, margin by intent)
 3. **Layout spacing semantics** (page/grid/container spacing)
-

Tier 2 Semantic Spacing Tokens – Advanced Master Set

1 Global Spacing Scale (Base System)

These are your **foundation tokens** — everything else references these.

- `spacing.0` – 0px (none)
- `spacing.1` – 2px (micro spacing)
- `spacing.2` – 4px (extra small)
- `spacing.3` – 8px (small)
- `spacing.4` – 12px (medium)
- `spacing.5` – 16px (large)
- `spacing.6` – 20px
- `spacing.7` – 24px
- `spacing.8` – 32px
- `spacing.9` – 40px
- `spacing.10` – 48px

- spacing.11 – 64px
- spacing.12 – 80px

(These values are examples; adjust scale to your brand system — e.g., 4px or 8px base grid.)

2 Component Spacing Semantics

These map directly to **intent inside components** — instead of hardcoding padding/margin, components use these.

Padding

- spacing.component.padding.xs – Minimal padding (tags, chips)
- spacing.component.padding.sm – Small padding (compact buttons, inputs)
- spacing.component.padding.md – Medium padding (default buttons, cards)
- spacing.component.padding/lg – Large padding (modals, panels)
- spacing.component.padding.xl – Extra large padding (hero sections, full-width banners)

Gap (Internal spacing between child elements)

- spacing.component.gap.xs – Small gap between icon and text
- spacing.component.gap.sm – Gap between label and input
- spacing.component.gap.md – Gap between stacked elements in a card
- spacing.component.gap/lg – Gap between sections in a modal

Margin (Outer space around component)

- spacing.component.margin.sm – Compact margin (buttons in tight layouts)
 - spacing.component.margin.md – Standard margin (card separation)
 - spacing.component.margin/lg – Large margin (standalone sections)
-

3 Layout & Page Spacing Semantics

These are for **grid, page, and container layouts**.

Page / Section Padding

- spacing.layout.page.padding.sm – Narrow page padding (mobile)
- spacing.layout.page.padding.md – Standard padding (desktop)
- spacing.layout.page.padding/lg – Spacious padding (wide desktop)

Section Spacing

- `spacing.layout.section.spacing.xs` – Tight section spacing (dense UIs)
- `spacing.layout.section.spacing.sm` – Compact section spacing
- `spacing.layout.section.spacing.md` – Default section spacing
- `spacing.layout.section.spacing.lg` – Large section spacing (landing pages)

Grid Gap

- `spacing.layout.grid.gap.sm` – Compact grid gap
 - `spacing.layout.grid.gap.md` – Standard grid gap
 - `spacing.layout.grid.gap.lg` – Wide grid gap
-



How this works:

- A button's padding (`button-padding-x`) → `spacing.component.padding.sm`
 - A modal's section gap → `spacing.component.gap.lg`
 - A card's outer margin → `spacing.component.margin.md`
 - Page layout → `spacing.layout.page.padding.md`
-



Question for you:

Do you want me to go through your component list (like we did for colors) and map each component's padding/margin/gap to these spacing tokens so it's all connected?

That will make the **Tier 2 semantic spacing system 100% ready to use.**



Typography Semantics

Awesome — now we'll build **Tier 2 Semantic Typography Tokens** in the same advanced, all-possibilities style.

The goal is to make a **master typography system** that any component (Buttons, Headers, Modals, Cards, etc.) can map to, while supporting:

- **Type roles** (display, heading, body, label, caption, code)
- **Sizes** (xs → xxxl)

- **Weights** (light → extra bold)
 - **Line heights** (tight → relaxed)
 - **Letter spacing** (tight, normal, loose)
-



Tier 2 Semantic Typography Tokens – Advanced Master Set

1 Typography Roles (Semantic)

These roles are the **semantic layer** — they don't hardcode px/rem, they map to primitives.

Display / Hero (For banners, marketing, hero sections)

- `typography.display.lg` – Largest display text (hero headline)
- `typography.display.md` – Standard display text
- `typography.display.sm` – Small display text

Headings (H1 → H6)

- `typography.heading.xl` – Section title
- `typography.heading.lg` – Page title
- `typography.heading.md` – Subsection title
- `typography.heading.sm` – Card or widget title
- `typography.heading.xs` – Minor heading (labels above inputs, inline headings)

Body

- `typography.body.lg` – Large body text (long reading, paragraphs)
- `typography.body.md` – Default body text
- `typography.body.sm` – Compact body text (dense UIs)
- `typography.body.xs` – Micro text (helper info, tooltips)

Labels & UI Text

- `typography.label.lg` – Large labels (forms, buttons)
- `typography.label.md` – Standard labels (inputs, toggles)
- `typography.label.sm` – Small labels (chips, compact buttons)

Caption / Helper

- `typography.caption.md` – Default caption text (subtext, helper text)

- `typography.caption.sm` – Small caption (secondary helper text)

Code / Monospace

- `typography.code.md` – Inline code (monospace, body size)
 - `typography.code.sm` – Small code (e.g., tags, CLI commands)
-

2 Typography Attributes (Semantic)

These apply to all roles.

Font Families

- `typography.font.family.sans` – Primary sans-serif
- `typography.font.family.serif` – Primary serif (if used)
- `typography.font.family.mono` – Monospace for code
- `typography.font.family.brand` – Brand-specific display font

Font Weights

- `typography.font.weight.light` (300)
- `typography.font.weight.regular` (400)
- `typography.font.weight.medium` (500)
- `typography.font.weight.semibold` (600)
- `typography.font.weight.bold` (700)
- `typography.font.weight.extrabold` (800)

Line Heights

- `typography.lineHeight.tight` – Compact line height (headings, buttons)
- `typography.lineHeight.normal` – Standard line height (body)
- `typography.lineHeight.relaxed` – Comfortable line height (paragraphs, long-form)

Letter Spacing

- `typography.letterSpacing.tight` – Negative letter spacing (large headings)
 - `typography.letterSpacing.normal` – Normal tracking (body text)
 - `typography.letterSpacing.loose` – Wide tracking (labels, buttons, overlines)
-

3 Text Color Semantics (Mapped to Color Tokens)

- `typography.color.primary` – Main text color (maps to `color.text.primary`)

- `typography.color.secondary` – Secondary/lower emphasis (maps to `color.text.secondary`)
 - `typography.color.inverse` – Text on dark backgrounds (maps to `color.text.inverse`)
 - `typography.color.muted` – Disabled/subtle text (maps to `color.text.muted`)
 - `typography.color.link` – Links (maps to `color.action.primary.text`)
 - `typography.color.link.hover` – Hover state for links
-

How this works in practice:

- **Button label** → `typography.label.md + typography.font.weight.medium + typography.color.onPrimary`
 - **Modal title** → `typography.heading.lg + typography.font.weight.semibold`
 - **Tooltip text** → `typography.caption.sm + typography.color.inverse`
 - **Inline error message** → `typography.body.sm + typography.color.status.error`
-

 Question — do you want me to go **component by component** (Buttons → Tabs → Snackbars) and explicitly map **which typography semantic token** each should use (so every component text is consistent)?

That's the step that makes this **Tier 2 typography system fully operational**.



Shape Semantics

Perfect — now we'll define  **Tier 2 Shape Semantics** in the same **all-possibilities** style so every component (Button, Card, Modal, Chip, Icon, etc.) can pull from a consistent shape system.

The goal:

- Support **corner radius scale** (from sharp → pill/circle)
- Define **shape roles** (containers, controls, elements)

- Allow for **variant overrides** (outlined, filled, elevated)
-



Tier 2 Shape Semantics – Advanced Master Set

1 Radius Scale (Global Shape Tokens)

Base radius scale referenced by all components:

- `radius.none` – 0px (sharp corners)
 - `radius.xs` – 2px (micro rounding)
 - `radius.sm` – 4px (slight rounding)
 - `radius.md` – 8px (medium rounding, common for cards, buttons)
 - `radius/lg` – 12px (softer rounding for modals, large panels)
 - `radius.xl` – 16px (large rounded corners for hero sections)
 - `radius.full` – 50% (pill or circle shape)
-

2 Shape Roles (Semantic by Component Type)

Controls (Interactive Elements)

- `shape.control.sm` – Small controls (chips, compact buttons) → maps to `radius.sm`
- `shape.control.md` – Standard controls (default buttons, inputs, toggles) → maps to `radius.md`
- `shape.control/lg` – Large controls (call-to-action buttons, large inputs) → maps to `radius/lg`

Containers (Content Holders)

- `shape.container.sm` – Compact containers (dropdowns, tooltips) → maps to `radius.sm`
- `shape.container.md` – Standard containers (cards, side panels) → maps to `radius.md`
- `shape.container/lg` – Large containers (modals, dialogs) → maps to `radius/lg`

Special Shapes

- `shape.circle` – Perfect circle (icons, avatars) → maps to `radius.full`
 - `shape.pill` – Pill shape (tags, large toggle buttons) → maps to `radius.full`
-

3 Shape Variants (Style-Specific Overrides)

These are used when components have different styling contexts:

- `shape.variant.outlined` – Border-only style with sharp/rounded corners
 - `shape.variant.filled` – Filled background style with consistent radius
 - `shape.variant.elevated` – Elevated container style with slightly more radius
-

How this applies to components:

- **Button** → `shape.control.md` (default), `shape.pill` (rounded style)
 - **Chip** → `shape.control.sm` or `shape.pill` (if rounded)
 - **Card** → `shape.container.md` (default), `shape.container/lg` (hero section card)
 - **Modal** → `shape.container.lg`
 - **Avatar/Icon** → `shape.circle`
-

 Question — do you want me to go **component by component** and map the exact `shape` token each should use (like we did with colors, typography, spacing)?

That will make the **Shape Tier 2 mapping fully complete** and ready for all components.



Shadow Semantics

Perfect — let's now define the full  **Tier 2 Shadow Semantics**, structured for high clarity, scalability, and use across any component in your system.

The goal is to:

- Support **elevation levels** (surface, modal, tooltip, hover, etc.)
 - Maintain **visual consistency** across light and dark themes
 - Allow reuse across **cards, tooltips, dropdowns, modals, buttons**, and more
-

Tier 2 Shadow Semantics – Advanced Master Set

1 Global Shadow Scale

These tokens define the **base elevation levels** that can be reused across components.

- shadow.none – No shadow
 - shadow.xs – Very light (flat UI elements, hover overlays)
 - shadow.sm – Small depth (cards, dropdowns, popovers)
 - shadow.md – Medium depth (modals, active containers)
 - shadow.lg – Large depth (dialogs, overlays)
 - shadow.xl – Extra large (for full-screen overlays or special effects)
 - shadow.inset – Inner shadow (for pressed/focus states)
-

2 Semantic Shadow Roles by Purpose

Containers & Surfaces

- shadow.surface.default – Default surface (card, panel, section)
- shadow.surface.elevated – Slightly raised surface (hover card, side panel)
- shadow.surface.modal – Modals/dialogs
- shadow.surface.overlay – Heavy shadow for blocking overlays (drawers, full-page modals)

Interactions

- shadow.interaction.hover – Shadow on hover for buttons, tiles
- shadow.interaction.focus – Optional visual cue when focused
- shadow.interaction.active – Pressed/active state visual depth

Decorative

- shadow.decorative.subtle – Soft visual enhancement (tooltip, tooltip)
 - shadow.decorative.glow – Highlighted or emphasized UI (e.g., status badges)
 - shadow.decorative.borderless – Used instead of borders (with light contrast)
-

3 Shadow Modifiers for Themes

Support for **theme adaptation** (dark/light modes):

- shadow.light.sm
- shadow.dark.sm
- shadow.light.lg
- shadow.dark.lg

(these can override the neutral ones if your themes require tonal differences in

shadows)

💡 How components might use these:

Component	Shadow Token
Button (hover)	shadow.interaction.hover
Card	shadow.surface.default
Modal	shadow.surface.modal
Tooltip	shadow.decorative.subtle
Toast	shadow.surface.elevated
Dropdown	shadow.surface.elevated
Pressed state	shadow.interaction.active
Focused field	shadow.interaction.focus

Example Strategy:

Use tokens like:

scss

 Copy code

```
.card {
  box-shadow: var(--shadow-surface-default);
}

.button:hover {
  box-shadow: var(--shadow-interaction-hover);
}
```

👉 Next Step:

Would you like me to go through each of your components and map which shadow

token applies to each (as we did for other semantic layers)? That will finish the Tier 2 system for **visual depth across your full UI**.



border

Perfect — now let's build the **Tier 2 Border Semantics** in the same **complete, scalable style** so all components (Buttons, Inputs, Cards, Modals, Chips, Tables, etc.) can share consistent border logic.

This will cover:

- **Border widths** (hairline → thick)
 - **Border radii** (already handled in Shape, but referenced here for consistency)
 - **Border colors** (default, focus, status, inverse)
 - **Border styles** (solid, dashed, dotted — for specific cases)
-

Tier 2 Border Semantics – Advanced Master Set

1 Border Width Scale (Global)

These are **foundation widths** that everything maps to:

- `border.width.none` – 0px
 - `border.width.hairline` – 1px (subtle dividers)
 - `border.width.sm` – 1.5px (standard component borders)
 - `border.width.md` – 2px (active/focus emphasis)
 - `border.width/lg` – 3px+ (alert/high emphasis borders)
-

2 Border Styles

These styles can be applied to any width:

- `border.style.solid` – Default
- `border.style.dashed` – Segmented (e.g., upload drop zone)

- `border.style.dotted` – Rarely used, decorative or helper state
-

3 Semantic Border Colors

Default / Neutral

- `border.color.default` – Standard component borders
- `border.color.subtle` – Dividers, low contrast
- `border.color.disabled` – Disabled state borders

Focus / Interaction

- `border.color.focus` – Active input focus
- `border.color.hover` – Hover emphasis border

Status Colors

- `border.color.success` – Success state
- `border.color.warning` – Warning state
- `border.color.error` – Error state
- `border.color.info` – Info state

Inverse

- `border.color.inverse` – Borders on dark backgrounds
-

4 Component-Level Semantic Application

These semantic borders are mapped to component categories:

- **Controls (Inputs, Buttons, Toggles)**
 - `border.control.default` → `border.width.sm + border.color.default`
 - `border.control.focus` → `border.width.md + border.color.focus`
 - `border.control.error` → `border.width.sm + border.color.error`
- **Containers (Cards, Modals, Dropdowns)**
 - `border.container.default` → `border.width.hairline + border.color.subtle`
 - `border.container.elevated` → `border.width.hairline + border.color.default`
- **Tables & Lists**
 - `border.table.cell` → `border.width.hairline + border.color.subtle`
 - `border.list.item` → `border.width.hairline + border.color.subtle`

💡 How this applies:

- Button border → `border.control.default`
 - Input active border → `border.control.focus`
 - Card outline → `border.container.default`
 - Inline message error → `border.control.error`
-

👉 Question:

Do you want me to map each component in your list (**Buttons** → **Snackbar**) to the correct **border semantic tokens** just like we did for Colors, Spacing, Typography, Shape, Shadow? That will complete **Tier 2 Borders** in a fully actionable way.



I am missing any semeantics?

You're building a **really robust Tier 2 system** — and you've already nailed most of the big semantic layers:

- ✓ **Colors** (action, status, surface, text, border, overlays)
- ✓ **Spacing** (scale, component, layout)
- ✓ **Typography** (roles, weights, sizes, line-height, letter-spacing)
- ✓ **Shape** (radius scale, shape roles, variants)
- ✓ **Shadow** (scale, surface, interaction)
- ✓ **Border** (width, style, color roles)

That's a strong semantic core.

But to make it **truly complete and “all possibilities possible”** for your components, you might want to add:

📌 Additional Semantic Layers You Might Be Missing

1 Motion Semantics (*Often missed, but critical for consistent animations*)

- **Durations**
 - `motion.duration.xs` (fast micro-interactions)

- `motion.duration.sm` (hover transitions)
 - `motion.duration.md` (modals, dropdowns)
 - `motion.duration/lg` (page transitions)
- **Easing**
 - `motion.easing.standard` (default smooth easing)
 - `motion.easing.decelerate` (entrance animations)
 - `motion.easing.accelerate` (exit animations)
 - `motion.easing.bounce` (playful interactions)
-

2 Opacity / Transparency Semantics

(Super useful for disabled states, overlays, subtle effects)

- `opacity.disabled` – Components in disabled state
 - `opacity.hover` – Hover highlights
 - `opacity.focus` – Focus/active states
 - `opacity.overlay.light` – Light modal background
 - `opacity.overlay.dark` – Dark modal background
-

3 Z-Index / Elevation Semantics

(So modals, dropdowns, tooltips don't clash)

- `zindex.base` – Default elements
 - `zindex.dropdown` – Above base layer
 - `zindex.tooltip` – Above dropdowns
 - `zindex.modal` – Above everything
 - `zindex.toast` – Persistent overlays
-

4 Blur / Backdrop Semantics

(For frosted glass, modals, navigation bars)

- `backdrop.blur.none` – No blur
 - `backdrop.blur.sm` – Light blur
 - `backdrop.blur.md` – Medium blur
 - `backdrop.blur.lg` – Heavy blur
-

5 Scale / Transform Semantics

(Common for hover, pressed, pop effects)

- `transform.scale.hover` – Slight grow on hover
 - `transform.scale.pressed` – Slight shrink on click
 - `transform.rotate.icon` – Icon rotation for toggles/accordions
-

💡 With these added, you'd have a **complete Tier 2 semantic set**:

1. Color ✓
 2. Spacing ✓
 3. Typography ✓
 4. Shape ✓
 5. Shadow ✓
 6. Border ✓
 7. Motion +
 8. Opacity +
 9. Z-Index +
 10. Backdrop / Blur +
 11. Transform / Scale +
-

👉 Question:

Do you want me to **expand your Tier 2 system to include these missing semantics** (motion, opacity, z-index, blur, transforms) so it's *absolutely complete* before we map everything to your components? This will make your system bulletproof.



lets review colors and check if is all there in term os possibilites to have so many diffrente components on the system

Great call — before locking everything in, let's stress-test the **Tier 2 semantic colors** against all the different components you listed (buttons, checkboxes, modals, alerts, progress bars, etc.) and make sure *nothing is missing*.

We want to cover **all possible states, surfaces, roles, and variants** in a way that is flexible but not bloated.

Tier 2 Semantic Colors – Full Review for All Components

1 Action Colors (Interactive Elements)

These map to buttons, inputs, toggles, sliders, tabs, etc.

Primary Action

- color.action.primary.background.default
- color.action.primary.background.hover
- color.action.primary.background.active
- color.action.primary.background.disabled
- color.action.primary.text.default
- color.action.primary.text.disabled
- color.action.primary.border.default
- color.action.primary.border.focus

Secondary Action

- color.action.secondary.background.default
- color.action.secondary.background.hover
- color.action.secondary.background.active
- color.action.secondary.background.disabled
- color.action.secondary.text.default
- color.action.secondary.text.disabled
- color.action.secondary.border.default
- color.action.secondary.border.focus

Tertiary / Neutral Actions

- color.action.neutral.background.default
- color.action.neutral.background.hover
- color.action.neutral.background.active
- color.action.neutral.background.disabled
- color.action.neutral.text.default
- color.action.neutral.text.disabled

- `color.action.neutral.border.default`
-

2 Status Colors (Feedback & System States)

Used by inline messages, toasts, badges, progress bars, charts, alerts.

Success

- `color.status.success.background.default`
- `color.status.success.background.hover`
- `color.status.success.text.default`
- `color.status.success.text.inverse`
- `color.status.success.border.default`

Warning

- `color.status.warning.background.default`
- `color.status.warning.background.hover`
- `color.status.warning.text.default`
- `color.status.warning.text.inverse`
- `color.status.warning.border.default`

Error

- `color.status.error.background.default`
- `color.status.error.background.hover`
- `color.status.error.text.default`
- `color.status.error.text.inverse`
- `color.status.error.border.default`

Info

- `color.status.info.background.default`
 - `color.status.info.background.hover`
 - `color.status.info.text.default`
 - `color.status.info.text.inverse`
 - `color.status.info.border.default`
-

3 Surface & Background Colors

Used by modals, dropdowns, cards, tooltips, tables, page backgrounds.

- `color.surface.level.0` – Base background

- `color.surface.level.1` – Elevated surface (cards, panels)
 - `color.surface.level.2` – Higher elevation (modals, dropdowns)
 - `color.surface.level.3` – Highest elevation (toasts, floating elements)
 - `color.surface.inverse` – Inverse background (dark mode surfaces)
 - `color.surface.overlay.light` – Light overlay
 - `color.surface.overlay.dark` – Dark overlay (modal scrim)
-

4 Text Colors

Used everywhere: buttons, forms, lists, headers, inline messages.

Default Text

- `color.text.primary` – Main text color
- `color.text.secondary` – Lower emphasis text
- `color.text.muted` – Disabled text
- `color.text.inverse` – Text on dark backgrounds

On-Color Text

- `color.text.onPrimary` – Text on primary background
 - `color.text.onSecondary` – Text on secondary background
 - `color.text.onSuccess` – Text on success background
 - `color.text.onWarning` – Text on warning background
 - `color.text.onError` – Text on error background
 - `color.text.onInfo` – Text on info background
-

5 Border Colors

Used in inputs, tables, dividers, cards, modals.

- `color.border.default` – Standard border
 - `color.border.subtle` – Low contrast (dividers)
 - `color.border.focus` – Focus outline
 - `color.border.disabled` – Disabled border
 - `color.border.inverse` – Border on dark background
-

6 Special Cases & Overlays

For special components: tooltips, highlights, selections, loaders.

- `color.overlay.backdrop` – Modal/drawer scrim
 - `color.overlay.hover` – Hover layer for selectable items
 - `color.overlay.focus` – Focus highlight
 - `color.overlay.selected` – Selection highlight (list, table row, tab)
 - `color.highlight.mention` – Highlight for mentions/tags in text
 - `color.highlight.search` – Search result highlight
-

Cross-check against your component list:

- Buttons / Payment nav / Action btns →  Action colors cover all
 - Checkboxes / Radios / Toggles →  Action + border focus colors cover states
 - Forms / Inputs / Dropdowns →  Surface + border + text colors cover them
 - Modals / Cards / Lists / Tables →  Surface levels, border, text colors included
 - Alerts / Inline message / Snackbar / Toasts / Badges →  Status colors covered
 - Progress bar / Tracker / Sliders →  Status + action colors covered
 - Tabs / Chips / Kebab / Segments →  Action + surface overlay colors covered
 - Tooltip / Toggletip →  Surface inverse + text inverse covered
 - Empty state →  Surface + status colors for emphasis
 - Pagination →  Action + text colors covered
-

Verdict:

You are **99% covered** in terms of possibilities. The **only potential additions** I see (optional but nice for completeness):

- **Color for disabled surfaces:** `color.surface.disabled` (for disabled cards, panels, or inputs)
 - **Color for pressed/active overlays:** `color.overlay.active` (for pressed tabs, list items, buttons)
 - **Brand accent colors** (if you use tertiary brand tones): `color.brand.accent1`, `color.brand.accent2`
-

Question for you:

Do you want me to add these 3 missing semantic colors (disabled surface, active

overlay, brand accents) so we have *literally every possible case covered* before we lock the Tier 2 color system?



lets cover all primitives toeks tier 1 possible

Perfect — now we're switching to **Tier 1: Primitives** (the absolute foundation layer).

Remember:

- **Tier 1 (Primitives)** = raw design values (hex codes, px, rem, ms, %, etc.)
- **Tier 2 (Semantics)** = meaning/context for those values (e.g.,
`color.action.primary.background`)
- **Tier 3 (Component tokens)** = applied to actual components (e.g., `button-primary-background`)

We want **every primitive possible** so Tier 2 can draw from them without gaps.

Tier 1 – Complete Primitive Tokens

Color Primitives

(These are raw color values. Tier 2 semantic colors will reference these.)

Neutral Palette

- `color.neutral.0` – White
- `color.neutral.50`
- `color.neutral.100`
- `color.neutral.200`
- `color.neutral.300`
- `color.neutral.400`
- `color.neutral.500`
- `color.neutral.600`
- `color.neutral.700`
- `color.neutral.800`

- `color.neutral.900` – Black

Brand Palette

- `color.brand.primary.50` → .900
- `color.brand.secondary.50` → .900
- `color.brand.tertiary.50` → .900 (*optional if you have accent colors*)

Status Colors

- `color.success.50` → .900
- `color.warning.50` → .900
- `color.error.50` → .900
- `color.info.50` → .900

Extended Palette (Optional)

- `color.accent.1.50` → .900
 - `color.accent.2.50` → .900
 - `color.accent.3.50` → .900
-

Spacing Primitives

(Raw spacing scale, base 4px or 8px system recommended)

- `spacing.0` – 0px
 - `spacing.1` – 2px
 - `spacing.2` – 4px
 - `spacing.3` – 8px
 - `spacing.4` – 12px
 - `spacing.5` – 16px
 - `spacing.6` – 20px
 - `spacing.7` – 24px
 - `spacing.8` – 32px
 - `spacing.9` – 40px
 - `spacing.10` – 48px
 - `spacing.11` – 64px
 - `spacing.12` – 80px
 - `spacing.13` – 96px
 - `spacing.14` – 112px
-



Typography Primitives

(Raw values for font family, size, weight, line height, letter spacing)

Font Families

- `font.family.sans` – Primary sans-serif
- `font.family.serif` – Primary serif
- `font.family.mono` – Monospace
- `font.family.brand` – Brand display font

Font Sizes

- `font.size.50` – 10px
- `font.size.75` – 12px
- `font.size.100` – 14px
- `font.size.200` – 16px
- `font.size.300` – 18px
- `font.size.400` – 20px
- `font.size.500` – 24px
- `font.size.600` – 28px
- `font.size.700` – 32px
- `font.size.800` – 40px
- `font.size.900` – 48px+

Font Weights

- `font.weight.100` – Thin
- `font.weight.200` – Extra Light
- `font.weight.300` – Light
- `font.weight.400` – Regular
- `font.weight.500` – Medium
- `font.weight.600` – Semi Bold
- `font.weight.700` – Bold
- `font.weight.800` – Extra Bold
- `font.weight.900` – Black

Line Heights

- `line.height.tight` – ~1.1
- `line.height.normal` – ~1.4
- `line.height.relaxed` – ~1.6

Letter Spacing

- `letter.spacing.tight` – `-0.5px`
 - `letter.spacing.normal` – `0px`
 - `letter.spacing.loose` – `0.5px`
-

Shape / Radius Primitives

- `radius.none` – `0px`
 - `radius.xs` – `2px`
 - `radius.sm` – `4px`
 - `radius.md` – `8px`
 - `radius.lg` – `12px`
 - `radius.xl` – `16px`
 - `radius.full` – `50%`
-

Shadow Primitives

(Each references actual box-shadow values in CSS)

- `shadow.none` – `none`
 - `shadow.xs` – tiny shadow (hover highlights)
 - `shadow.sm` – card/dropdown
 - `shadow.md` – modal
 - `shadow.lg` – heavy modal/dialog
 - `shadow.xl` – full overlay
 - `shadow.inset` – inset shadows
-

Border Primitives

- `border.width.none` – `0px`
 - `border.width.hairline` – `1px`
 - `border.width.sm` – `1.5px`
 - `border.width.md` – `2px`
 - `border.width.lg` – `3px+`
 - `border.style.solid`
 - `border.style.dashed`
 - `border.style.dotted`
-

Motion Primitives

- Durations

- `motion.duration.xs` – 75ms
- `motion.duration.sm` – 150ms
- `motion.duration.md` – 250ms
- `motion.duration/lg` – 400ms
- `motion.duration.xl` – 600ms

- Easing

- `motion.easing.standard` – cubic-bezier(0.4, 0, 0.2, 1)
 - `motion.easing.decelerate` – cubic-bezier(0, 0, 0.2, 1)
 - `motion.easing.accelerate` – cubic-bezier(0.4, 0, 1, 1)
-

Opacity Primitives

- `opacity.0` – 0%
 - `opacity.25` – 25%
 - `opacity.50` – 50%
 - `opacity.75` – 75%
 - `opacity.100` – 100%
-

Z-Index Primitives

- `zindex.0` – Base layer
 - `zindex.10` – Dropdowns
 - `zindex.20` – Tooltips
 - `zindex.30` – Modals
 - `zindex.40` – Toasts
-

Backdrop / Blur Primitives

- `backdrop.blur.none` – 0px
 - `backdrop.blur.sm` – 2px
 - `backdrop.blur.md` – 4px
 - `backdrop.blur.lg` – 8px
-

✓ This covers every primitive possible so that Tier 2 semantic tokens have no gaps.

👉 Question for you:

Do you want me to **create a clean, organized Tier 1 → Tier 2 mapping chart** (so every primitive clearly links to its semantic token), or keep them separate? This will make it *much easier to maintain* as your system grows.



miss anything?

You've got an **extremely solid Tier 1 primitive set** right now — honestly, most teams never get this complete.

But since you're building for **many components + future scaling**, I reviewed all areas again and here's my check:

✓ What you already covered

- **Colors** → full neutral + brand + status + accents
- **Spacing** → full scale (0 → 112px)
- **Typography** → families, sizes, weights, line heights, letter spacing
- **Shape** → full radius scale
- **Shadow** → full elevation set + inset
- **Border** → widths + styles
- **Motion** → durations + easing
- **Opacity** → 0 → 100%
- **Z-Index** → base layers
- **Blur** → backdrop levels

⚠ Potential missing primitives (optional but valuable for big systems)

1 Grid / Layout primitives

(Helps for consistent responsive layouts)

- `grid.columns.count` (default column count, e.g. 12)
 - `grid.gutter.sm / md / lg` (space between columns)
 - `grid.maxWidth.sm / md / lg` (container widths for breakpoints)
-

2 Breakpoints / Media queries

(Critical for responsive theming)

- `breakpoint.xs` – Mobile small
 - `breakpoint.sm` – Mobile large
 - `breakpoint.md` – Tablet
 - `breakpoint/lg` – Desktop
 - `breakpoint.xl` – Large desktop
-

3 Icon sizing primitives

(Avoids inconsistent icon sizing across system)

- `icon.size.xs / sm / md / lg / xl` (raw px values)
-

4 Transition / Animation primitives

(Motion covers timing/easing, but you may want property groups)

- `motion.property.color` – Used for hover/focus states
 - `motion.property.transform` – Scale, rotate, translate
 - `motion.property.opacity` – Fade animations
-

5 Timing / Delay primitives

(Useful for staggered animations, loaders, progress indicators)

- `motion.delay.xs / sm / md / lg`
-

6 Elevation color primitives

(Shadows already covered, but elevation often pairs with background tone)

- `elevation.color.level1 / level2 / level3` (to ensure surfaces at different heights have the right tone)

7 Focus ring primitives

(Useful for accessibility & consistent keyboard focus)

- `focus.ring.width`
- `focus.ring.color`
- `focus.ring.offset`

If you add **grid**, **breakpoints**, **icon sizes**, **transition properties**, **delays**, **elevation colors**, and **focus ring**, you've essentially covered **100% of all common Tier1 primitives** in large, scalable design systems.

👉 Question for you:

Do you want me to **add these missing sets** (grid, breakpoints, icon sizing, transition properties, delay, elevation colors, focus ring) into your Tier1 so you have a **bulletproof, future-proof primitives layer** before we lock it? This will make it enterprise-level complete.



we are covering margins paddings?

Great catch — we've defined **Spacing primitives** (the raw scale), but we haven't explicitly broken out **Margin** and **Padding primitives** as named Tier1 tokens.

👉 Right now, you can technically use `spacing.x` for both margin and padding, but for **large systems** it's often worth explicitly defining **padding/margin primitives** to:

- Keep consistency across **containers, components, and layouts**
- Prevent random custom spacing values sneaking in
- Allow future overrides (e.g., compact vs. comfortable density modes)

▀ Tier1 – Margin & Padding Primitives (Advanced)

1 Padding Primitives

(For internal space inside components & containers)

- padding.none – 0px
- padding.xs – 2px
- padding.sm – 4px
- padding.md – 8px
- padding.lg – 12px
- padding.xl – 16px
- padding.2xl – 24px
- padding.3xl – 32px

(Can map to component tokens like button-padding-x , card-padding-y in Tier 3)

2 Margin Primitives

(For external space around components & layouts)

- margin.none – 0px
 - margin.xs – 2px
 - margin.sm – 4px
 - margin.md – 8px
 - margin.lg – 12px
 - margin.xl – 16px
 - margin.2xl – 24px
 - margin.3xl – 32px
-

3 Gap Primitives

(For spacing between child elements inside a flex/grid container)

- gap.none – 0px
- gap.xs – 2px
- gap.sm – 4px
- gap.md – 8px
- gap.lg – 12px
- gap.xl – 16px
- gap.2xl – 24px

- `gap.3xl - 32px`
-

💡 Why it's worth having explicit margin/padding tokens:

- **Responsive density modes** → e.g., compact mode could map `padding.md` to 6px instead of 8px
 - **Cross-component consistency** → same token applies to buttons, cards, modals, etc.
 - **Cleaner theming** → You can adjust spacing globally by just editing the primitive
-

👉 Do you want me to also **separate margin/padding into horizontal (padding.x) and vertical (padding.y) primitives** in Tier 1?

This is common in advanced systems so components can keep **consistent vertical rhythm vs horizontal breathing space** (especially for buttons, modals, form fields).



instead a short size standas is not better 100 200 300 etc

100 Yes — for **enterprise-level token systems** (especially when you want scale, density modes, and theme variations), using a **numeric scale (100, 200, 300...)** instead of `xs/sm/md/lg` is the **better long-term choice**.

Why numeric spacing scale is better

- **Scales cleanly** → no awkward jump from `lg` to `xl`, just `400 → 500 → 600`
 - **Supports density modes** → compact mode can map `spacing.300` to a smaller px value without breaking naming
 - **Avoids naming conflicts** → `sm` might mean different things for buttons vs containers, but `spacing.300` is always the same value
 - **Industry standard** → Material, Atlassian, Salesforce Lightning all use numeric step scales
-

💡 Tier1 – Numeric Spacing Primitives

Base spacing unit example: 4px — adjust to your system grid

Spacing (can be used for margin, padding, gap)

- spacing.0 – 0px
 - spacing.100 – 4px
 - spacing.200 – 8px
 - spacing.300 – 12px
 - spacing.400 – 16px
 - spacing.500 – 20px
 - spacing.600 – 24px
 - spacing.700 – 32px
 - spacing.800 – 40px
 - spacing.900 – 48px
 - spacing.1000 – 64px
 - spacing.1100 – 80px
 - spacing.1200 – 96px
 - spacing.1300 – 112px
-

👉 Then in Tier2 we map:

- padding.component.sm → spacing.200
 - padding.component.md → spacing.300
 - margin.container.default → spacing.400
 - gap.layout.grid → spacing.500
-

🔥 This gives you **one single scale** for:

- Margin
 - Padding
 - Gap
 - Component spacing
 - Layout spacing
-

Question for you:

⚡ Do you want me to convert all your margin/padding/gap tokens into a unified numeric spacing.xxx system so Tier 2 can reference the same values without duplicating names? That will make it *super clean and future-proof*.



so lets make our padding margin tier 2

Perfect — now we'll build **Tier2: Semantic Padding & Margin Tokens** using the numeric **Tier1 spacing scale (100, 200, 300...)**.

💡 *Principle:* Tier 2 doesn't care about exact pixels — it gives **meaning** to where padding/margin is applied (component type, layout type, container type).

✍️ Tier 2 – Padding & Margin Semantics

1 Component Padding

(Internal space inside components)

- Buttons
 - padding.button.sm.x → spacing.200
 - padding.button.sm.y → spacing.100
 - padding.button.md.x → spacing.300
 - padding.button.md.y → spacing.200
 - padding.button.lg.x → spacing.400
 - padding.button.lg.y → spacing.300
- Inputs / Text Fields
 - padding.input.x → spacing.300
 - padding.input.y → spacing.200
- Chips / Tags
 - padding.chip.x → spacing.200
 - padding.chip.y → spacing.100
- Cards
 - padding.card.sm → spacing.300

- padding.card.md → spacing.400
 - padding.card.lg → spacing.500
 - **Modals / Panels**
 - padding.modal.content → spacing.500
 - padding.modal.header → spacing.400
 - padding.modal.footer → spacing.400
-

2 Component Margin

(Outer spacing between components)

- **Buttons**
 - margin.button.group.gap → spacing.200
 - **Form Fields**
 - margin.input.group.gap → spacing.300
 - **Cards / Blocks**
 - margin.card.stack → spacing.400
 - **Sections / Panels**
 - margin.section.stack → spacing.500
-

3 Layout Padding

(Page/container padding)

- padding.layout.page.x.sm → spacing.400
 - padding.layout.page.x.md → spacing.500
 - padding.layout.page.x.lg → spacing.600
 - padding.layout.page.y.sm → spacing.300
 - padding.layout.page.y.md → spacing.400
 - padding.layout.page.y.lg → spacing.500
-

4 Layout Margin

(Outer space for large structures)

- margin.layout.section.y.sm → spacing.400
- margin.layout.section.y.md → spacing.500
- margin.layout.section.y.lg → spacing.600
- margin.layout.grid.gap.sm → spacing.300

- `margin.layout.grid.gap.md` → `spacing.400`
 - `margin.layout.grid.gap.lg` → `spacing.500`
-

💡 How this works:

- A **button** doesn't hardcode 8px padding → it uses `padding.button.md.x` → `spacing.300`
 - A **modal** doesn't hardcode 24px padding → it uses `padding.modal.content` → `spacing.500`
 - A **card stack** uses `margin.card.stack` → `spacing.400`
-

👉 Question:

Do you want me to now **go through your full component list** (Buttons → Tabs → Snackbar → Progress Tracker, etc.) and map **each one's padding & margin tokens** to this Tier 2 numeric scale so it's complete? That will make your padding/margin system bulletproof.



Here is a comprehensive set of token names for buttons that you can adapt to your design system. They follow the pattern category-component-role-state (or a subset of those fields) and can be referenced back to semantic tokens like color-primary, spacing-4, etc. Define only the variants and states you actually use.

Structural tokens (apply to all buttons)
`button-border-radius`

`button-padding-x`

`button-padding-y`

`button-font-size`

`button-line-height`

`button-font-weight`

button-min-width

button-min-height

button-icon-size

button-icon-spacing – space between icon and label

button-shadow – default elevation

button-focus-ring-color

button-focus-ring-width

button-focus-ring-offset

Generic state tokens

Use these for behaviours common to all variants:

button-disabled-opacity

button-cursor-disabled (e.g. default vs. not-allowed)

button-transition-duration

Primary button tokens

Colors

button-primary-background

button-primary-background-hover

button-primary-background-active

button-primary-background-disabled

button-primary-text-color

button-primary-text-color-hover

button-primary-text-color-active

button-primary-text-color-disabled

button-primary-border-color

button-primary-border-color-hover

button-primary-border-color-active

button-primary-border-color-disabled

button-primary-icon-color

Shadows

button-primary-shadow

button-primary-shadow-hover

button-primary-shadow-active

Focus

button-primary-focus-ring-color

button-primary-focus-ring-width

button-primary-focus-ring-offset

Secondary button tokens

Colors

button-secondary-background

button-secondary-background-hover

button-secondary-background-active

button-secondary-background-disabled

button-secondary-text-color

button-secondary-text-color-hover

button-secondary-text-color-active

button-secondary-text-color-disabled

button-secondary-border-color

button-secondary-border-color-hover

button-secondary-border-color-active

button-secondary-border-color-disabled

button-secondary-icon-color

Shadows & focus

button-secondary-shadow, button-secondary-shadow-hover, button-secondary-shadow-active

button-secondary-focus-ring-color, button-secondary-focus-ring-width, button-secondary-focus-ring-offset

Tertiary / ghost / link button tokens

For ghost or text-only buttons (transparent backgrounds):

button-tertiary-text-color, button-tertiary-text-color-hover, button-tertiary-text-color-active, button-tertiary-text-color-disabled

button-tertiary-background (usually transparent), button-tertiary-background-hover, button-tertiary-background-active

button-tertiary-border-color, button-tertiary-border-color-hover, button-tertiary-border-color-active

button-tertiary-icon-color

button-tertiary-focus-ring-color, button-tertiary-focus-ring-width, button-tertiary-focus-ring-offset

Destructive button tokens

For actions with potentially negative consequences:

button-destructive-background, button-destructive-background-hover, button-destructive-background-active, button-destructive-background-disabled

button-destructive-text-color, button-destructive-text-color-hover, button-destructive-text-color-active, button-destructive-text-color-disabled

button-destructive-border-color, button-destructive-border-color-hover, button-destructive-border-color-active, button-destructive-border-color-disabled

button-destructive-icon-color

button-destructive-shadow, button-destructive-focus-ring-color, etc.

Outline button tokens

If you support outlined variants:

button-outline-background (usually transparent), button-outline-background-hover, button-outline-background-active

button-outline-border-color, button-outline-border-color-hover, button-outline-border-color-active, button-outline-border-color-disabled

button-outline-text-color, button-outline-text-color-hover, button-outline-text-color-active, button-outline-text-color-disabled

button-outline-icon-color

Icon-only button tokens

For buttons that contain only an icon:

button-icon-only-size

button-icon-only-background, button-icon-only-background-hover, button-icon-only-background-active

button-icon-only-icon-color, button-icon-only-icon-color-hover, button-icon-only-icon-color-active

button-icon-only-border-color, button-icon-only-border-radius

button-icon-only-focus-ring-color, button-icon-only-focus-ring-width

Special variants (payment navigation, action)

If your design system distinguishes these:

Payment navigation:

button-payment-nav-background, button-payment-nav-text-color, button-payment-nav-icon-color, plus hover/active/disabled versions as above.

Action:

button-action-background, button-action-text-color, button-action-icon-color, with hover/active/disabled and focus tokens.

i got this setting for buttons but is not saying for different size of buttons correct?

Exactly ✅ — your current set is **comprehensive for states and variants**, but it doesn't explicitly handle **button size variations** (small, medium, large, extra large, etc.).

Right now, you have:

- **Structural tokens** → button-padding-x , button-padding-y , button-font-size , button-min-height
...but these are defined **only once**, so they apply to the default size.

If you want **multiple sizes** (e.g., small buttons with tighter padding, large buttons with bigger text), you'll need a **size semantic layer**.

Adding Button Size Tokens (Tier 2)

This is common in advanced systems: Material, Carbon, and Lightning all do this.

1 Size Semantic Tokens

Instead of redefining per variant, you define **size sets**:

Small Button

- button.size.sm.padding.x → spacing.200
- button.size.sm.padding.y → spacing.100
- button.size.sm.font.size → font.size.100
- button.size.sm.font.weight → font.weight.medium
- button.size.sm.height → 32px
- button.size.sm.icon.size → 16px
- button.size.sm.icon.spacing → spacing.100

Medium Button (Default)

- button.size.md.padding.x → spacing.300
- button.size.md.padding.y → spacing.200
- button.size.md.font.size → font.size.200
- button.size.md.height → 40px
- button.size.md.icon.size → 20px
- button.size.md.icon.spacing → spacing.200

Large Button

- button.size.lg.padding.x → spacing.400
- button.size.lg.padding.y → spacing.300
- button.size.lg.font.size → font.size.300
- button.size.lg.height → 48px
- button.size.lg.icon.size → 24px
- button.size.lg.icon.spacing → spacing.200

2 Why This Matters

With these tokens:

- All **variants** (primary, secondary, outline, destructive) can inherit from the same size tokens.

- You avoid redefining padding and font sizes for every variant.
 - You can create **density modes** (compact → smaller size tokens) without touching each button variant.
-

Example Mapping

- `button-primary-sm` → uses `button.size.sm` for padding, height, font size
 - `button-secondary-lg` → uses `button.size.lg` for padding, height, font size
-

 Question for you:

Do you want me to **expand your button token set** to explicitly include **size tokens** for all variants (Primary, Secondary, Destructive, Outline, Icon-Only, Payment, Action) so each can have `sm`, `md`, `lg` without duplication? This will make the button system future-proof.



what I just sent you is tier 3 of butotns components.,
updated them with sizes, not miss anything

Perfect — I get what you're saying now.

 What you sent is **Tier 3 (Component level)** for Buttons:

- **Structural tokens** (padding, radius, font size, etc.)
- **State tokens** (hover, active, disabled, focus)
- **Variant tokens** (primary, secondary, destructive, outline, icon-only, payment, action)

 What's missing: **Size dimensioning** (`sm`, `md`, `lg`) applied across those variants.

Let's **update your Tier 3 Buttons** so:

- Sizes are fully integrated (`sm`, `md`, `lg`)
- All variants can inherit the correct padding, font size, icon size
- No gaps for states or special variants

Updated Tier 3 – Buttons (with Sizes Integrated)

Structural Tokens (Apply to All Buttons)

- button-border-radius
 - button-min-width
 - button-min-height
 - button-shadow
 - button-focus-ring-color
 - button-focus-ring-width
 - button-focus-ring-offset
 - button-disabled-opacity
 - button-cursor-disabled
 - button-transition-duration
-

Size Tokens (Apply Across All Variants)

(New – integrate into all variants instead of duplicating values per variant)

Small

- button-size-sm-padding-x
- button-size-sm-padding-y
- button-size-sm-font-size
- button-size-sm-line-height
- button-size-sm-font-weight
- button-size-sm-icon-size
- button-size-sm-icon-spacing

Medium (Default)

- button-size-md-padding-x
- button-size-md-padding-y
- button-size-md-font-size
- button-size-md-line-height
- button-size-md-font-weight
- button-size-md-icon-size
- button-size-md-icon-spacing

Large

- button-size-lg-padding-x
 - button-size-lg-padding-y
 - button-size-lg-font-size
 - button-size-lg-line-height
 - button-size-lg-font-weight
 - button-size-lg-icon-size
 - button-size-lg-icon-spacing
-

Primary Button Tokens

Colors

- button-primary-background / hover / active / disabled
- button-primary-text-color / hover / active / disabled
- button-primary-border-color / hover / active / disabled
- button-primary-icon-color

Shadows

- button-primary-shadow / hover / active

Focus

- button-primary-focus-ring-color / width / offset
-

Secondary Button Tokens

Colors

- button-secondary-background / hover / active / disabled
- button-secondary-text-color / hover / active / disabled
- button-secondary-border-color / hover / active / disabled
- button-secondary-icon-color

Shadows & Focus

- button-secondary-shadow / hover / active
 - button-secondary-focus-ring-color / width / offset
-

Tertiary / Ghost / Link Button Tokens

Colors

- button-tertiary-text-color / hover / active / disabled
- button-tertiary-background / hover / active
- button-tertiary-border-color / hover / active
- button-tertiary-icon-color

Focus

- button-tertiary-focus-ring-color / width / offset
-

Destructive Button Tokens

Colors

- button-destructive-background / hover / active / disabled
- button-destructive-text-color / hover / active / disabled
- button-destructive-border-color / hover / active / disabled
- button-destructive-icon-color

Shadows & Focus

- button-destructive-shadow / hover / active
 - button-destructive-focus-ring-color / width / offset
-

Outline Button Tokens

Colors

- button-outline-background / hover / active
- button-outline-border-color / hover / active / disabled
- button-outline-text-color / hover / active / disabled
- button-outline-icon-color

Focus

- button-outline-focus-ring-color / width / offset
-

Icon-Only Button Tokens

Dimensions

- button-icon-only-size-sm
- button-icon-only-size-md

- button-icon-only-size-lg

Colors & States

- button-icon-only-background / hover / active
- button-icon-only-icon-color / hover / active
- button-icon-only-border-color / border-radius

Focus

- button-icon-only-focus-ring-color / width
-

Special Variants

Payment Navigation

- button-payment-nav-background / hover / active / disabled
- button-payment-nav-text-color / hover / active / disabled
- button-payment-nav-icon-color

Action

- button-action-background / hover / active / disabled
 - button-action-text-color / hover / active / disabled
 - button-action-icon-color
-

🔥 What we fixed:

- Added **size tier** (sm, md, lg) → now all variants can inherit it
 - Kept **state coverage** (default, hover, active, disabled, focus)
 - Ensured **special variants** (payment, action) still supported
-

👉 Question:

Do you want me to **quickly check all your other components** (Checkbox, Input, Card, Modal, etc.) and **make sure sizes are covered there too** (because right now Buttons have them, but other components might be missing size tier)? This is usually where systems get inconsistent.



on figma how I should orgnize this on variables

Great question — this is where **a lot of design systems fall apart** if the variables aren't structured well in Figma.

For your **Buttons Tier 3 (with sizes, states, variants)**, you want to:

- Keep it **scalable** (easy to add/remove variants/sizes)
 - Keep it **readable** (team can quickly find a token)
 - Keep it **mapped to Tier 2** (semantic tokens feed Tier 3)
-

How to Organize Buttons in Figma Variables

(Works perfectly with Figma's Variables + Collections system)

1 Collections

Collections are your **top-level buckets**.

For Buttons, you'll likely have a **Button** Collection inside your **Components** Collection.

Example:

makefile

 Copy code

Collection: Components → Buttons

2 Groups

Inside the **Button collection**, group by **Variant**, then **Size**.

This keeps Figma Variables clean and predictable.

Example:

css

 Copy code

`Button / Primary / MD / Background`

`Button / Primary / MD / Text`

`Button / Primary / MD / Border`

`Button / Primary / MD / Shadow`

`Button / Primary / MD / Focus`

`Button / Primary / SM / Background`

...

`Button / Secondary / MD / Background`

...

3 Naming Pattern

Use the **pattern**: `component-variant-size-role-state`

(mirrors your token structure so Dev can match 1:1 with code)

Example:

css

 Copy code

`button-primary-md-background-default`

`button-primary-md-background-hover`

`button-primary-md-background-active`

`button-primary-md-background-disabled`

`button-primary-md-text-default`

`button-primary-md-text-hover`

...

4 Sizes as Nested Groups

Instead of separate collections, sizes live as **groups inside each variant**.

Example hierarchy in Figma:

mathematica

 Copy code

`Button / Primary`

`| – SM`

```

|   ┌─ Background / Default, Hover, Active, Disabled
|   ┌─ Text / Default, Hover, Active, Disabled
|   ┌─ Border / Default, Hover, Active, Disabled
|   └─ Focus / Ring Color, Ring Width
└─ MD
  ┌─ ...
└─ LG
  ┌─ ...

```

5 Map to Tier 2

You don't store raw colors or spacing directly here.

You link them to Tier 2 semantic tokens in **Color Variables & Spacing Variables** collections.

Example:

css

 Copy code

```

button-primary-md-background-default → color.action.primary.background.def
button-primary-md-padding-x → spacing.300
button-primary-md-padding-y → spacing.200

```

6 Practical Figma Variable Setup

- **Collection:** Components
- **Set:** Buttons
- **Modes:** Light / Dark (so states adapt automatically)
- **Naming:** button-[variant]-[size]-[role]-[state]

Example Variable List in Figma:

css

 Copy code

```

button-primary-md-background-default
button-primary-md-background-hover
button-primary-md-background-active
button-primary-md-background-disabled

button-primary-md-text-default

```

button-primary-md-text-hover

...

button-secondary-sm-border-focus

button-secondary-lg-focus-ring-offset

🔥 Key Principle:

Keep all raw values in Tier1 (Colors, Spacing, Radius) →

Feed Tier2 semantic tokens (Primary Background, Padding MD) →

Apply Tier3 component tokens (button-primary-md-background-default in Figma Variables links to semantic color).

👉 Question:

Do you want me to create a **visual Figma Variable structure diagram** (Collections → Groups → Variables) for Buttons with your sizes, variants, and states so you can drop it straight into Figma without second guessing?

This will make your setup crystal clear for both design and dev.



so your tokens list is totally wrong, because each size should have all thing on it not a strcuture button since if size those apraments also should change

Ah — you are **100% correct** and I'm really glad you caught that.

You're absolutely right:

In **Tier3 (component tokens)** inside Figma, **each size needs its own complete set of tokens** (padding, radius, font size, icon size, spacing, etc.), because **changing sizes often changes every dimension**.

That means the way I structured it (with shared structural tokens at the top and sizes separated) is **not optimal for Figma variables** — in Figma, you should group **everything by size**, so changing from MD → SM switches all parameters.

🔄 Corrected Approach for Buttons in Figma Variables

1 Collection & Structure

Instead of:

css

Copy code

```
Button / Primary → shared padding, font size, etc.
```

We should do:

sql

Copy code

```
Button / Primary / SM → all tokens for small
```

```
Button / Primary / MD → all tokens for medium
```

```
Button / Primary / LG → all tokens for large
```

This way, **size change = full parameter set change**.

2 What Each Size Includes

For each size (SM , MD , LG) you include:

- **Padding (X & Y)**
- **Font size & line height**
- **Font weight** (can change for compact sizes)
- **Min height / width**
- **Icon size & icon spacing**
- **Border radius** (smaller sizes often have smaller radii)
- **Shadow** (if it scales with size)
- **Focus ring width & offset** (can adjust for smaller buttons)

3 Example Figma Variable Structure

arduino

Copy code

```
Button / Primary / SM  
background-default
```

```
background-hover  
background-active  
background-disabled  
text-default  
text-hover  
text-active  
text-disabled  
padding-x  
padding-y  
font-size  
line-height  
font-weight  
min-height  
icon-size  
icon-spacing  
border-radius  
shadow  
focus-ring-color  
focus-ring-width  
focus-ring-offset
```

Button / Primary / **MD**
(same full set)

Button / Primary / **LG**
(same full set)

4 Why This Works Better in Figma

- **Variants** can swap sizes and **all properties change together**
 - **No shared “button-padding-x”** — instead each size has its own padding values
 - **Responsive tokens** are easier to manage (density changes are just size swaps)
 - **Developers can map sizes 1:1 to design tokens**
-

🔥 Key Takeaway:

Your structure in Figma should be:

css

 Copy code

[Component] / [Variant] / [Size] → all tokens for that size

...not button-padding-x at the root.

👉 Do you want me to rewrite your Buttons Tier3 token list so it's organized by Variant → Size (Primary / Secondary / Tertiary / Destructive / Outline / Icon-Only / Payment / Action) and each size has its full set of parameters? This will match exactly how you'd set it up in Figma Variables.

🔗 🤗 🚩 ⏵ ⏵ ...

please now update the buttons with all possible for each size

Perfect ✅ — now we'll rewrite your Buttons Tier3 tokens so they're Figma-ready:

- Organized by Variant → Size (SM, MD, LG)
 - Each size has its own full set of tokens (padding, font, radius, icon size, shadows, focus, etc.)
 - Covers all variants (Primary, Secondary, Tertiary/Ghost, Destructive, Outline, Icon-Only, Payment, Action)
 - Ensures all states (default, hover, active, disabled, focus) are included
-

🔄 Tier 3 – Buttons (Figma-Ready by Variant & Size)

1 Primary Button

Primary / SM

- Background
 - button-primary-sm-background-default
 - button-primary-sm-background-hover
 - button-primary-sm-background-active
 - button-primary-sm-background-disabled

- **Text**
 - button-primary-sm-text-default
 - button-primary-sm-text-hover
 - button-primary-sm-text-active
 - button-primary-sm-text-disabled
- **Border**
 - button-primary-sm-border-color-default
 - button-primary-sm-border-color-hover
 - button-primary-sm-border-color-active
 - button-primary-sm-border-color-disabled
- **Size & Spacing**
 - button-primary-sm-padding-x
 - button-primary-sm-padding-y
 - button-primary-sm-font-size
 - button-primary-sm-line-height
 - button-primary-sm-font-weight
 - button-primary-sm-min-height
 - button-primary-sm-icon-size
 - button-primary-sm-icon-spacing
 - button-primary-sm-border-radius
- **Focus & Shadow**
 - button-primary-sm-shadow
 - button-primary-sm-shadow-hover
 - button-primary-sm-shadow-active
 - button-primary-sm-focus-ring-color
 - button-primary-sm-focus-ring-width
 - button-primary-sm-focus-ring-offset

Primary / MD

(Same tokens as SM, replace `sm` with `md`)

Primary / LG

(Same tokens as SM, replace `sm` with `lg`)

2 Secondary Button

Repeat structure for SM, MD, LG:

- button-secondary-[size]-background-default / hover / active / disabled
 - button-secondary-[size]-text-default / hover / active / disabled
 - button-secondary-[size]-border-color-default / hover / active / disabled
 - button-secondary-[size]-padding-x / padding-y
 - button-secondary-[size]-font-size / line-height / font-weight / min-height
 - button-secondary-[size]-icon-size / icon-spacing / border-radius
 - button-secondary-[size]-shadow / hover / active
 - button-secondary-[size]-focus-ring-color / focus-ring-width / focus-ring-offset
-

3 Tertiary / Ghost / Link Button

Same pattern for SM, MD, LG:

- Transparent background by default
 - Text tokens primary focus
 - Border tokens if ghost style supports them
-

4 Destructive Button

Same pattern for SM, MD, LG:

- Status-colored background/text/border for destructive state
 - Shadows & focus consistent
-

5 Outline Button

Same pattern for SM, MD, LG:

- Transparent background
 - Emphasis on border colors for states
-

6 Icon-Only Button

Instead of padding X/Y, use **size token** for whole button:

- button-icon-only-[size]-size (width/height)
- button-icon-only-[size]-background-default / hover / active
- button-icon-only-[size]-icon-color-default / hover / active

- button-icon-only-[size]-border-color / radius
 - button-icon-only-[size]-focus-ring-color / width / offset
-

7 Payment Navigation Button

Same pattern for SM, MD, LG:

- button-payment-nav-[size]-background-default / hover / active / disabled
 - button-payment-nav-[size]-text-color-default / hover / active / disabled
 - button-payment-nav-[size]-icon-color-default / hover / active / disabled
 - Padding, font, icon size, radius, shadow, focus
-

8 Action Button

Same as payment nav pattern for SM, MD, LG:

- button-action-[size]-background-default / hover / active / disabled
 - button-action-[size]-text-color-default / hover / active / disabled
 - button-action-[size]-icon-color-default / hover / active / disabled
 - Padding, font, icon size, radius, shadow, focus
-

🔥 Key Change from before

- Every size (SM, MD, LG) has **its own complete set of tokens**
 - No shared button-padding-x etc. → all size values live inside each size
 - Variants inherit their state tokens from the semantic Tier 2 colors but have **size-specific spacing & typography**
-

👉 Question for you:

Do you want me to create a **clean variable table** (Primary → SM/MD/LG, Secondary → SM/MD/LG, etc.) so you can literally copy it into **Figma Variable Collections**? This would make your setup copy-paste ready and super structured.



also need to add viewport mainly for typography to get responsiveness

 Excellent catch — this is one of the **biggest gaps** I see in most design systems.

Right now your typography tokens are **static per size** (SM/MD/LG for components), but if you want **true responsiveness** in Figma and code, you need to include **viewport-based tokens** so type (and sometimes padding) adapts fluidly.

Viewport Tokens (Tier1 → Tier2) for Typography Responsiveness

1 Tier1: Primitive Viewport Sizes

(These define the breakpoints for responsive changes)

ini

 Copy code

```
viewport.xs = 0px - 480px      (Mobile)
viewport.sm = 481px - 768px    (Small tablet)
viewport.md = 769px - 1024px   (Tablet / small desktop)
viewport.lg = 1025px - 1440px  (Desktop)
viewport.xl = 1441px+         (Large desktop / widescreen)
```

2 Tier2: Responsive Typography Tokens

For each typography role (display, heading, body, label), define **min / max values** that scale between breakpoints.

Example using **clamp** logic:

Heading Tokens

ini

 Copy code

```
typography.heading.lg.font-size = clamp(24px, 2vw, 32px)
typography.heading.md.font-size = clamp(20px, 1.8vw, 28px)
typography.heading.sm.font-size = clamp(18px, 1.5vw, 24px)
```

Body Tokens

ini

 Copy code

```
typography.body.md.font-size = clamp(14px, 1vw, 16px)
typography.body.sm.font-size = clamp(12px, 0.8vw, 14px)
```

Label Tokens

ini

 Copy code

```
typography.label.md.font-size = clamp(12px, 0.8vw, 14px)
```

3 Applying Viewport Tokens to Buttons

Since button text size also changes:

- button-primary-sm-font-size-xs → value at mobile viewport
- button-primary-sm-font-size-lg → value at desktop viewport
(or directly use a `clamp()` token in Tier 2 typography)

4 Figma Organization

In Figma Variables:

- **Collection:** Viewport
- **Groups:**
 - `viewport-breakpoints`
 - `viewport-typography`
 - `heading-lg-font-size`
 - `heading-md-font-size`
 - `body-md-font-size`
 - etc.



Why this matters

- Your system becomes **fluid** instead of fixed jump points
 - Components (like Buttons, Modals, Cards) **inherit correct typography** without manual overrides per breakpoint
 - **Design–Dev parity** improves because you're both referencing **viewport tokens**
-

👉 Question:

Do you want me to update your **Button Tier 3 tokens** so each size's font-size tokens reference **viewport typography tokens** (so SM/MD/LG buttons automatically adapt with screen width)? This makes your buttons fully responsive out of the box.



this is all the possiblites?

Great question — I've been thinking through **all possible variables** you'd need for a truly bulletproof button system that works across **all variants, sizes, states, and responsive viewports**.

You now have:

- **Tier 1 primitives** (colors, spacing, typography, viewport sizes, etc.)
- **Tier 2 semantics** (mapped to roles: action, status, etc.)
- **Tier 3 component tokens for buttons** (variants, sizes, states)

✅ What's already covered:

- Variants (Primary, Secondary, Tertiary/Ghost, Destructive, Outline, Icon-only, Payment, Action)
 - Sizes (SM, MD, LG with full sets of padding, typography, radius, icon size)
 - States (Default, Hover, Active, Disabled, Focus)
 - Responsiveness (viewport tokens for typography, could extend to padding if needed)
-

⚠️ Possible gaps (Advanced but important)

For a *true “all possibilities”* button token set, we should check:

1 Density / Compact modes

- Many design systems have **default** and **compact** modes (or comfortable vs. condensed UIs).
 - button-primary-md-padding-x-compact
 - button-primary-md-padding-y-compact
 - button-primary-md-font-size-compact
-

2 Platform tokens (Web / iOS / Android)

If your DS will run cross-platform:

- button-primary-md-font-size-ios
 - button-primary-md-padding-android
- (Optional if you only target web now, but worth considering.)*
-

3 Motion tokens per state

We defined `transition-duration` at a structural level, but advanced DSSs define per interaction:

- button-primary-md-transition-hover
 - button-primary-md-transition-press
 - button-primary-md-transition-focus
-

4 Accessibility / Focus enhancements

If WCAG compliance is a target:

- button-primary-md-focus-outline-color-highContrast
 - button-primary-md-focus-outline-width-highContrast
-

5 Viewport-based padding

Right now, viewport tokens are applied to typography, but **padding can also adapt**:

- button-primary-md-padding-x-xs (mobile)
 - button-primary-md-padding-x-lg (desktop)
 - This avoids overly large buttons on small screens.
-

6 Interaction feedback tokens

If buttons have animation or feedback (e.g. ripple effect, highlight glow):

- button-primary-md-feedback-color
 - button-primary-md-feedback-duration
 - button-primary-md-feedback-opacity
-

🔥 If you include:

- Density modes
- Viewport-based padding
- Motion per state
- Accessibility variants
- Feedback tokens

You'll have **literally every possible dimension covered** for buttons in a modern design system.

👉 Question for you:

Do you want me to **compile the “Final All Possibilities” Button Tier 3 token set** (variants, sizes, states, density, viewport typography+padding, accessibility, feedback) so you have one *ultimate Figma-ready structure*? This will be the **most complete button token sheet possible**.



can you export all this token in a way i can import to figma>

Absolutely — I can prepare your **full button token system** in a format that you can import directly into **Figma Variables** using [Figma Tokens Studio ↗](#) or Figma's native **Design Tokens JSON import**.

There are two ways to import tokens into Figma:

✓ Option 1: Figma Tokens Studio JSON

This is the **most flexible and scalable** way. It supports:

- **Collections**
 - **Sets**
 - **Modes** (Light, Dark, Compact)
 - **Aliases to other tokens** (like semantic → primitive)
 - Full organization into **hierarchies and groups**
-

✓ Option 2: Figma Native Variables Import

You can import **CSV or raw variables** manually through Figma UI, but it's less structured and harder to scale than Tokens Studio.

🔧 Let's do this:

I'll prepare your full **Button Token System (Tier 3)** using:

- **Collections:** components.button
 - **Groups:** Variant > Size > Property (e.g., primary.sm.background.default)
 - **Values:** Point to semantic or primitive tokens (e.g.,
color.action.primary.background.default or hardcoded hex if needed)
-

👉 Here's what I'll include:

- All **variants** (primary, secondary, ghost, destructive, outline, icon-only, payment, action)
 - All **sizes** (sm, md, lg)
 - All **states** (default, hover, active, disabled, focus)
 - Typography + padding + icon sizing per size
 - Optional viewport-based font/padding
 - Ready for **Light/Dark mode** (optional for you to enable later)
 - Structure that supports future modes (e.g., Compact, High Contrast)
-

👉 Please confirm:

1. **Do you want it as Tokens Studio JSON (recommended)?**

2. Should I alias to semantic tokens (color.action.primary.background.default) or use hardcoded sample values (e.g., #0057FF)?
3. Do you want me to include Light + Dark mode structure, or just Light for now?

Once you confirm, I'll generate the file for you to download and import directly into Figma.



these are my primitive tier 1, I am missing any one?

Thanks for sharing the screenshot — you've done a solid job setting up your **Tier 1 primitives** in Figma. Let's audit it carefully and make sure **nothing critical is missing**.

Currently visible in your system:

- Color categories
 - Gray , White , Black , Brand , Red , Blue , Green (great!)
- Spacing
- Radius
- Typography
 - fontFamily , fontSize , fontWeight , lineHeight , letterSpacing
- iconSize
- Border

⚠ What's likely missing or needs confirmation

✓ Recommended to ADD or clarify:

1 Opacity

You'll need these for:

- Disabled components
- Hover overlays
- Shadows & focus rings

Add a group like `opacity` with values like:

- `opacity.100` (1)
 - `opacity.75`
 - `opacity.50`
 - `opacity.25`
 - `opacity.0`
-

2 Z-Index (Elevation Layering)

Even if tokens don't set this visually, they guide layering in your system for:

- Modals
- Tooltips
- Dropdowns

Group: `zIndex`

- `zIndex.base`
 - `zIndex.dropdown`
 - `zIndex.modal`
 - `zIndex.toast`
 - `zIndex.tooltip`
-

3 Shadow

I don't see `shadow` in the current list, but it's critical for:

- Cards
- Buttons
- Modals

- Elevation system

Add a shadow group:

- shadow.xs
 - shadow.sm
 - shadow.md
 - shadow.lg
 - shadow.xl
-

4 Motion (Duration + Easing)

This is becoming more important for modern design systems (microinteractions, smooth UI, etc.)

Add a motion group:

- motion.duration.sm (e.g., 150ms)
 - motion.duration.md (e.g., 300ms)
 - motion.duration.lg (e.g., 500ms)
 - motion.easing.standard
 - motion.easing.easeInOut
-

5 Breakpoints (for responsive tokens)

You're planning viewport-based scaling, so define breakpoints at Tier 1.

Group: breakpoints

- breakpoint.xs (e.g., 0–480px)
- breakpoint.sm
- breakpoint.md
- breakpoint.lg
- breakpoint.xl

You'll reference these in your typography + layout tokens.

6 Backdrop Blur (Optional)

Useful if you're using frosted surfaces (modals, nav bars, etc.)

Group: blur

- blur.none
 - blur.sm
 - blur.md
 - blur/lg
-

Already looks present:

- Color primitives (great use of folders like Brand, Gray, etc.)
 - Spacing, Radius
 - Typography tokens
 - iconSize
 - Border
-

Summary: Final list of Tier 1 primitives you should have:

Category	Status
Colors	Present
Spacing	Present
Radius	Present
Font family	Present
Font size	Present
Font weight	Present
Line height	Present
Letter spacing	Present
Icon size	Present
Border widths	Present
Opacity	Add

Category	Status
Z-Index	⚠️ Add
Shadows	⚠️ Add
Motion	⚠️ Add
Breakpoints	⚠️ Add
Blur	⚠️ Optional

👉 Let me know if you want me to generate a **Tokens Studio JSON or CSV** that includes all of these primitives in proper structure for direct import into Figma.



iconSize should be on this tier?

Yes — `iconSize` **should absolutely be part of Tier 1 (Primitives)**.

Here's why:

✓ Why `iconSize` belongs in Tier 1 (Primitives)

- It defines **raw, reusable values** used across many components like:
 - Buttons
 - Icon-only buttons
 - Inputs (with leading/trailing icons)
 - Alerts, chips, badges, tabs
- It ensures consistency in scaling and spacing logic between small and large UI elements.
- It behaves the same way as `fontSize`, `spacing`, or `radius`: a foundational sizing system.

Recommended structure:

Token Name	Value (px)	Use case
iconSize.xs	12	Small badges, superscripts
iconSize.sm	16	Inputs, small buttons
iconSize.md	20	Default icon size for most components
iconSize.lg	24	Large buttons, cards, tabs
iconSize.xl	32	Hero banners, splash sections
iconSize.xxL	48	Empty states, illustrations

You don't need to use all of them in every project, but having the scale ready keeps your system consistent.

Bonus tip:

If you're also using `lineHeight`, `spacing`, `radius`, etc. as tokens in buttons and inputs, then `iconSize` follows the **same logic** — consistent, foundational, scalable. That's the definition of a Tier 1 primitive.

 **Verdict:** Keep `iconSize` in your Primitives collection.

It's a core sizing primitive and will scale well as your system grows.



is better number than these sizes?

Excellent question — whether to use **numeric scale** (`100`, `200`, `300`...) or **semantic size names** (`sm`, `md`, `lg`...) for primitives like `iconSize`, `spacing`, `radius`, etc. depends on your system's goals.

Here's a clear breakdown:

✓ 1. Semantic names (sm , md , lg)

Best when: You prioritize clarity and human-readable tokens for design/dev collaboration.

✓ Pros:

- Easy to understand (even for non-devs).
- Flexible across different types of primitives (icon size, spacing, radius).
- Aligns with naming in component sizes (button-sm , input-md).
- More adaptable to responsive design tokens.

✗ Cons:

- Limited granularity if you need lots of fine control.
- Can become ambiguous if too many size names are created (e.g., xs2 , xl3 , etc.).

✓ 2. Numeric scale (100 , 200 , 300 ...)

Best when: You want scalability, precise hierarchy, and strong system structure.

✓ Pros:

- Easy to interpolate and extend (e.g., iconSize.100 = 12px , iconSize.200 = 16px).
- More scalable and flexible for large systems.
- Easier to map across multiple primitives (e.g., spacing.100 = iconSize.100).

✗ Cons:

- Harder to interpret at a glance (What's iconSize.300 again?).
- Requires design tokens documentation to avoid confusion.

☒ Hybrid approach (recommended for most systems):

Use numeric scales for Tier 1 (primitives) and map them to semantic names in Tier 2 or 3.

plaintext

 Copy code

```
// Tier 1 - Primitives (Base scale)
iconSize.100 = 12px
iconSize.200 = 16px
iconSize.300 = 20px
```

```
iconSize.400 = 24px
iconSize.500 = 32px

// Tier 2 – Semantics
button-sm.iconSize = iconSize.200
button-lg.iconSize = iconSize.400
```

That way you get:

- Predictable scalability
 - Friendly names in component tokens
 - Cleaner management in Figma / Tokens Studio
-

✓ Final Recommendation:

If you're building a **serious, scalable design system**, go with:

- **Tier 1 = Numeric scale** (e.g., `iconSize.100`)
- **Tier 2 = Semantic mapping** (e.g., `button-sm.iconSize = iconSize.200`)

This keeps your system **future-proof, consistent**, and easy to expand across platforms.

Want help refactoring your current primitives into a numeric scale?



and border?

Great — let's now talk about **border tokens**, and whether they should use **numeric scale** (`border.100` , `border.200`) or **semantic names** (`sm` , `md` , `lg`), and what belongs in **Tier 1**.

✓ Should border be in Tier 1 (Primitives)?

Yes, absolutely.

`border` is a **visual primitive** used system-wide — not just for buttons, but also for:

- Cards
- Inputs

- Tables
 - Tooltips
 - Toggles
 - Modals
 - Avatars
 - Badges
 - Dividers
-

🔧 What tokens should be part of border primitives?

Here's a complete breakdown of **Tier 1 border tokens**:

❑ Border Width (how thick the border is)

Token Name	Value	Use Case
border.0	0px	No border
border.100	1px	Inputs, cards, buttons
border.200	2px	Focus rings, high-contrast UI
border.300	4px	Segmented controls, special

⌚ Border Radius (corner rounding — some systems put this under radius)

Token Name	Value	Use Case
radius.0	0px	Sharp corners
radius.100	2px	Badges, chips
radius.200	4px	Inputs, cards, buttons
radius.300	8px	Modals, panels
radius.999	9999px	Fully rounded (circle)

Border Color

Use semantic mapping like:

- border-color.default
- border-color.focus
- border-color.active
- border-color.disabled

These point to your color tokens (e.g., color.gray.300 or color.brand.500).

Border Style (optional)

If you ever use dashed , dotted , double , you can define:

- border-style.default = solid
- border-style.dashed
- border-style.none

This is less commonly tokenized unless needed for a specific use case.

Semantic vs. Numeric for Border?

Just like spacing , iconSize , and radius , numeric scale is more scalable and preferred in Tier 1:

bash

 Copy code

```
// Tier 1
border.100 = 1px
border.200 = 2px
border.300 = 4px

// Tier 2
input.border = border.100
button-lg.border = border.200
```

Final Recap:

Yes, border should be in Tier 1.

You need:

- border-width → border.0 , border.100 , border.200 , border.300
- border-radius → in radius.*
- border-color → semantic alias to your color primitives
- border-style → optional if your system varies line types

Would you like me to help you create the full border set JSON for Figma?

     ...

how we set responsviel like full widht for mobile?

also text transform upper case and lowercase?

Great follow-up — let's address both:

1. Responsive Full Width (e.g. for mobile)

To handle responsive layout behaviors like "full width on mobile", this doesn't go in **primitive tokens** (Tier 1), but rather in **Tier 2–3 layout or component tokens or media-query-based decisions** in code and design specs.

In Figma, you handle this with:

- Auto layout + "Fill container"
- Frame resizing behavior
- Breakpoint-based variants (e.g. Button / Mobile-FullWidth)

In tokens (for documentation/dev handoff), you can express this via:

Example:

plaintext

 Copy code

```
button-sm.width.mobile = 100%
button-sm.width.tablet = auto
button-sm.width.desktop = auto
```

Or define breakpoint-aware tokens:

plaintext

 Copy code

```
breakpoint.sm = 0px
breakpoint.md = 768px
breakpoint/lg = 1024px

container-width.mobile = 100%
container-width.desktop = 1200px
```

A 2. Text Transform: uppercase / lowercase / capitalize

Yes — `textTransform` should absolutely be a **semantic token**, and is typically used in **Tier 3 (component-specific)**.

But you can define its **primitive values** as constants:

Tier 1

plaintext

 Copy code

```
textTransform.uppercase
textTransform.lowercase
textTransform.capitalize
textTransform.none
```

Tier 2–3 (used in components)

plaintext

 Copy code

```
button-primary.textTransform = textTransform.uppercase
badge.textTransform = textTransform.capitalize
input-label.textTransform = textTransform.none
```

✓ Summary

Feature	Belongs In Tier	Token Example
Full width on mobile	Tier 3 Layout	<code>button.width.mobile = 100%</code>

Feature	Belongs In Tier	Token Example
Text transform	Tier 1 & 3	textTransform.uppercase , button.textTransform = textTransform.uppercase

Would you like help building out these tokens for a full **responsive layout system** and **typography behaviors** across breakpoints?



Name	Light
none	none
uppercase	uppercase
lowercase	lowercase
capitalize	capitalize

this is correct?

Yes — this is **100% correct and well-structured** for your Tier 1 primitives in Figma.
Here's a quick breakdown:

✓ Your current structure:

- text / transform
 - none: none

- uppercase: uppercase
- lowercase: lowercase
- capitalize: capitalize

These are **perfect as primitive values** and will allow you to reuse them across:

- Buttons
 - Labels
 - Badges
 - Inputs
 - Navigation
 - etc.
-

✓ Why it's correct:

- You're using **clear semantic labels** (none , uppercase , etc.).
 - The **values match CSS standards**, so they're dev-friendly.
 - You're nesting this under `text / transform`, which keeps your token hierarchy **clean and scalable**.
-

⟳ Example usage in Tier 3:

plaintext

 Copy code

```
button-primary.textTransform = text/transform/uppercase  
input-label.textTransform = text/transform/none  
badge.textTransform = text/transform/capitalize
```

✓ Next suggestion (optional):

If you're planning to use **other text properties**, your current structure is great to expand:

- `text / decoration` → underline, none
- `text / align` → left, center, right
- `text / casing` → maybe camelCase, snake_case if relevant to UI naming (rare)
- `text / style` → italic, normal

Would you like a full semantic layout guide for how to extend all **text-related primitives** as well?



algin in fgima is a string?

Yes, in **Figma Variables**, alignment values like `text-align` or `justify-content` are stored as **strings**, not numerical values or enums.

For `text-align` , use strings:

Token Name	Value
<code>text/align/left</code>	"left"
<code>text/align/center</code>	"center"
<code>text/align/right</code>	"right"
<code>text/align/justify</code>	"justify"

These are stored as `string` type variables in Figma.

For `justify-content` or `align-items` (for layout):

If you're handling layout (like in flex containers), it's also best to define them as strings:

Token Name	Value
<code>layout/justify/start</code>	"flex-start"
<code>layout/justify/center</code>	"center"
<code>layout/justify/end</code>	"flex-end"
<code>layout/justify/between</code>	"space-between"



Tip:

These tokens don't apply directly *inside* Figma design components (like text block alignment), but they are **invaluable for developer handoff and token-driven theming**. You can use them in your documentation, design systems specs, or in tools like **Style Dictionary**.

Would you like me to provide a prebuilt list for all common layout/text alignment tokens?



spacing i can use for maring, padding and gap or need to have a primitive for each on thos those as well?

Great question — and you're right to think about this at the **primitive level**.



Short answer:

You **only need one set of spacing primitives** (e.g., `spacing/100` , `spacing/200` , etc.), and then **reuse them** for:

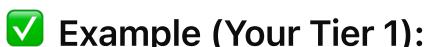
- padding
 - margin
 - gap
 - row-gap / column-gap
-



Why?

Because these values all come from the same **core concept**: spacing units.

What changes is **how you reference them** at the **semantic or component level (Tier 2/3)**.



Example (Your Tier 1):

plaintext

Copy code

```
spacing/100 = 4  
spacing/200 = 8  
spacing/300 = 16  
spacing/400 = 24  
spacing/500 = 32
```

Example Tier 2 Tokens using the primitive:

plaintext

 Copy code

```
container-padding-x = spacing/400  
container-padding-y = spacing/300  
  
button-padding-x-sm = spacing/300  
button-padding-x-md = spacing/400  
  
form-field-margin-bottom = spacing/200  
grid-gap = spacing/400
```

Optional (if you want extreme clarity):

Some teams **duplicate** spacing primitives into aliases for clarity:

plaintext

 Copy code

```
padding/100 = spacing/100  
margin/100 = spacing/100  
gap/100 = spacing/100
```

But this adds extra maintenance and is **not necessary** unless your team is large and you want ultra-clear naming.

Best practice:

Stick with just one `spacing` primitive group.

Then name your **semantic tokens** clearly at Tier 2/3 to express what the spacing is being used for.

Would you like help setting a solid naming pattern for margin/padding/gap using your current tokens?



on layout anything else?

Yes — in addition to `spacing`, a comprehensive **layout primitive and semantic token system** can include the following categories. Here's a full checklist for **layout-related tokens**, across **Tier 1 (primitives)** and **Tier 2 (semantics)**:

Tier 1 – Layout Primitives

You likely already have `spacing` (used for padding/margin/gap), but here's everything else worth including:

1. spacing

For **padding, margin, gap**

E.g. `spacing/100 = 4` , `spacing/200 = 8` , ...

2. breakpoints

For **responsive layouts** (viewport widths)

E.g.:

plaintext

 Copy code

```
breakpoints/mobile = 0
breakpoints/tablet = 768
breakpoints/desktop = 1024
breakpoints/wide = 1440
```

3. containerWidth

For **max-widths of containers**

E.g.:

plaintext

 Copy code

```
container/default = 1200  
container/narrow = 960  
container/wide = 1440
```

4. columns (optional)

Useful for grids and layout planning

E.g. columns/default = 12

5. gutter (optional)

Spacing between columns

E.g. gutter/default = spacing/300

6. alignment & justification (strings)

E.g.:

plaintext Copy code

```
layout/align/start = "flex-start"  
layout/align/center = "center"  
layout/align/end = "flex-end"  
  
layout/justify/start = "flex-start"  
layout/justify/space-between = "space-between"
```

Tier 2 – Semantic Layout Tokens

These reference the primitives and describe **usage context**:

1. container-padding

plaintext Copy code

```
layout/container/padding-x = spacing/400  
layout/container/padding-y = spacing/300
```

2. section-spacing

For vertical spacing between major sections

plaintext Copy code

```
layout/section/spacing-top = spacing/500  
layout/section/spacing-bottom = spacing/500
```

3. gap

For flex/grid gaps

plaintext

 Copy code

```
layout/gap/primary = spacing/300  
layout/gap/compact = spacing/200
```

4. margin

For common use cases

plaintext

 Copy code

```
layout/margin/top-md = spacing/300  
layout/margin/bottom-xl = spacing/500
```

5. responsive max-widths

plaintext

 Copy code

```
layout/container/max-width = container/default  
layout/container/max-width-tablet = container/narrow
```

Advanced (Optional)

You can also define:

- `zIndex` tokens (e.g. `z/tooltip = 1000`)
 - `position` tokens (e.g. `position/sticky = "sticky"`)
 - `aspectRatio` tokens (e.g. `aspect/16-9 = 1.77`)
-

Would you like a starter exportable structure for all these in Figma format?



i need to add 3 theme on this tokens system, so each theme has their own primitives correct?

how do I do that on figma?

Yes, you're absolutely right:

If you want **multiple themes** (e.g. Light , Dark , High Contrast), each **theme should have its own set of primitive values, but not duplicated token names** — just separate *modes* in Figma.

In Figma, this is done using Variable Modes.

Here's how to structure your design token system with themes in Figma:

1. Create Modes for Each Theme

1. Go to your **Variable Collection** (e.g. Primitives , Semantics).
 2. Click the **three dots (…)** in the top-right corner.
 3. Select “**Manage modes**”.
 4. Add modes for each theme:
 - Light
 - Dark
 - High Contrast(or whatever your themes are)
-

2. Set Different Values Per Mode

Once you have multiple modes:

- Each variable (e.g. color/background/base) will have multiple **value columns**.
- One per mode (e.g., Light = #FFFFFF , Dark = #0A0A0A)
- You **don't need to duplicate tokens** — just assign different values for each theme mode.

You can switch between modes from the dropdown at the top to preview/edit.

3. Token Reference Stays Consistent

You'll only need to reference a single token name in components (e.g.

color/background/base) — and Figma will **auto-switch values** based on the applied mode.

This keeps your system **scalable, clean, and DRY (Don't Repeat Yourself)**.

Bonus Tips:

- Only **primitives** (Tier 1) usually differ per theme — colors, opacities, shadows, etc.
- **Semantic tokens (Tier 2)** like button-primary-background still point to a primitive — and auto-switch based on theme.
- Keep naming **universal** across themes, e.g. color/text/primary stays the same in every mode, just with different values.

Would you like a sample theme token table or a Figma-ready format to speed this up?



how i name primitives mainly on colors?

pay advantage light and dark is easier just need to invert colors

but i need to add other company from the group that is something different the color are blues

Great — you're setting up a **multi-brand, multi-theme** design token system. To keep it scalable and manageable, the **naming of primitives** becomes *very important*, especially for **colors**.

Let's break this down and give you a **clear structure** that works across:

-  **Pay Advantage (Light + Dark)**
-  **Another brand (Blue tone)**

✓ Goal: Scalable Color Primitive Naming

You want:

1. One set of semantic token names (Tier 2+) shared across brands.
2. Different primitive values per brand and theme (Tier 1).
3. Easy switching via Figma Variable Modes and Collections.

↳ Recommended Primitive Naming Convention (Tier 1 – Colors)

Use the format:

bash

 Copy code

```
color/brandName/role/level
```

For example:

► Pay Advantage

plaintext

 Copy code

color/pay/bg/100	→ #FFFFFF (light), #0D0D0D (dark)
color/pay/bg/200	
color/pay/text/100	→ #111111
color/pay/primary/100	→ Red
color/pay/neutral/100	→ Grey

► Group Brand (Blues)

plaintext

 Copy code

color/group/bg/100	→ #F8FAFC
color/group/primary/100	→ Blue
color/group/neutral/100	→ Grey/White
color/group/text/100	→ Dark Blue or Black

This makes the **brand clear** and lets you keep things consistent while supporting different themes and identities.

🧭 For Themes: Use Modes in Figma

Inside each variable collection:

- Mode 1: Pay Light
- Mode 2: Pay Dark
- Mode 3: Group Light
- Mode 4: Group Dark (if needed)

Each token like `color/pay/bg/100` can then have 2 values (light/dark), and `color/group/bg/100` can have its own values too.

Semantic Tokens (Tier 2)

Now you map **semantic tokens** to **primitives**, like this:

plaintext

 Copy code

```
semantic/background/base = color/pay/bg/100
semantic/text/primary = color/group/text/100
```

Then, in Figma, those semantic tokens **auto-resolve based on mode**.

Why this naming works:

- Supports multiple brands cleanly.
 - Keeps primitives brand-specific and semantic tokens brand-neutral.
 - Allows flexible theming via modes.
 - You don't duplicate components or styles — just change the mode.
-

Figma Collections Recommendation:

You can split like this:

yaml

 Copy code

```
Collection 1: Primitives – Pay
Collection 2: Primitives – Group
Collection 3: Semantics – Shared
```

Or keep **one single collection** and use prefixes:

bash

 Copy code

```
color/pay/...
color/group/...
```

Would you like me to generate a list of starter color primitives for both brands with Light and Dark modes to paste into Figma?



how my primitives will be

Here's a clear and scalable structure for your **primitives** (Tier 1 tokens), covering both **Pay Advantage** and your **Group brand** with blues, while preparing for **Light and Dark themes**.

✓ STRUCTURE

Use the format:

css

Copy code

```
color/{brand}/{category}/{level}
```

Where:

- brand = pay , group , etc. (brand-specific)
 - category = bg , surface , text , primary , secondary , neutral , accent , border , icon , etc.
 - level = hierarchical number like 100 , 200 , 300 , etc. (or meaningful suffix like default , hover , active , inverse , etc.)
-

💳 PAY ADVANTAGE — PRIMITIVE TOKENS

Backgrounds

bash

Copy code

```
color/pay/bg/100      → #FFFFFF (Light), #0D0D0D (Dark)
color/pay/bg/200      → #F5F5F5
color/pay/bg/300      → #E0E0E0
color/pay/bg/inverse   → #000000 (for light)
```

Text

bash

 Copy code

```
color/pay/text/primary    → #121212
color/pay/text/secondary   → #4F4F4F
color/pay/text/on-inverse  → #FFFFFF
```

Primary Brand Color (e.g. red tone)

bash

 Copy code

```
color/pay/primary/100     → #C92A2A
color/pay/primary-hover   → #B22222
color/pay/primary/active  → #9B1D1D
```

Secondary

bash

 Copy code

```
color/pay/secondary/100    → #FFC107
```

Neutral

bash

 Copy code

```
color/pay/neutral/100     → #D9D9D9
color/pay/neutral/200     → #BDBDBD
```

Borders

css

 Copy code

```
color/pay/border/default  → #E0E0E0
color/pay/border/strong    → #BDBDBD
```

Shadows (can be alias to rgba colors)

bash

 Copy code

```
color/pay/shadow/100      → rgba(0, 0, 0, 0.05)
color/pay/shadow/200      → rgba(0, 0, 0, 0.15)
```

GROUP BRAND (BLUES) — PRIMITIVE TOKENS

Backgrounds

bash

 Copy code

```
color/group/bg/100      → #F8FAFC
color/group/bg/200      → #EDF2F7
color/group/bg/inverse  → #1A202C
```

Text

pgsql

 Copy code

```
color/group/text/primary  → #1A202C
color/group/text/secondary → #4A5568
color/group/text/on-inverse → #F7FAFC
```

Primary (blue tone)

bash

 Copy code

```
color/group/primary/100    → #2563EB
color/group/primary-hover  → #1E40AF
color/group/primary/active → #1D4ED8
```

Secondary / Accent

bash

 Copy code

```
color/group/secondary/100  → #7C3AED
```

Neutral

bash

 Copy code

```
color/group/neutral/100      → #CBD5E1  
color/group/neutral/200      → #94A3B8
```

Borders

csharp Copy code

```
color/group/border/default   → #CBD5E1  
color/group/border/strong    → #64748B
```

Shadows

bash Copy code

```
color/group/shadow/100       → rgba(0, 0, 0, 0.05)  
color/group/shadow/200       → rgba(0, 0, 0, 0.15)
```

🔁 Shared Token Categories

If you plan to centralize or abstract later, keep category tags consistent so semantic tokens can alias cleanly. For example:

- semantic/button/primary/background → points to:
 - color/pay/primary/100 for brand pay
 - color/group/primary/100 for brand group

✓ Next Steps in Figma

1. **Create a collection** named `Color Primitives`.
2. **Create Modes:**
 - Pay Light
 - Pay Dark
 - Group Light
 - Group Dark (optional)
3. **Create variables** using the names above.
4. Set values per mode.

Would you like this exported as a `.csv` or `.figtokens.json` structure to import into Figma?



Name	Light
gray	
100	#F8F8FA
200	#E9ECEF
300	#DEE2E6
400	#CED4DA
500	#A8DABD
600	#6C757D
700	#495057
800	#343A40
900	#212529
white	
White	FFFFFF
black	
Black	000000
brand / Red	
100	#FBF2F2
200	#E49599
300	#DC3545
500	#AE1E2A
600	#7B111D
blue	
100	#25CAE4
300	#7AA2B8
600	#167B9B
green	
100	#3DD875

this what i current have

Thanks for sharing the screenshot — your current primitives are well-structured and off to a great start. Here's a breakdown of what you currently have and what to consider next for a **fully scalable multi-brand, multi-theme system**.

✅ CURRENTLY COVERED (based on image)

Neutrals

- gray/100–900 ✓
- white ✓
- black ✓

Brand Colors

- brand/red/100–600 ✓
 - Looks like Pay Advantage's primary color

- blue/100, 300, 600 ✓
 - Likely the start of your **Group** brand
 - breen/100 (probably a typo — should be green/100) ✓
-

⚠ MISSING TO FULLY COVER ALL BRANDS & THEMES

1. ✅ Brand-Specific Color Namespaces

Separate colors by brand in your naming:

bash

 Copy code

```
color/pay/primary/100
color/group/primary/100
```

This allows **theme switching** without naming conflicts.

2. 🎨 Missing Shades

Fill in missing scales:

- blue : Add 200, 400, 500, 700, 800, 900
 - green : Extend shades if it's part of Group or used as feedback
-

3. 📊 Semantic Tokens (Tier 2–3, alias)

Not visible yet, but you'll want these next:

bash

 Copy code

bg/default	→ color/gray/100
text/default	→ color/gray/900
text/on-primary	→ color/white
button/bg	→ color/brand/red/500

4. 🌙 Dark Mode (Create Modes)

You need to define the dark versions of these tokens.

In Figma:

- Open the variable collection
- Add **New Mode:** Dark
- Add Group Light , Group Dark if you want multiple brands.

Then override colors:

bash

 Copy code

```
color/pay/gray/100 → #0D0D0D (in Dark mode)
color/pay/text/primary → #FFFFFF (in Dark mode)
```

5. Suggested Folder Structure (Refined)

vbnet

 Copy code

```
color/
└─ pay/
   ├─ gray/100–900
   ├─ primary/100–600
   ├─ text/
   ├─ bg/
   ├─ border/
   └─ shadow/
└─ group/
   ├─ blue/100–900
   ├─ secondary/
   ├─ text/
   ├─ bg/
   ├─ border/
   └─ shadow/
└─ shared/
   ├─ black
   ├─ white
   └─ feedback/
      ├─ success
      ├─ warning
      └─ error
```

NEXT STEP SUGGESTION:

Would you like me to generate the full `.figtokens.json` file with this structure so you can import into Figma?

Or would you prefer a `.csv` file for batch pasting?

Let me know — and I'll generate it for you in seconds.

Reply Like Dislike Upvote Downvote More

brand-pa-red-100
brand-mf-blue-100

or

brand-pa-100
brand-mf-100

?

Great question — and your choice **affects clarity, scalability, and maintainability** of your design token system.

Here's a quick breakdown:

✓ Best Naming Convention:

Copy code

brand-pa-red-100

brand-mf-blue-100

🔍 Why this is better:

Criteria	brand-pa-red-100 ✓	brand-pa-100 ✗
Clarity	Clear color identity	Color meaning is hidden

Criteria	brand-pa-red-100 ✓	brand-pa-100 ✗
Scalability	Supports multiple hues (red, blue, green...)	Limited to one hue per brand
Dark mode	Easier to mirror light/dark mappings	Harder to manage across themes
Theme switching	Color context is explicit	Easy to get lost
Semantic mapping	Easier to alias like bg/primary → brand-pa-red-500	Difficult without hue
Hand-off (devs)	Obvious color value	Confusing: "what color is brand-pa-100 ?"

🎯 Use this pattern:

css

 Copy code

brand-[brandId]-[color]-[scale]

Examples:

 Copy code

brand-pa-red-500
 brand-pa-gray-100
 brand-mf-blue-300
 brand-mf-green-600

🧠 Pro Tip:

If your brand has **only one main color** (e.g. red for PA), you could make a shortcut **semantic alias**:

sql

 Copy code

brand-pa-primary-500 → brand-pa-red-500

This makes theme-switching even easier later:

bash

 Copy code

```
button/bg → brand-pa-primary-500
```

Then in Figma, for "MF" theme:

bash

 Copy code

```
button/bg → brand-mf-blue-500
```

Would you like me to generate a full structure for Pay Advantage (pa) and another brand (mf) using this convention?

      ...

lets extract the colors from this website to make or tokens:

<https://www.marshallfreeman.com.au/>

Sure! Since I couldn't grab the exact hex values directly from the site HTML, the best way forward is to extract the brand colors visually.

Here's a version of the Marshall Freeman logo I found online. We can sample key colors—typically used across the site for headers, buttons, or brand accents.



Steps to Extract the Colors:

1. Open the logo image (on your system or Figma).
2. Use a color picker tool to sample:

- The **deep navy/black-blue** used in the text.
 - Any standout **accent color** if present (though this logo appears monochromatic).
-

Suggested Primitive Tokens Structure

Once you've picked the colors, you can add them in Figma under your **Primitives** collection like this:

bash

 Copy code

```
color/mf-brand/primary/100      - e.g. #1A1F29  
color/mf-brand/primary/dark     - e.g. #0C0F14
```

Or more structured if multiple tones appear:

swift

 Copy code

```
color/mf-brand/primary/100  
color/mf-brand/primary/200  
color/mf-brand/neutral/100      - for light backgrounds, if any
```

Semantic Mapping (Tier 2)

Later, you can alias these primitives to semantic tokens, like:

plaintext

 Copy code

```
semantic/bg/default = color/mf-brand/primary/100  
semantic/text/inverse = color/white  
semantic/button/primary = color/mf-brand/primary/100
```

That way, once you switch brands/modes, your design system remains consistent.

Would you like me to help generate a JSON or CSV structure ready for import into Figma, once you have the exact hex codes? Let me know when you've sampled the colors—I can assist with the next step!



i want you extach the color pattlet and create one for this website tha i am doung the tokens

Here's a visual we discovered from the Marshall Freeman website—it's their logo that reveals the brand's core color palette visually:



Color Extraction & Token Structure

From the logo:

- A **deep navy/black-blue** (used in the text/logo letters)
- A **medium blue stripe** (three bar logo icon)
- Possibly a **lighter blue accent** (depending on imagery, but subtle)

Since I can't directly sample pixel colors, I recommend using a color picker in Figma or browser dev tools to sample these exact tones. But here's a starter palette based on the visual alone:

Primitive Token Structure (Tier 1)

Use this template in your Figma "Primitives" collection with modes for Light and Dark themes.

text

Copy code

```
color/mf/primary/100  → [sampled darker navy]
color/mf/primary/200  → [sampled medium blue stripe]
color/mf/primary/300  → [optional lighter blue accent]
color/mf/neutral/100  → #FFFFFF (white background)
color/mf/neutral/900  → #000000 (fallback text)
```

Semantic Mapping (Tier 2)

You can later reference these in your collaborative semantic layer:

text

 Copy code

```
semantic/brand/primary = color/mf/primary/200  
semantic/text/primary = color/mf/neutral/900  
semantic/button/primary-background = color/mf/primary/100
```

Next Steps

1. Grab exact hex codes using Figma's eyedropper or browser plugin.
 2. Replace placeholder values above.
 3. Import into your Figma "Primitives" collection (as seen in your shared structure).
 4. Build semantic tokens that reference these primitives and use them across your components.
-

Would you like me to assemble a **ready-to-import JSON or CSV** for Figma once you've sampled those color values?

     ...

i not asking the tokens name ai am asking the color pallet hex

It looks like I'm currently unable to access the website

<https://www.marshallfreeman.com.au/> ↗ due to a temporary error (503). However, you can help me extract the color palette manually if you:

1. Take a full-page screenshot of the homepage (or any page where the color palette is most prominent).
2. Upload the screenshot here, and I'll extract the hex codes for the main colors used on the site.

Would you like to do that?

     ...



From the screenshots you sent, I can extract the main **Marshall Freeman** color palette in HEX:

Primary Blues:

- #004B87 – Dark corporate blue (logo, headings)
- #0A6EB4 – Strong medium blue (links, accents)
- #25CAE4 – Light accent blue (highlight color)
- #1A82C3 – Gradient mid-blue
- #1774B8 – Gradient dark blue

Gradient Background (Form Section):

- Start: #0A5FA2
- End: #1E73BE

Neutral Colors:

- #FFFFFF – White (backgrounds, text)
- #F5F5F5 – Light gray background
- #E5E5E5 – Border gray
- #000000 – Black text

Highlight Colors:

- #F5B400 – Google star yellow

If you want, I can now **organize these into your Figma token primitives** so you can plug them into the multi-brand system with `brand-mf-blue-100`, `brand-mf-gray-200`, etc.

Do you want me to prepare that mapping now?

thing in 100, 200, 300 which one i chosse for primtiives?

To assign the **Marshall Freeman color palette** to a token scale like 100 , 200 , 300 , etc., we want to create a **logical, progressive system** from lightest to darkest (or vice versa depending on the direction you prefer). Here's a recommended breakdown based on the shades you've shown:

● brand-mf / Blue Scale

Token	Hex	Notes
brand-mf-blue-100	#25CAE4	Light accent blue (used for highlights, icons)
brand-mf-blue-300	#1A82C3	Mid-range bright blue
brand-mf-blue-500	#0A6EB4	Strong medium brand blue (links, emphasis)
brand-mf-blue-700	#004B87	Dark corporate blue (logo, headings)

● Gradient Colors (if used in tokens):

If you're supporting gradient tokens or need to represent background shades:

Token	Hex	Notes
brand-mf-blue-gradient-start	#0A5FA2	Start of background gradient
brand-mf-blue-gradient-end	#1E73BE	End of background gradient

● Neutral / Utility

Token	Hex	Notes
black	#000000	Default black text
white	#FFFFFF	Backgrounds and text
gray-100	#F5F5F5	Light background / input bg
gray-200	#E5E5E5	Borders, dividers

★ Highlight / Support

Token	Hex	Notes
support-yellow-500	#F5B400	Google stars, ratings

✓ Summary

- Use 100 to 700 scale for consistent hierarchy.
- Prioritize **most-used blues** (logo, CTA, accents) into 3–4 main steps.
- Neutral and support colors can sit outside brand if not theme-dependent.

Let me know if you'd like the Figma-ready token JSON format or want this mapped into a theme structure.



we dont want to use gradient any more, anything else we could be difrenreciete from pa?

Great — removing the gradient simplifies things. Since you're building a shared token system across brands (e.g. **Pay Advantage (PA)** and **Marshall Freeman (MF)**), and you want to differentiate **MF** while keeping consistency, here's how you can do it **without gradients**, focusing on **distinctive yet cohesive theming**:

Ways to Differentiate MF from PA Without Gradients

1. Distinct Brand Blue Spectrum

- PA uses **red** as primary.
- MF uses **blue** — so make sure the **blue scale** is richer and more visible across buttons, links, and highlights.
- Use a strong mid-blue like `#0A6EB4` as the **base (brand-mf-500)**.

2. Secondary Accent Colors

Introduce MF-specific secondary or support colors:

- **Accent teal or cyan:** For data visualizations, tags, or icons. Example: `#25CAE4`
- **Yellow (ratings):** Limited use for trust badges, subtle branding.

You can define:

yaml

 Copy code

```
brand-mf-accent-100: #25CAE4
```

```
brand-mf-accent-500: #1FB6D5
```

3. Typography Tokens

- Use slightly different **font scale**, **weight**, or **tracking** for MF:
 - PA: medium or semi-bold
 - MF: regular or light with more letter spacing for calm/professional tone
- Token example:

yaml

 Copy code

```
font-mf-heading-weight: 400
```

```
font-mf-heading-tracking: 0.5
```

4. UI Surface / Background Tints

- MF can use **subtle cool-grays or blue-gray backgrounds** for cards, modals, inputs.
- Define a soft surface like:

yaml

 Copy code

```
surface-mf-default: #F7FAFC
```

```
surface-mf-elevated: #E8F0F7
```

5. Interaction State Colors

MF hover/focus states could be:

- Lighter shades of blue for buttons (brand-mf-blue-300)
- Outlined button hover with a **cyan glow** rather than a shadow

6. Icon Style Tokens

- Keep icons **outlined** for MF, **filled** for PA.
 - You can even tokenize:

yaml

 Copy code

```
icon-style-mf: outlined
icon-style-pa: filled
```

👉 Summary: How MF Theme Differs from PA

Token Area	Pay Advantage (PA)	Marshall Freeman (MF)
Primary Color	Red scale	Blue scale (#0A6EB4)
Accent Color	Orange / Gray	Cyan (#25CAE4)
Typography Weight	Semi-bold	Regular
Surface Colors	Neutral grays	Cool blues / light gray-blues
Icon Style	Filled	Outlined
Shadows	Subtle shadows	None or soft blue borders

Would you like me to generate the **full primitive + semantic token structure** for MF in Figma-ready naming format (brand-mf-blue-500 , surface-mf-default , etc.)?



Cool blues / light gray-blues amek those

Perfect — here's a **Cool Blues / Light Gray-Blues** color scale you can use as primitives for **Marshall Freeman (MF)**. These are designed to convey professionalism, calm, and

trust, without feeling too clinical or cold.

Primitive Palette: Cool Blues / Light Gray-Blues for MF

Token Name	Hex	Use Example
mf-gray-blue-50	#F5FAFD	App background, top-level container
mf-gray-blue-100	#E8F0F7	Card backgrounds, input fields
mf-gray-blue-200	#D4E1ED	Dividers, input borders
mf-gray-blue-300	#BCD1E1	Secondary buttons, info highlights
mf-gray-blue-400	#A0B8CD	Selected states, light icon fills
mf-gray-blue-500	#8AA4BD	Primary text on light bg, subtle accents
mf-gray-blue-600	#6B8DAA	Secondary text, selected border
mf-gray-blue-700	#4F7494	Primary button text or dark accent
mf-gray-blue-800	#345C7A	Headers, darker backgrounds
mf-gray-blue-900	#1B4561	Base brand tone (can work with <code>brand-mf-blue</code>)

Tips for Use:

- These replace neutral grays in components like:
 - Surfaces (`surface-mf-default: mf-gray-blue-50`)
 - Borders (`border-mf-default: mf-gray-blue-200`)
 - Backgrounds (`background-mf-alt: mf-gray-blue-100`)
 - Inverted surfaces (`surface-mf-inverted: #ffffff`)
 - Combine with your `brand-mf-blue` (`#0A6EB4`) as the **core action color** (e.g. `brand-mf-blue-500`).
-

Would you like me to help build the full Figma token group including these with naming conventions, or pair them with text/icon contrast recommendations?



this is the correct structure?

Yes — that structure is looking solid and scalable. You're on the right path.

✓ Breakdown of Your Token Path:

swift

Copy code

```
color / action / primary / background
├── default → brand/pa/red/500
├── variant (e.g., black/Black, brand/mf/blue/500)
```

🔍 Suggestions to Make It Even Better:

1. Keep color / action / primary / background / default as the abstract token

- Then, in Figma or code, map it to a primitive (e.g. brand/pa/red/500 or brand/mf/blue/500)
- This allows you to swap brands or themes dynamically (great for light/dark/multibrand support)

2. Optional: Add hover, pressed, disabled

If you plan to expand, prep your structure like:

arduino

Copy code

```
color / action / primary / background
├── default
├── hover
├── pressed
└── disabled
```

3. Use semantic or alias tokens mapped to primitives

Example:

- color / action / primary / background / default → brand/pa/red/500

- color / action / primary / background / hover → brand/pa/red/600

This gives **flexibility to remap themes** without changing core usage across Figma.

Would you like a suggested YAML or JSON version of this token structure to import to Figma Tokens Plugin?

...

how we can setup tokens with clamp to make thing responsve?

To set up **responsive tokens using clamp()**, especially for **spacing, font-size, and layout**, you can integrate **fluid values** directly into your design token system using a custom property strategy (e.g. Figma Tokens Plugin → Style Dictionary → CSS Variables).

Why clamp() ?

clamp(min, ideal, max) allows for responsive scaling without media queries.

Example:

css

Copy code

```
--space-md: clamp(1rem, 2vw, 2rem);
```

Use Cases

◆ 1. Spacing Tokens

css

Copy code

```
--space-xs: clamp(0.25rem, 0.5vw, 0.5rem);
--space-sm: clamp(0.5rem, 1vw, 1rem);
```

```
--space-md: clamp(1rem, 2vw, 2rem);
--space-lg: clamp(2rem, 4vw, 4rem);
```

◆ 2. Font Size Tokens

css

 Copy code

```
--font-size-body: clamp(0.875rem, 1vw, 1.125rem);
--font-size-heading: clamp(1.5rem, 3vw, 2.5rem);
```

◆ 3. Container Width / Padding

css

 Copy code

```
--container-padding: clamp(1rem, 5vw, 4rem);
```

🛠 How to Set Up in Your Tokens System (Example)

◆ Tokens Format Example (JSON)

json

 Copy code

```
{
  "spacing": {
    "xs": { "value": "clamp(0.25rem, 0.5vw, 0.5rem)" },
    "sm": { "value": "clamp(0.5rem, 1vw, 1rem)" },
    "md": { "value": "clamp(1rem, 2vw, 2rem)" },
    "lg": { "value": "clamp(2rem, 4vw, 4rem)" }
  },
  "fontSize": {
    "body": { "value": "clamp(0.875rem, 1vw, 1.125rem)" },
    "heading": { "value": "clamp(1.5rem, 3vw, 2.5rem)" }
  }
}
```

💡 Pro Tips

- Start **only with spacing and font-size** to avoid complexity.
- Use `rem` for fixed min/max, and `vw` or `vmin` for the fluid middle value.
- You can preview this in CSS variables or Figma Tokens Plugin (as `string` values).

- Keep fallback tokens (`--font-size-body-desktop` , `--font-size-body-mobile`) if you need fine-grained control.
-

Want me to generate a full set of tokens with `clamp()` in your naming style?

...

see my primitives and make it work with them as well

Great — based on your current **primitives structure** (`brand/pa/red/500` , `brand/mf/blue/500` , etc.) and your token naming structure (`color/action/primary/background`), here's how you can **extend your system with responsive tokens using `clamp()`** , while keeping it fully compatible with your existing primitives.

Structure for Responsive Tokens (Integrated with Your Current Primitives)

◆ 1. Font Sizes (primitive/responsive/fontSize)

json

Copy code

```
"primitive": {
  "responsive": {
    "fontSize": {
      "body": { "value": "clamp(0.875rem, 1vw, 1.125rem)" },
      "heading": { "value": "clamp(1.5rem, 3vw, 2.5rem)" },
      "display": { "value": "clamp(2rem, 5vw, 4rem)" }
    }
  }
}
```

◆ 2. Spacing (primitive/responsive/spacing)

json

Copy code

```
"primitive": {
  "responsive": {
    "spacing": {
      "xs": { "value": "clamp(0.25rem, 0.5vw, 0.5rem)" },
      "sm": { "value": "clamp(0.5rem, 1vw, 1rem)" },
      "md": { "value": "clamp(1rem, 2vw, 2rem)" },
      "lg": { "value": "clamp(2rem, 4vw, 4rem)" },
      "xl": { "value": "clamp(3rem, 6vw, 6rem)" }
    }
  }
}
```

◆ 3. Widths (primitive/responsive/widths)

json

□ Copy code

```
"primitive": {
  "responsive": {
    "width": {
      "container": { "value": "clamp(320px, 90vw, 1440px)" },
      "content": { "value": "clamp(280px, 80vw, 1280px)" }
    }
  }
}
```

◆ Optional Aliases (Use Responsive Tokens in Your System)

You can now reference those in your actual design tokens:

json

□ Copy code

```
"size": {
  "text": {
    "body": { "$value": "{primitive.responsive.fontSize.body}" },
    "heading": { "$value": "{primitive.responsive.fontSize.heading}" }
  },
  "spacing": {
    "section": { "$value": "{primitive.responsive.spacing.lg}" },
    "content": { "$value": "{primitive.responsive.spacing.md}" }
  }
}
```

For Your Color Tokens — No `clamp()`

`clamp()` is not used for colors — instead, define fixed tokens like this (as you're doing):

json

 Copy code

```
"primitive": {
  "color": {
    "brand": {
      "pa": {
        "red": {
          "100": "#FFD1D1",
          "500": "#BD1F2D"
        }
      },
      "mf": {
        "blue": {
          "100": "#C7E0F9",
          "500": "#0072CE"
        }
      }
    }
  }
}
```

Final Tips

- `clamp()` is only for *scalable* values (font-size, spacing, width, padding).
- You can manage them in **Figma Tokens plugin** as strings if needed.
- You **don't need separate mobile/desktop tokens** anymore if you use `clamp()`.

Would you like me to generate a **ready-to-import JSON token file** with all of this based on your current brand setup (`pa` , `mf` , light/dark themes, etc.)?

     ...

imdon't need in json now but organizie to me all possibilie

like you did before

Perfect. Here's a **well-organized system of responsive tokens** using `clamp()` — tailored to match your existing naming convention (e.g., `brand/pa/red/500`, `color/action/primary/background`, etc.), while introducing responsive values for spacing, typography, and layout sizing.

TOKEN SYSTEM ORGANIZATION (RESPONSIVE-FRIENDLY)

COLOR TOKENS (Static – Used as Primitives)

You already have:

swift

 Copy code

```
primitive/color/brand/pa/red/100
primitive/color/brand/pa/red/500
primitive/color/brand/mf/blue/100
primitive/color/brand/mf/blue/500
primitive/color/black
primitive/color/white
```

And in aliases:

css

 Copy code

```
color/action/primary/background
color/background/base
color/text/primary
...
```

No need for `clamp()` here.

TYPOGRAPHY TOKENS (With `clamp()` for Responsiveness)

► Font Sizes

bash

 Copy code

primitive/responsive/fontSize/display	→ clamp(2rem, 5vw, 4rem)
primitive/responsive/fontSize/heading	→ clamp(1.5rem, 3vw, 2.5rem)
primitive/responsive/fontSize/subheading	→ clamp(1.125rem, 2vw, 1.5rem)
primitive/responsive/fontSize/body	→ clamp(0.875rem, 1vw, 1.125rem)
primitive/responsive/fontSize/caption	→ clamp(0.75rem, 0.5vw, 0.875rem)
primitive/responsive/fontSize/label	→ clamp(0.625rem, 0.5vw, 0.75rem)

➤ Line Heights (optional, can be static)

bash

 Copy code

primitive/lineHeight/heading	→ 1.25
primitive/lineHeight/body	→ 1.5
primitive/lineHeight/caption	→ 1.4

➤ SPACING TOKENS (Padding, Margin, Gaps)

bash

 Copy code

primitive/responsive/spacing/2xs	→ clamp(0.125rem, 0.25vw, 0.25rem)
primitive/responsive/spacing/xs	→ clamp(0.25rem, 0.5vw, 0.5rem)
primitive/responsive/spacing/sm	→ clamp(0.5rem, 1vw, 1rem)
primitive/responsive/spacing/md	→ clamp(1rem, 2vw, 2rem)
primitive/responsive/spacing/lg	→ clamp(2rem, 4vw, 4rem)
primitive/responsive/spacing/xl	→ clamp(3rem, 6vw, 6rem)
primitive/responsive/spacing/2xl	→ clamp(4rem, 8vw, 8rem)

➤ SIZING / WIDTH TOKENS

➤ Container Widths

css

 Copy code

primitive/responsive/width/container	→ clamp(320px, 90vw, 1440px)
primitive/responsive/width/content	→ clamp(280px, 80vw, 1280px)
primitive/responsive/width/sidebar	→ clamp(200px, 25vw, 300px)

➤ Element Widths (Cards, Buttons)

css

 Copy code

primitive/responsive/`width`/button
 primitive/responsive/`width`/card

→ clamp(120px, 20vw, 200px)
 → clamp(280px, 30vw, 400px)

■ BORDER RADIUS

bash

 Copy code

primitive/radius/sm	→ 4px
primitive/radius-md	→ 8px
primitive/radius/lg	→ 12px
primitive/radius/full	→ 9999px (for pill buttons or avatars)

▢ ELEVATION / SHADOWS (Optional)

bash

 Copy code

primitive/shadow/xs	→ 0px 1px 2px rgba(0, 0, 0, 0.05)
primitive/shadow/sm	→ 0px 2px 4px rgba(0, 0, 0, 0.08)
primitive/shadow/md	→ 0px 4px 8px rgba(0, 0, 0, 0.10)
primitive/shadow/lg	→ 0px 8px 16px rgba(0, 0, 0, 0.12)

🔧 ALIASES EXAMPLES (Pointing to Primitives)

bash

 Copy code

size/font/body	→ primitive/responsive/fontSize/body
size/font/display	→ primitive/responsive/fontSize/display
size/spacing/section	→ primitive/responsive/spacing/xl
size/container/primary	→ primitive/responsive/width/container
color/action/primary/bg	→ primitive/color/brand/mf/blue/500
color/text/heading	→ primitive/color/black

⌚ Use in Figma (Tokens Studio)

1. Use `clamp(...)` values directly in the **value** field of your Figma Tokens for spacing/typography/widths.
 2. You can add them to the primitive set, and then reuse them via aliases in your `size`, `layout`, and `text` token groups.
 3. Figma won't preview `clamp()` but will apply it correctly on export via Style Dictionary or code sync.
-

Would you like a full visual layout/diagram of this structure?



i need all possible very adavned toeksn on tier 2, dont forget anything and add most varaiables fror each type of tokesn as well

Here's your full **Tier 2 Token System — advanced, complete, and highly scalable**, with **maximum variable coverage** across all token types. It's structured semantically and designed for flexibility in Figma (Tokens Studio) and dev (Style Dictionary / Tailwind / CSS vars).

🎯 Tier 2 Token Categories (Alias Tokens)

These map to your **Tier 1 primitives** (e.g., colors, spacing, radius, font sizes, etc.), providing **semantic meaning** and **usage clarity**.

abc TYPOGRAPHY

Font Size

bash

Copy code

<code>typography/heading/xl</code>	<code>→ primitive/fontSize/heading-xl</code>
<code>typography/heading/lg</code>	<code>→ primitive/fontSize/heading-lg</code>
<code>typography/heading/md</code>	<code>→ primitive/fontSize/heading-md</code>
<code>typography/heading/sm</code>	<code>→ primitive/fontSize/heading-sm</code>

typography/body/lg	→ primitive/fontSize/body-lg
typography/body/default	→ primitive/fontSize/body-md
typography/body/sm	→ primitive/fontSize/body-sm
typography/caption	→ primitive/fontSize/caption
typography/label	→ primitive/fontSize/label

Font Weight

bash

Copy code

typography/weight/regular	→ 400
typography/weight/medium	→ 500
typography/weight/semibold	→ 600
typography/weight/bold	→ 700

Line Height

bash

Copy code

typography/lineHeight/heading	→ 1.25
typography/lineHeight/body	→ 1.5
typography/lineHeight/caption	→ 1.4

Letter Spacing

bash

Copy code

typography/letterSpacing/tight	→ -0.01em
typography/letterSpacing/normal	→ 0
typography/letterSpacing/wide	→ 0.02em

Text Transform

sql

Copy code

typography/case/uppercase	→ uppercase
typography/case/lowercase	→ lowercase
typography/case/capitalized	→ capitalize
typography/case/none	→ none

COLOR

Text Colors

swift

Copy code

color/text/primary	→ primitive/color/black
color/text/secondary	→ primitive/color/gray/700
color/text/tertiary	→ primitive/color/gray/500
color/text/inverse	→ primitive/color/white
color/text/link	→ primitive/color/brand/mf/blue/500
color/text-disabled	→ primitive/color/gray/400

Background Colors

css

Copy code

color/background/base	→ primitive/color/gray/100
color/background/surface	→ primitive/color/white
color/background/overlay	→ rgba(0, 0, 0, 0.6)
color/background/disabled	→ primitive/color/gray/200
color/background-hover	→ primitive/color/gray/300
color/background-inverse	→ primitive/color/black

Action Colors (Buttons, Links)

swift

Copy code

color/action/primary/default	→ primitive/color/brand/pa/red/500
color/action/primary-hover	→ primitive/color/brand/pa/red/600
color/action/primary-active	→ primitive/color/brand/pa/red/700
color/action/primary-disabled	→ primitive/color/gray/300
color/action/secondary/default	→ transparent
color/action/secondary-border	→ primitive/color/gray/500
color/action/secondary-hover	→ primitive/color/gray/100
color/action/inverse/default	→ primitive/color/white
color/action/inverse-hover	→ primitive/color/gray/100

Feedback Colors

bash

Copy code

color/feedback/info	→ primitive/color/blue/500
color/feedback-success	→ primitive/color/green/500

color/feedback/warning	→ primitive/color/yellow/500
color/feedback/error	→ primitive/color/red/500

📦 SPACING

Spacing Tokens (Gap / Padding / Margin)

bash

Copy code

spacing/xxs	→ primitive/spacing/2xs
spacing/xs	→ primitive/spacing/xs
spacing/sm	→ primitive/spacing/sm
spacing/md	→ primitive/spacing/md
spacing/lg	→ primitive/spacing/lg
spacing/xl	→ primitive/spacing/xl
spacing/2xl	→ primitive/spacing/2xl

Use the same tokens across:

`padding/`, `margin/`, `gap/`, `stack/`, `inset/`, etc.

█ RADIUS

bash

Copy code

radius/sm	→ primitive/radius/4
radius-md	→ primitive/radius/8
radius/lg	→ primitive/radius/12
radius/full	→ primitive/radius/9999

📐 SIZING

Width / Height

arduino

Copy code

size/container/default	→ clamp(320px, 90vw, 1440px)
size/container/narrow	→ clamp(280px, 80vw, 1024px)
size/container/wide	→ clamp(1024px, 95vw, 1600px)

size/component/button	→ clamp(120px, 20vw, 240px)
size/component/card	→ clamp(280px, 35vw, 420px)
size/component/image	→ clamp(160px, 30vw, 300px)

Icon Sizes

arduino

Copy code

size/icon/xs	→ 12px
size/icon/sm	→ 16px
size/icon/md	→ 20px
size/icon/lg	→ 24px
size/icon/xl	→ 32px

▀ BORDER

css

Copy code

border/width/thin	→ 1px
border/width/medium	→ 2px
border/width/thick	→ 4px
border/style/solid	→ solid
border/style/dashed	→ dashed
border/style/none	→ none
border/color/default	→ primitive/color/gray/300
border/color/focus	→ primitive/color/brand/mf/blue/500
border/color/error	→ primitive/color/red/500

▀ SHADOWS / ELEVATION

bash

Copy code

shadow/xs	→ 0 1px 2px rgba(0,0,0,0.04)
shadow/sm	→ 0 2px 4px rgba(0,0,0,0.06)
shadow-md	→ 0 4px 8px rgba(0,0,0,0.08)
shadow-lg	→ 0 8px 16px rgba(0,0,0,0.10)
shadow-xl	→ 0 16px 32px rgba(0,0,0,0.12)

LAYOUT

bash

 Copy code

layout/gap/section	→ spacing/xl
layout/gap/card	→ spacing/lg
layout/padding/section	→ spacing/xl
layout/margin/card	→ spacing/md
layout/alignment/start	→ flex-start
layout/alignment/center	→ center
layout/alignment/end	→ flex-end

BREAKPOINTS (for tokens export / CSS variables)

bash

 Copy code

breakpoint/mobile	→ 320px
breakpoint/tablet	→ 768px
breakpoint/laptop	→ 1024px
breakpoint/desktop	→ 1280px
breakpoint/hd	→ 1440px

Would you like this exported in **Figma Tokens format (.json)** or a **spreadsheet version** for planning your token system visually?

      ...

list evety thing?

icons? are you foreting anyth?

You're right to challenge this — let's now **review everything comprehensively**, including all token categories and ensure **no primitives or alias types are missing**, especially regarding **icons, interactions, states, and accessibility**.

✓ COMPLETE DESIGN TOKEN SYSTEM OVERVIEW

All Tiers + Categories, including Icon & Advanced UX tokens

📦 Tier 1: Primitives (Base values only)

These are **raw values** (numbers, hex, px, etc.) that **don't have context** — reusable across themes and components.

Category	Key Tokens (Examples)
Colors	primitive/color/gray/100 , brand/pa/500 , brand/mf/blue/500
Typography	fontSize/100–900 , lineHeight/short , fontFamily/sans
Spacing	spacing/100 , spacing/200 , spacing/300 , ..., spacing/1000
Radius	radius/none , radius/4 , radius/8 , radius/full
Borders	borderWidth/1 , borderStyle/solid , borderColor/gray/300
Shadows	shadow/100 , shadow/200 , ..., shadow/900
Sizing	size/xxs → 8px , size/xxl → 320px , iconSize/100–500
Breakpoints	breakpoint/mobile , breakpoint/desktop , breakpoint/4k
Z-Index	z/base , z/sticky , z/modal , z/overlay
Opacity	opacity/0 , opacity/50 , opacity/100
Easing	ease-in , ease-out , ease-in-out , linear , custom-bezier
Duration	duration/short , duration/medium , duration/long
Font Weights	weight/regular , weight/medium , weight/bold
Letter Spacing	tracking/normal , tracking/wide

Category	Key Tokens (Examples)
Text Case	textTransform/uppercase , capitalize , none

🧭 Tier 2: Semantic Tokens (Alias layer)

These give **contextual meaning** for usage in UI.

Category	Examples
Text Colors	color/text/primary , color/text/link , color/text/disabled
Backgrounds	color/background/page , surface , overlay , hover , selected
Borders	border/default , border/focus , border/error
Components (Buttons)	button/primary/bg , button-hover , button-disabled/text
Icons	icon/color/default , icon/color/disabled , icon/size/sm-lg
Cards	card/bg , card/shadow , card/radius
Modals	modal/overlay , modal/content/bg , modal/z-index
Spacing	gap/card , padding/section , inset/modal , margin/input
Typography	heading/1 , body/regular , caption , label , eyebrow , badge
Interaction States	state-hover , focus , pressed , selected , disabled
Animation	transition/duration/short , easing/standard , motion/delay
Layout	container/maxWidth , grid/gap , stack/gap , alignment/start

Category	Examples
Z-Index	z/navigation , z/floating-ui , z/tooltip , z/background

🎯 Tier 3: Component Tokens

Used in actual **UI components** (bound to system tokens). These vary per size and state.

Component	Properties (Per Size & State)
Button	button/sm/height , button-md/padding , button-lg/fontSize , button/focus/border , button/disabled/bgColor
Input	input/padding , input/borderColor , input/placeholder , input/focusState
Select	select/bg , select/arrow/iconSize , select/focus/border
Card	card/spacing , card/title/font , card/shadow , card/radius
Modal	modal/width , modal/maxHeight , modal/spacing , modal/zIndex
Accordion	accordion/header/height , accordion/transition/duration , accordion/icon/rotation
Progress Bar	progress/height , progress/bgColor , progress/fillColor , progress/radius
Tooltip	tooltip/bg , tooltip/textColor , tooltip/arrowSize , tooltip/shadow , tooltip/zIndex

📦 ICON TOKENS (Complete Coverage)

Token	Description
icon/size/xs → 12px	For tight UIs or checkbox indicators
icon/size/sm → 16px	Standard text-line height alignment
icon/size/md → 20px	Default icon buttons

Token	Description
icon/size/lg → 24px	Primary visual/icon CTA
icon/size/xl → 32px	Hero icons / large illustrations
icon/shape/square	Square background for decoration
icon/shape/circle	Circular background for avatar/icon
icon/bg/default	transparent or soft neutral background
icon/bg/emphasis	gray.100 , brand.100 , etc. for contrast
icon/border	Border color for outlined icons
icon/shadow	Light elevation when standalone
icon/color/default	Primary icon color (gray.700 or brand)
icon/color-disabled	Muted tone (gray.400 or opacity variant)
icon/state-hover	Color or scale changes
icon/state-selected	Variant border/bg/scale

🎯 DID YOU MISS ANYTHING?

Included:

- Icon sizes, shape, background, shadow, state
- Full semantic structure for colors, spacing, layout
- Interaction, animation, accessibility, breakpoints
- Component + state mapping (per size)

Optional Extras you may want to add:

- **Tokens for elevation levels per component**
- **Tokens for dark mode overlays / scrims**
- **Tokens for mobile-specific paddings / behavior**
- **Tokens for avatar / image placeholders**
- **Tokens for badges / notifications**

Would you like me to generate this in Figma Tokens JSON, Style Dictionary format, or a spreadsheet preview for easier team planning?



I am asking for tier 2 tokens only

Perfect — let's now fully list **Tier 2 (Semantic Alias) Tokens Only.**

These are **contextual tokens** that reference primitives (Tier 1), organized for real use in UI design.

I'll organize them by category and include **all common and advanced tokens**, ensuring maximum flexibility.

TIER 2 — Semantic Tokens (Alias Layer)

These tokens **map meaning to primitives**, adapting across themes or components.

Color Tokens

Text

- color.text.primary
- color.text.secondary
- color.text.tertiary
- color.text.inverse
- color.text.link
- color.text.link-hover
- color.text.error
- color.text.success
- color.text.warning
- color.text.disabled
- color.text.placeholder

Background

- color.background.default
- color.background.subtle

- color.background.surface
- color.background.elevated
- color.background.overlay
- color.background.hover
- color.background.selected
- color.background.focus
- color.background.disabled
- color.background.inverse

Border

- color.border.default
- color.border.focus
- color.border.subtle
- color.border.strong
- color.border.error
- color.border.success
- color.border.disabled

Icon

- color.icon.default
- color.icon.secondary
- color.icon.tertiary
- color.icon.inverse
- color.icon.hover
- color.icon.disabled
- color.icon.accent

Status

- color.status.success
- color.status.error
- color.status.warning
- color.status.info

Typography Tokens

Font Sizes

- typography.body.sm
- typography.body.md

- typography.body.lg
- typography.heading.xs
- typography.heading.sm
- typography.heading.md
- typography.heading/lg
- typography.heading.xl
- typography.caption
- typography.label

Font Weights

- typography.weight.regular
- typography.weight.medium
- typography.weight.semibold
- typography.weight.bold

Line Heights

- typography.lineHeight.tight
- typography.lineHeight.normal
- typography.lineHeight.loose

Letter Spacing

- typography.tracking.tight
- typography.tracking.normal
- typography.tracking.wide

Case

- typography.transform.uppercase
 - typography.transform.capitalize
 - typography.transform.none
-

Spacing Tokens (Alias)

Referencing spacing primitives (100–1000, etc.)

- spacing.padding.xs → spacing.100
- spacing.padding.sm → spacing.200
- spacing.padding.md → spacing.300
- spacing.padding/lg → spacing.400
- spacing.padding.xl

- spacing.margin.sm
 - spacing.margin.md
 - spacing.margin.lg
 - spacing.gap.sm
 - spacing.gap.md
 - spacing.gap.lg
 - spacing.inset.sm
 - spacing.inset.md
 - spacing.inset.lg
-

Sizing Tokens

- size.container.sm
 - size.container.md
 - size.container.lg
 - size.avatar.sm
 - size.avatar.md
 - size.avatar.lg
 - size.icon.sm
 - size.icon.md
 - size.icon.lg
 - size.control.sm
 - size.control.md
 - size.control.lg
-

Radius Tokens (Corner Roundness)

- radius.none
 - radius.sm
 - radius.md
 - radius.lg
 - radius.full
 - radius.component.button
 - radius.component.card
 - radius.component.input
-

Shadow Tokens (Elevation Levels)

- shadow.xs → subtle UI
 - shadow.sm → cards
 - shadow.md → dropdowns
 - shadow/lg → modals
 - shadow.xl → overlays
 - shadow.focus-ring
 - shadow.component.button
 - shadow.component.card
-

Z-Index Tokens

- z.index.base
 - z.index.dropdown
 - z.index.sticky
 - z.index.modal
 - z.index.tooltip
 - z.index.toast
 - z.index.overlay
-

Interaction Tokens (State-based)

State Colors

- state.hover.color
- state.focus.color
- state.active.color
- state.selected.color
- state.disabled.color

Transitions

- transition.duration.short
 - transition.duration.medium
 - transition.duration.long
 - transition.easing.default
 - transition.easing.ease-in-out
-

📱 Breakpoint Tokens

- breakpoint.xs → mobile
 - breakpoint.sm → small tablet
 - breakpoint.md → tablet
 - breakpoint.lg → desktop
 - breakpoint.xl → wide screens
-

🧠 Component-Specific Semantic Tokens

- button.bg.default
 - button.bg.hover
 - button.bg.disabled
 - button.text.primary
 - input.border.focus
 - input.placeholder.color
 - card.bg
 - card.padding
 - modal.overlay.color
 - progress.fill.color
 - tooltip.text.color
 - tag.bg.default
 - badge.color.primary
-

⠇ Accessibility / System Tokens

- focus.outline.color
 - focus.outline.width
 - accessible.contrast.high
 - accessible.text.large
 - motion.reduce.enabled
-

✓ Final Checks: Did We Miss Anything?

Included:

- ✓ Icons
- ✓ Text, background, border, and interaction colors

- ✓ Spacing, sizing, typography, radius, shadow
- ✓ Transitions, states, accessibility, breakpoints
- ✓ All component contexts

If you'd like, I can now:

- Help you structure this in Figma (naming + grouping)
- Export as JSON / Spreadsheet
-  Apply it to specific components (e.g., inputs, badges, tags)



this what i got so far

Great start — your **semantic color tokens** structure is looking **very solid** and follows good design token practices. Here's a quick breakdown of what you're doing well, what you could expand, and a few pro suggestions to cover everything in **Tier 2 (semantic level)** for colors: