

provide a full list and most complete and advanced for complex SAAS design tokens a list of tokens on tier 2

Here's a **Tier 2-level** list of advanced design tokens for a complex SaaS design system. Tier 1 usually covers the pure **core primitives** (e.g., base colors, font families, spacing units, breakpoints), while **Tier 2 maps those primitives into semantic, contextual, and component-level tokens** that directly serve your product's UI patterns.

1. Color Tokens (Semantic Layer)

These are **role-based** rather than raw colors.

- color-bg-primary
- color-bg-secondary
- color-bg-tertiary
- color-bg-inverse
- color-bg-accent
- color-bg-disabled
- color-bg-hover
- color-bg-active
- color-bg-selected
- color-bg-success
- color-bg-warning
- color-bg-error
- color-bg-info
- color-border-primary
- color-border-secondary
- color-border-accent
- color-border-disabled
- color-border-focus
- color-border-error
- color-text-primary
- color-text-secondary
- color-text-inverse
- color-text-accent
- color-text-disabled

- color-text-link
 - color-text-link-hover
 - color-icon-primary
 - color-icon-secondary
 - color-icon-disabled
 - color-icon-accent
-

2. Typography Tokens (Contextual)

Applied meaningfully to hierarchy & context.

- font-heading-xl
 - font-heading-lg
 - font-heading-md
 - font-heading-sm
 - font-heading-xs
 - font-body-lg
 - font-body-md
 - font-body-sm
 - font-body-xs
 - font-label-lg
 - font-label-md
 - font-label-sm
 - font-mono-md (code blocks, terminal UIs)
 - line-height-tight
 - line-height-normal
 - line-height-loose
 - letter-spacing-tight
 - letter-spacing-normal
 - letter-spacing-loose
-

3. Spacing & Layout Tokens (Component-Specific)

Spacing that maps to UI purposes, not just raw values.

- space-inset-sm (padding inside components)
- space-inset-md
- space-inset-lg

- `space-stack-sm` (vertical spacing between stacked elements)
 - `space-stack-md`
 - `space-stack-lg`
 - `space-inline-sm` (horizontal spacing between items in a row)
 - `space-inline-md`
 - `space-inline-lg`
 - `space-gap-card` (grid/card gap)
 - `space-gap-form` (form element spacing)
 - `space-gap-list` (list item spacing)
-

4. Border & Radius Tokens

For corners, outlines, and UI shapes.

- `radius-none`
 - `radius-sm`
 - `radius-md`
 - `radius-lg`
 - `radius-pill` (full rounded)
 - `radius-circle`
 - `border-width-thin`
 - `border-width-medium`
 - `border-width-thick`
 - `border-style-solid`
 - `border-style-dashed`
 - `border-style-dotted`
-

5. Shadow & Elevation Tokens

Contextual shadow layers for SaaS UIs.

- `shadow-sm` (subtle element lift)
- `shadow-md` (card/base pop-out)
- `shadow-lg` (modal/dialog)
- `shadow-xl` (onboarding spotlight)
- `shadow-focus-ring`
- `elevation-dropdown`
- `elevation-tooltip`

- `elevation-modal`
-

6. Motion & Animation Tokens

Standardized timing and curves for micro-interactions.

- `motion-duration-fast` (e.g., 100ms)
 - `motion-duration-medium` (200–300ms)
 - `motion-duration-slow` (400–500ms)
 - `motion-ease-in`
 - `motion-ease-out`
 - `motion-ease-in-out`
 - `motion-bounce`
 - `motion-spring`
 - `motion-delay-short`
 - `motion-delay-long`
-

7. Opacity Tokens

Control visual hierarchy & disabled states.

- `opacity-none`
 - `opacity-low`
 - `opacity-medium`
 - `opacity-high`
 - `opacity-disabled`
 - `opacity-hover`
-

8. Component-Specific Semantic Tokens

Tokens that are scoped to a UI component or pattern.

Buttons

- `btn-bg-primary`
- `btn-bg-primary-hover`
- `btn-bg-primary-active`
- `btn-text-primary`
- `btn-border-primary`

- btn-shadow-primary
- btn-bg-secondary
- btn-bg-danger
- btn-bg-success

Forms

- form-bg-field
- form-bg-field-hover
- form-bg-field-disabled
- form-border-field
- form-border-focus
- form-border-error
- form-text-placeholder
- form-label-text

Tables

- table-bg-header
 - table-bg-row
 - table-bg-row-hover
 - table-border
 - table-text-header
 - table-text-cell
-

9. State & Feedback Tokens

Covering hover, active, focus, disabled, and status feedback.

- state-hover-bg
- state-hover-border
- state-hover-text
- state-active-bg
- state-active-border
- state-active-text
- state-focus-ring
- state-disabled-bg
- state-disabled-text
- status-success-bg
- status-success-text

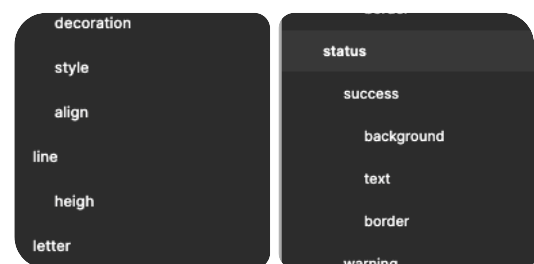
- status-warning-bg
 - status-warning-text
 - status-error-bg
 - status-error-text
 - status-info-bg
 - status-info-text
-

10. Data Visualization Tokens

For charts, graphs, and analytics dashboards.

- chart-color-series-1 through chart-color-series-12
 - chart-color-positive
 - chart-color-negative
 - chart-color-neutral
 - chart-color-highlight
 - chart-bg
 - chart-gridline
 - chart-axis-label
 - chart-tooltip-bg
 - chart-tooltip-text
-

If you want, I can also give you the **Tier 3 “component-instance” token mapping** where these semantic tokens are tied directly to **real SaaS components** like dashboards, modals, and onboarding flows. That would make this a complete **primitives → semantic → applied** hierarchy.



look at this tokens and check if I am doing it right

Yes — you're on the right track.

From what I see in your screenshots, your token setup follows the **correct layered approach**:

- **Core tokens** = raw primitives (colors, spacing, font families/sizes/weights, letter spacing, line height, shadows, z-index, opacity, borders, etc.).
- **Semantic tokens** = role-based mappings (e.g.,
`color.action.primary.background` → maps to a core blue;
`color.status.success.text` → maps to a core green, etc.).

That's exactly the **Tier 1 → Tier 2** structure.

What You're Doing Well

1. Clear separation of primitives vs semantics

- Core tokens are abstract and not tied to meaning.
- Semantic tokens are contextual and reference the core set.

2. Consistent token naming convention

- You're using `category → subcategory → property` (e.g., `color.action.primary.background`), which scales well.

3. Coverage of states and statuses

- Action states (primary, secondary, tertiary)
 - Statuses (success, warning, error, info)
 - Surfaces & overlays
-

Gaps & Possible Improvements

1. Missing Non-Color Semantic Tokens

Your semantic set is **only for colors** right now.

For a SaaS-level Tier 2, you'd ideally also have:

- **Typography semantic roles:**
`text.heading.lg`, `text.body.md`, `text.label.sm`, `text.mono.code`, etc.
→ These map to core font families, sizes, weights, line-heights.
- **Spacing semantic roles:**
`space.section.xl`, `space.card.md`, `space.form.sm`
→ These map to core spacing primitives.

- **Border & radius roles:**
radius.card , radius.input , border-width.focusRing , border-color.input.focus
 - **Shadow roles:**
shadow.dropdown , shadow.modal , shadow.toast → mapped from shadow.md , etc.
-

2. Interaction State Tokens

Instead of only static states (background/text/border), you can create **state tokens** for interactive components:

Example:

pgsql

 Copy code

```
color.action.primary.hover.bg
color.action.primary.hover.text
color.action.primary.active.bg
color.action.primary.disabled.bg
color.action.primary.disabled.text
```

This makes hover/active/disabled consistent across components.

3. Surface Hierarchy

You have surface and overlay , but SaaS dashboards often need a **layer hierarchy**:

pgsql

 Copy code

```
surface.base.bg
surface.section.bg
surface.card.bg
surface.table.header.bg
overlay.scrim.bg
overlay.modal.bg
```

4. Data Visualization Tokens

If your SaaS has analytics or charts, define semantic chart tokens:

```
chart.series.1  
chart.series.2  
chart.series.positive  
chart.series.negative  
chart.axis.label  
chart.gridline  
chart.tooltip.bg
```

5. Motion Tokens

Not common in basic systems, but for advanced SaaS:

 Copy code

```
motion.duration.fast  
motion.duration.medium  
motion.duration.slow  
motion.easing.standard  
motion.easing.bounce
```

✅ Summary:

Your structure is correct — Core for raw values, Semantic for contextual meaning. The main upgrade would be to **extend semantics beyond color** into typography, spacing, borders, shadows, and motion, and add **interaction state variants** so they're consistent everywhere.

If you want, I can map **your exact current tokens** into a **Tier 3 applied-component layer** so they directly connect to things like buttons, inputs, tables, and modals. That would make it fully production-ready for complex SaaS.

     ...

with that provide the full list

Got it — here's the **full, advanced Tier 2 semantic token list** built on top of what you already have in your screenshots, expanded to cover **all major SaaS contexts**.

This assumes your **Core** tokens (Tier 1) already have raw colors, spacing, fonts, etc., and here we map them into **meaningful, role-based tokens**.

1. Color Tokens (Semantic Layer)

Action Colors

CSS

 Copy code

```
color.action.primary.background
color.action.primary.text
color.action.primary.border
color.action.primary.hover.background
color.action.primary.hover.text
color.action.primary.hover.border
color.action.primary.active.background
color.action.primary.active.text
color.action.primary.active.border
color.action.primary.disabled.background
color.action.primary.disabled.text
color.action.primary.disabled.border

color.action.secondary.background
color.action.secondary.text
color.action.secondary.border
color.action.secondary.hover.background
color.action.secondary.hover.text
color.action.secondary.hover.border
color.action.secondary.active.background
color.action.secondary.active.text
color.action.secondary.active.border
color.action.secondary.disabled.background
color.action.secondary.disabled.text
color.action.secondary.disabled.border

color.action.tertiary.background
color.action.tertiary.text
color.action.tertiary.border
color.action.tertiary.hover.background
color.action.tertiary.hover.text
```

```
color.action.tertiary.hover.border  
color.action.tertiary.active.background  
color.action.tertiary.active.text  
color.action.tertiary.active.border  
color.action.tertiary.disabled.background  
color.action.tertiary.disabled.text  
color.action.tertiary.disabled.border
```

Status Colors

CSS

 Copy code

```
color.status.success.background  
color.status.success.text  
color.status.success.border  
color.status.success.icon
```

```
color.status.warning.background  
color.status.warning.text  
color.status.warning.border  
color.status.warning.icon
```

```
color.status.error.background  
color.status.error.text  
color.status.error.border  
color.status.error.icon
```

```
color.status.info.background  
color.status.info.text  
color.status.info.border  
color.status.info.icon
```

Surface & Layout Colors

CSS


 Copy code

```
color.surface.base.background  
color.surface.section.background  
color.surface.card.background  
color.surface.table.background
```

```
color.surface.table.header.background  
color.surface.navigation.background  
color.surface.sidebar.background  
color.surface.footer.background
```

Overlay & Utility Colors

css

 Copy code

```
color.overlay.scrim.background  
color.overlay.modal.background  
color.overlay.tooltip.background  
color.overlay.popover.background  
color.overlay.backdrop.background
```

```
color.utility.focusRing  
color.utility.selection  
color.utility.highlight
```

2. Typography Semantic Tokens

pgsql

 Copy code

```
text.heading.xl  
text.heading.lg  
text.heading.md  
text.heading.sm  
text.heading.xs
```

```
text.body.lg  
text.body.md  
text.body.sm  
text.body.xs
```

```
text.label.lg  
text.label.md  
text.label.sm
```

```
text.code.inline  
text.code.block
```

`text.placeholder`
`text.link`
`text.link.hover`
`text.error`

3. Spacing Semantic Tokens

cpp

 Copy code

`space.section.xl`
`space.section.lg`
`space.section.md`
`space.section.sm`

`space.card.lg`
`space.card.md`
`space.card.sm`

`space.form.lg`
`space.form.md`
`space.form.sm`

`space.list.lg`
`space.list.md`
`space.list.sm`

`space.inline.xl`
`space.inline.lg`
`space.inline.md`
`space.inline.sm`

`space.stack.xl`
`space.stack.lg`
`space.stack.md`
`space.stack.sm`

4. Border & Radius Semantic Tokens

CSS

```
radius.card  
radius.input  
radius.button  
radius.modal  
radius.full
```

```
border-width.hairline  
border-width.thin  
border-width.medium  
border-width.thick
```

```
border.color.default  
border.color.focus  
border.color.active  
border.color.disabled  
border.color.error
```


5. Shadow & Elevation Tokens

 Copy code

```
shadow.card  
shadow.dropdown  
shadow.tooltip  
shadow.modal  
shadow.toast  
shadow.navigation  
shadow.focusRing
```

6. Opacity Tokens

CSS

 Copy code

```
opacity.hover  
opacity.disabled  
opacity.overlay  
opacity.focus
```

`opacity.faint`
`opacity.subtle`

7. Motion Tokens

arduino

 Copy code

`motion.duration.xfast`
`motion.duration.fast`
`motion.duration.medium`
`motion.duration.slow`
`motion.duration.xslow`

`motion.easing.standard`
`motion.easing.accelerate`
`motion.easing.decelerate`
`motion.easing.spring`
`motion.easing.bounce`

`motion.delay.short`
`motion.delay.medium`
`motion.delay.long`

8. Data Visualization Tokens

arduino

 Copy code

`chart.series.1`
`chart.series.2`
`chart.series.3`
`chart.series.4`
`chart.series.5`
`chart.series.6`
`chart.series.7`
`chart.series.8`
`chart.series.positive`
`chart.series.negative`
`chart.series.neutral`
`chart.series.highlight`
`chart.axis.label`

```
chart.axis.line  
chart.gridline  
chart.tooltip.background  
chart.tooltip.text
```

9. Component-Specific Semantic Tokens

Buttons

CSS

 Copy code

```
button.primary.bg  
button.primary.text  
button.primary.border  
button.primary.hover.bg  
button.primary.hover.text  
button.primary.hover.border  
button.primary.active.bg  
button.primary.active.text  
button.primary.disabled.bg  
button.primary.disabled.text  
  
button.secondary.bg  
button.secondary.text  
button.secondary.border  
...
```

Form Inputs

pgsql

 Copy code

```
input.bg  
input.text  
input.border  
input.placeholder  
input.focus.border  
input.error.border  
input.disabled.bg  
input.disabled.text
```

Tables

pgsql

 Copy code

```
table.header.bg
table.header.text
table.row.bg
table.row.alt.bg
table.row.hover.bg
table.cell.text
table.border
```

Modals & Overlays

pgsql

 Copy code

```
modal.bg
modal.border
modal.shadow
modal.header.text
modal.body.text
modal.footer.bg
overlay.scrim.bg
```

If you want, I can **map each of these Tier 2 tokens directly to your current Core tokens** from the screenshots you sent — that way, you'll have a **ready-to-implement token JSON** linking primitives to semantics without guessing. That would make it instantly usable in your SaaS design system.



i think we could do more on Surface & Layout Colors like borders etc?

Yes — 100%.

Right now, your **Surface & Layout Colors** cover only backgrounds, but in a **complex SaaS design system**, surfaces should also define **borders, shadows, and text roles** tied to those surfaces.

That way, when you update a surface style (e.g., switch from light to dark theme, add more contrast), **everything related to that surface updates in sync.**

Expanded Surface & Layout Color Tokens

Base & App Structure

csharp

 Copy code

```
color.surface.base.background  
color.surface.base.border  
color.surface.base.text  
color.surface.base.icon  
color.surface.base.shadow
```

Section / Panel

css

 Copy code

```
color.surface.section.background  
color.surface.section.border  
color.surface.section.text  
color.surface.section.icon  
color.surface.section.shadow
```

Cards

css

 Copy code

```
color.surface.card.background  
color.surface.card.border  
color.surface.card.text  
color.surface.card.icon  
color.surface.card.shadow
```

Tables

pgsql

 Copy code

```
color.surface.table.background  
color.surface.table.border  
color.surface.table.text  
color.surface.table.icon  
color.surface.table.shadow
```

```
color.surface.table.header.background  
color.surface.table.header.border  
color.surface.table.header.text  
color.surface.table.header.icon  
color.surface.table.header.shadow
```

```
color.surface.table.row.background  
color.surface.table.row.alt.background  
color.surface.table.row.hover.background  
color.surface.table.row.border  
color.surface.table.row.text
```

Navigation / Sidebar / Footer

css

 Copy code

```
color.surface.navigation.background  
color.surface.navigation.border  
color.surface.navigation.text  
color.surface.navigation.icon  
color.surface.navigation.shadow
```

```
color.surface.sidebar.background  
color.surface.sidebar.border  
color.surface.sidebar.text  
color.surface.sidebar.icon  
color.surface.sidebar.shadow
```

```
color.surface.footer.background  
color.surface.footer.border  
color.surface.footer.text  
color.surface.footer.icon  
color.surface.footer.shadow
```

Forms & Inputs

CSS

 Copy code

```
color.surface.form.background
color.surface.form.border
color.surface.form.text
color.surface.form.icon
color.surface.form.shadow
```

```
color.surface.input.background
color.surface.input.border
color.surface.input.text
color.surface.input.icon
color.surface.input.shadow
```

Overlays & Modals

CSS

 Copy code

```
color.surface.overlay.background
color.surface.overlay.border
color.surface.overlay.text
color.surface.overlay.icon
color.surface.overlay.shadow
```

```
color.surface.modal.background
color.surface.modal.border
color.surface.modal.text
color.surface.modal.icon
color.surface.modal.shadow
```

Utility Surfaces

CSS

 Copy code

```
color.surface.highlight.background
color.surface.highlight.border
color.surface.highlight.text
```

`color.surface.highlight.icon`
`color.surface.highlight.shadow`

`color.surface.selection.background`
`color.surface.selection.border`
`color.surface.selection.text`
`color.surface.selection.icon`

This approach:

- Gives **every surface a full visual context** (bg, border, text, icon, shadow).
 - Makes **theme switching easier** — a card in dark mode might have no border but a heavy shadow; in light mode, it might use a subtle border instead.
 - Improves **UI consistency** — no more ad-hoc colors for component edges and shadows.
-

If you want, I can **merge this expanded Surface/Border set into the full Tier 2 token master list** I gave you before, so you'll have **one unified, production-ready token map**. That way, your system will be fully covered for SaaS at scale.



but this is a tier 2 or 3?

What I just gave you for **Surface & Layout** (with backgrounds, borders, text, icons, shadows per surface type) is still **Tier 2** — not Tier 3.

Here's why:

Tier 1 → Tier 2 → Tier 3

- **Tier 1 (Core / Primitives)**
 - Pure raw values with no meaning.
 - Example: `color.gray.100 = #F8F9FA` , `spacing.4 = 16px` ,
`font.size.md = 16px` .





































- **Tier 2 (Semantic / Role-based)** ✅ *Your current stage*
 - Abstract roles that map to Tier 1 values but are **not tied to a single component**.
 - Example:
 - `color.surface.card.background` → `color.gray.100`
 - `color.surface.card.border` → `color.gray.300`
 - `color.surface.card.text` → `color.gray.900`
 - Still reusable across components, not hardcoded for "Button" or "Modal" only.
 - **Tier 3 (Component-specific / Instance)**
 - Tokens **scoped directly to a specific component** or variant.
 - Example:
 - `button.primary.background` → `color.action.primary.background`
 - `table.header.background` →
`color.surface.table.header.background`
 - `modal.footer.border` → `color.surface.modal.border`
-

✅ Conclusion:

The expanded **Surface & Layout** set is **Tier 2** because it's **describing the role of the surface** (e.g., card, table, navigation) without locking it to a specific component implementation.

If you want, I can now **merge this expanded Surface/Border set into your full Tier 2 master list**, and then also **show you the Tier 3 mapping** so you can see how they cascade into components. That way, your SaaS token architecture will be fully layered and production-ready.



Name	Pay Advantage Light	Pay Advantage Dark	Marshal Freeman	
 inverse	 gray/black	 gray/100	 brand/blue/900	
color / surface / overlay				
 light	 gray/100	 gray/black	 brand/gray/100	
 dark	 gray/900	 gray/100	 brand/blue/900	
color / surface / level				
 0	 gray/100	 gray/black	 brand/gray/100	
 1	 gray/200	 gray/900	 brand/gray/200	
 2	 gray/300	 gray/800	 brand/gray/300	
 3	 gray/400	 gray/700	 brand/gray/400	
 4	 gray/500	 gray/600	 brand/gray/500	
 5	 gray/600	 gray/500	 brand/gray/600	

so this is not to good, i should change for waht you gave me now?

Yeah — what you have here works, but it's **too flat and color-value-driven** for a truly scalable SaaS design system.

Right now:

- Your “surface” tokens (`inverse` , `overlay.light` , `overlay.dark` , `level.0–5`) are basically **direct color mappings** (Tier 1.5) rather than fully semantic Tier 2 roles.
- They're useful for themes, but they don't give enough **context** about *where* and *how* a surface is used.
- “Level” tokens are okay for elevation, but they lack **paired borders, text, icons, shadows** — so every component has to guess how to style itself on that surface.

Why I'd move to the expanded model I gave you:

1. **Semantic clarity** — Instead of `surface.level.3` , you'd have `surface.card.background` , `surface.card.border` , `surface.card.text` , which is instantly understandable to designers/devs.
2. **Theme flexibility** — You can remap all “card” surfaces in dark/light themes without touching individual components.
3. **Component readiness** — Each surface role comes with border/text/icon/shadow, so components don't need to guess.

4. Tier 2 compliance — This keeps surfaces **role-based** instead of "just colors in disguise."

How I'd restructure what you have:

Current Token	Better Tier 2 Equivalent	Notes
surface.inverse	surface.inverse.background	Also define .text , .border , .icon
surface.overlay.light	surface.overlay.light.background	Also define .border , .text , .shadow
surface.overlay.dark	surface.overlay.dark.background	Same as above
surface.level.0	surface.base.background	Define border/text/icon/shadow for base
surface.level.1	surface.section.background	Or other semantic layer name
surface.level.2	surface.card.background	With .border , .text , .shadow
surface.level.3	surface.table.background	Or whatever context fits
surface.level.4	surface.navigation.background	Or sidebar depending on use
surface.level.5	surface.modal.background	For high elevation overlays

If you want, I can **take this exact table**, keep your brand palettes, and **remap it to the expanded surface/border/text/shadow system** I described earlier.

That way, you won't lose the work you've done — but it'll be **Tier 2 semantic-ready** and much more future-proof.



provide the final and most complete for surfaces

Thought for 11s >

Awesome—here's a **final, production-grade Tier-2 "Surfaces" token set** for a complex SaaS.

It's fully semantic (no raw values), covers **background/border/text/icon/shadow**, plus **interactive states** and **common UI contexts**. Use these to map to your Tier-1 primitives per theme (light, dark, HC).

Naming

`color.surface.<context>.<property>[.<state>|.<variant>]`

`shadow.surface.<context>[.<variant>]`

Properties: background | border | border-strong | divider | text | icon

States (where it makes sense): hover | active | selected | disabled

Variants (optional): subtle | raised | outlined | muted | emphasis-* | contrast-high

A. App Foundations

CSS

 Copy code

`color.surface.canvas.background`

`color.surface.canvas.text`

`color.surface.canvas.icon`

`color.surface.canvas.divider`

`color.surface.base.background`

`color.surface.base.border`

`color.surface.base.text`

`color.surface.base.icon`

`color.surface.base.divider`

`color.surface.base.hover.background`

B. Sections & Panels

CSS

 Copy code

```
color.surface.section.background
color.surface.section.border
color.surface.section.text
color.surface.section.icon
color.surface.section.divider
color.surface.section.raised.background
color.surface.section.outlined.border
shadow.surface.section.raised
```

C. Cards & Wells

CSS

 Copy code

```
color.surface.card.background
color.surface.card.border
color.surface.card.border-strong
color.surface.card.text
color.surface.card.icon
color.surface.card.divider
color.surface.card.hover.background
color.surface.card.active.background
color.surface.card.selected.background
color.surface.card.disabled.background
color.surface.card.subtle.background
color.surface.card.raised.background
color.surface.card.outlined.border
shadow.surface.card.raised
```

SCSS

 Copy code

```
color.surface.well.background      // inset container
color.surface.well.border
color.surface.well.text
color.surface.well.icon
```

D. Navigation (Topbar / Sidebar / Footer / Tabbar)

SCSS

 Copy code`color.surface.navbar.background``color.surface.navbar.border``color.surface.navbar.text``color.surface.navbar.icon``shadow.surface.navbar``color.surface.sidebar.background``color.surface.sidebar.border``color.surface.sidebar.text``color.surface.sidebar.icon``color.surface.sidebar.active.background``color.surface.sidebar.selected.background``color.surface.footer.background``color.surface.footer.border``color.surface.footer.text``color.surface.footer.icon``color.surface.tabbar.background``color.surface.tabbar.border``color.surface.tabbar.text``color.surface.tabbar.icon``color.surface.tabbar.active.background``color.surface.tabbar.active.text``color.surface.tabbar.indicator` *// underline/active indicator*

E. Toolbars & Headers

CSS

 Copy code`color.surface.toolbar.background``color.surface.toolbar.border``color.surface.toolbar.text``color.surface.toolbar.icon``shadow.surface.toolbar`

F. Tables & Lists

pgsql

 Copy code

```
color.surface.table.background  
color.surface.table.border  
color.surface.table.text  
color.surface.table.icon  
color.surface.table.divider
```

```
color.surface.table.header.background  
color.surface.table.header.border  
color.surface.table.header.text  
color.surface.table.header.icon
```

```
color.surface.table.row.background  
color.surface.table.row.alt.background  
color.surface.table.row.hover.background  
color.surface.table.row.active.background  
color.surface.table.row.selected.background  
color.surface.table.row.border  
color.surface.table.cell.border
```

G. Forms & Inputs (surface roles, not component tokens)

css

 Copy code

```
color.surface.field.background  
color.surface.field.border  
color.surface.field.text  
color.surface.field.icon  
color.surface.field.placeholder  
color.surface.field.hover.background  
color.surface.field.focus.border  
color.surface.field.disabled.background  
color.surface.field.disabled.text  
color.surface.field.invalid.border  
color.surface.field.readonly.background
```

H. Overlays (Scrims, Popovers, Tooltips, Modals, Drawers)

CSS

 Copy code

```
color.surface.overlay.scrim.light.background  
color.surface.overlay.scrim.dark.background
```

```
color.surface.modal.background  
color.surface.modal.border  
color.surface.modal.text  
color.surface.modal.icon  
shadow.surface.modal
```

```
color.surface.drawer.background  
color.surface.drawer.border  
color.surface.drawer.text  
color.surface.drawer.icon  
shadow.surface.drawer
```

```
color.surface.popover.background  
color.surface.popover.border  
color.surface.popover.text  
color.surface.popover.icon  
shadow.surface.popover
```

```
color.surface.tooltip.background  
color.surface.tooltip.border  
color.surface.tooltip.text  
shadow.surface.tooltip
```

I. Feedback Surfaces (Toast, Banner, Inline)

CSS

 Copy code

```
color.surface.toast.background  
color.surface.toast.border  
color.surface.toast.text  
color.surface.toast.icon  
shadow.surface.toast
```

```
color.surface.banner.background  
color.surface.banner.border
```

```
color.surface.banner.text  
color.surface.banner.icon
```

J. Chips / Tags / Pills (as surfaces, not component tokens)

CSS

 Copy code

```
color.surface.pill.background  
color.surface.pill.border  
color.surface.pill.text  
color.surface.pill.icon  
color.surface.pill.selected.background
```

K. Selection / Highlight / Focus

SCSS

 Copy code

```
color.surface.selection.background // e.g., list selection, editor s  
color.surface.selection.border  
color.surface.highlight.background // search match / code highlight  
color.surface.highlight.border  
color.surface.focusRing // could live under utility; kept
```

L. Inverse & Contrast

CSS

 Copy code

```
color.surface.inverse.background  
color.surface.inverse.border  
color.surface.inverse.text  
color.surface.inverse.icon  
  
color.surface.contrast-high.background  
color.surface.contrast-high.border  
color.surface.contrast-high.text  
color.surface.contrast-high.icon
```

M. Data/Chart Containers (the surface around viz)

SCSS

 Copy code

```
color.surface.chart.background
color.surface.chart.border
color.surface.chart.text
color.surface.chart.icon
color.surface.chart.gridline
color.surface.chart.plot.background    // inner plotting area
```

N. Emphasis/Depth System (portable alternative to “level 0–5”)

Map these to different bg/border/shadow intensities per theme.

CSS

 Copy code

```
color.surface.emphasis.0.background
color.surface.emphasis.0.border
shadow.surface.emphasis.0

color.surface.emphasis.1.background
color.surface.emphasis.1.border
shadow.surface.emphasis.1

color.surface.emphasis.2.background
color.surface.emphasis.2.border
shadow.surface.emphasis.2

color.surface.emphasis.3.background
color.surface.emphasis.3.border
shadow.surface.emphasis.3
```

O. Dividers & Hairlines (global, but surface-aware)

CSS

 Copy code

```
color.surface.divider.default
color.surface.divider.muted
```

color.surface.divider.emphasis

How to migrate from your current surface.level.*

- Map level.0 → surface.base.*
- level.1 → surface.section.*
- level.2 → surface.card.*
- level.3 → surface.table.* (or another high-usage panel)
- level.4 → surface.navigation.* (or sidebar/topbar)
- level.5 → surface.modal.* (or overlay/drawer)

Add the **missing pairs**: .border , .text , .icon , .divider , and the **stateful** tokens where relevant.

If you want, say the word and I'll **generate a JSON skeleton** of these tokens, pre-filled with references to your existing **Pay Advantage Light/Dark** and **Marshal Freeman** palettes so you can drop it into your token pipeline.



but this looks like is tier 3 on componetns level?

It looks like Tier 3 at first glance because there's a lot of detail, but most of what I listed is still **Tier 2** — just *granular Tier 2*.

Quick recap: Tier 1 / 2 / 3

Tier	Scope	Example	What it maps to
Tier 1 – Core / Primitives	Raw, context-free values	gray/100 = #F8F9FA , spacing/4 = 16px	Literal values
Tier 2 – Semantic / Role-based	Abstract roles, reusable across multiple components	surface.card.background → gray/100	A "card surface" color used by any card

Tier	Scope	Example	What it maps to
Tier 3 – Component instance	Tokens scoped to a specific component or variant	button.primary.backgrou nd → surface.action.primary. background	Component- specific usage

Why the big list is still Tier 2

- They are **surface roles**, not components.
For example:
 - `surface.card.background` is a **surface type** that could be used in Cards, Modals, Empty States — it's not hard-coded to "Card Component" only.
 - `surface.navigation.background` is a **role** for navigation areas — could be used in sidebars, topbars, tab bars, etc.
- In Tier 3, you would map them into **specific component tokens** like `navbar.background` , `sidebar.background` , `tabbar.background` → all pointing to `surface.navigation.background` .

Where Tier 2 ends & Tier 3 begins

- Tier 2 stops at defining *roles* like `surface.modal.background` .
- Tier 3 starts when you say:
`modal.header.background = surface.modal.background`
`modal.footer.background = surface.modal.background`
`button.primary.bg = action.primary.bg`

So for your SaaS tokens

- The **expanded surface system** I gave you is **Tier 2** — it's just more descriptive than your current `level.0-5` .
- Your **Tier 3** would be a much smaller mapping layer that references these Tier 2 tokens for each actual component.

If you want, I can now take the **expanded Tier 2 surface list** and show you **what a clean Tier 3 mapping looks like** so you see the difference clearly and avoid overstuffing Tier 2

with true component tokens.

That way you'll have **Core → Surface Roles → Components** cleanly layered.



i have a acontier or diffrente type of card that could use diffrente backgrounds?

Exactly — and that's one of the main reasons to keep those **surface tokens** at **Tier 2** instead of jumping straight into component-specific Tier 3.

How it works

Let's say you have two types of "card" in your SaaS:

- **Account Card** → Might use a branded, high-emphasis background.
- **Default Info Card** → Uses a neutral, subtle background.

If you do it **properly with Tier 2**, you'd have:

yaml

 Copy code

```
color.surface.card.background.default = gray/100
color.surface.card.background.emphasis = brand/blue/100
color.surface.card.border = gray/300
color.surface.card.text = gray/900
```

Then in Tier 3 (component layer):

yaml

 Copy code

```
component.accountCard.background = surface.card.background.emphasis
component.infoCard.background = surface.card.background.default
```

Benefits:

- If you change `surface.card.background.emphasis` (e.g., dark mode, rebrand), *all* Account Cards update automatically.
- Components **never use raw colors** — they only use their mapped surface role.

- You can have **as many variants** as needed (`subtle` , `raised` , `outlined` , `muted`) without bloating Tier 3.
-

Recommendation

- In **Tier 2**: Define the **role + variant**:
`surface.card.background.subtle`
`surface.card.background.emphasis`
`surface.card.background.raised`
 - In **Tier 3**: Map **specific components** to the right variant:
`component.accountCard.bg = surface.card.background.emphasis`
`component.billingCard.bg = surface.card.background.subtle`
-

If you want, I can **update the final Tier 2 surface list** I gave you so **every major surface role** supports `.default` , `.emphasis` , `.subtle` , etc., making it easier to handle different account card styles without jumping into hardcoded component tokens.



from A to) explain the use of each one

Got it — let's go through the **A to N "Surface" list** I gave you earlier and explain **exactly when and why you'd use each**, so you can tell what's truly Tier 2 and how it applies in a SaaS context.

A. App Foundations

Purpose: These are your "global canvas" and "base" surfaces — the backdrop for everything else.

- `canvas` → The *root background* of the application (under everything). Rarely changes except with themes (light/dark/high-contrast).
- `base` → The main working surface for content. Think of it as the default background for most areas that aren't panels or cards.

- **Why separate them?** It lets you do things like have `canvas` slightly darker or textured, while `base` stays plain and high-contrast for content.
-

B. Sections & Panels

Purpose: Grouped areas that contain content but aren't as visually isolated as cards.

- `section` → For dashboard sections, report areas, or split panels that need light separation from `base`.
 - Variants like `.raised` or `.outlined` can give subtle elevation or separation without fully switching to a card style.
-

C. Cards & Wells

Purpose: Isolated content blocks with clear boundaries.

- `card` → For typical cards used in dashboards, analytics panels, product tiles.
 - Variants:
 - `.subtle` → almost blending into base for minimal UI
 - `.emphasis` → more vibrant for important content (e.g., featured account card)
 - `.raised` → with shadow to pull it forward
 - `.outlined` → border instead of shadow
 - `well` → Inset areas for form grouping or "callout" boxes inside a card.
-

D. Navigation

Purpose: All navigational regions get their own surface so they can be themed independently.

- `navbar` → Top navigation bar or app header.
- `sidebar` → Left/right navigation side panels.
- `footer` → App footer or bottom toolbar.
- `tabbar` → Horizontal tab row at the top or bottom of a section.

Why separate? Dark sidebars with light content areas are a common pattern in SaaS; these tokens make that possible without hacking colors per component.

E. Toolbars & Headers

Purpose: Functional bars inside sections, cards, or panels.

- `toolbar` → For icon/action bars above tables, inside modals, or in detail views.
 - You might darken/lighten these relative to their parent surface to create visual hierarchy.
-

F. Tables & Lists

Purpose: Structured data display surfaces.

- `table` → For the entire table container.
 - `table.header` → Header row background, usually higher-contrast.
 - `table.row` → Base row background; `.alt` for zebra striping, `.hover` for mouseover highlight, `.selected` for user selection.
 - `list` could also be included here for vertical data lists, using similar background/hover/selected roles.
-

G. Forms & Inputs

Purpose: Defines backgrounds and borders for form fields without being tied to a specific input component.

- `field` → General field container (input, textarea, select).
 - Variants for states: `.focus`, `.disabled`, `.invalid`, `.readonly`.
 - **Why in surfaces?** Because they're mini-surfaces inside the UI, and theming them consistently across input types is critical.
-

H. Overlays

Purpose: Layers that sit above the base app.

- `overlay.scrim` → Semi-transparent layer behind modals/drawers to dim the app.
- `modal` → Central floating content box.
- `drawer` → Side panel that slides in.
- `popover` → Small contextual bubble (menus, dropdowns).
- `tooltip` → Small label popover.

Each has background/border/text/icon/shadow so they're readable against the scrim.

I. Feedback Surfaces

Purpose: Temporary or persistent UI messaging areas.

- `toast` → Small popup message that appears above content.
- `banner` → Persistent strip at the top or bottom for status info.

These need distinct colors to stand out without clashing.

J. Chips / Tags / Pills

Purpose: Small, inline surfaces used for categorization or filtering.

- `pill` → Rounded background element around text/icon, can be `.selected`, `.emphasis`, `.muted`.
-

K. Selection / Highlight / Focus

Purpose: State-driven surface roles for selection or highlighting.

- `selection` → For list/table selection highlight.
 - `highlight` → For search results, code highlights, or user attention callouts.
 - `focusRing` → Outline around focused elements (could be under `utility` too).
-

L. Inverse & Contrast

Purpose: Color-inverted or high-contrast modes.

- `inverse` → Swaps foreground/background contrast for specific contexts (e.g., dark header on light theme).
 - `contrast-high` → Ultra-high-contrast mode for accessibility.
-

M. Data/Chart Containers

Purpose: Visualisation-specific surfaces.

- `chart` → Background for chart container.
 - `chart.plot` → Inner plotting area background.
 - `chart.gridline` → Lines dividing chart sections.
-

N. Emphasis / Depth System

Purpose: Abstracted “depth” levels without tying to a specific component.

- `emphasis.0-3` → Gradually stronger backgrounds/borders/shadows for layering.
- These can be mapped from your old `surface.level.0-5` but with a clearer hierarchy.

If you want, I can now **merge this explanation with your palette** and give you a **Pay Advantage Light/Dark + Marshal Freeman** mapping so you see how each role plays out in a real theme. That way you can replace your current `level.0-5` table without losing the multi-theme capability.



this app foundations is for mobile app and the webapp as well?

Yes — **App Foundations** applies to both **mobile apps** and **web apps**, because it’s the root visual structure for any product surface.

The only difference is **how much of each foundation surface is visible** and **what elements sit on top of it**.

How it works in both contexts

canvas

- **Web App** → The absolute bottom layer of the UI (like the `html` or `body` background).
Often only visible at the edges or behind scroll areas.
- **Mobile App** → The root background of the main view. Even if the app uses lots of full-bleed surfaces, `canvas` ensures there’s a fallback color/texture behind them.

base

- **Web App** → The main working area background where most content lives. Could be slightly different from `canvas` for subtle depth (e.g., `canvas` = light gray, `base` = white).
 - **Mobile App** → The default background for scrollable content areas in views/screens. Often the first surface after the status/navigation bars.
-

Why it's cross-platform

1. **Design consistency** — Using the same foundational roles means your brand feels consistent across web and mobile.
 2. **Theming flexibility** — Dark/light/high-contrast modes are easier when `canvas` and `base` can change independently.
 3. **Separation of responsibility** —
 - `canvas` = global frame
 - `base` = main workspace/content layer
-

Example

If you wanted to introduce a "soft shadow" effect for your entire app:

- On **web**, you might make `canvas` slightly darker than `base` so cards and panels stand out.
 - On **mobile**, you could keep `canvas` hidden most of the time but still use it under system overlays (e.g., modals, drawers) to create depth.
-

If you want, I can give you a **diagram showing the surface hierarchy** from `canvas` → `base` → **panels/cards** → **overlays**, for both web and mobile views, so it's visually clear where these tokens apply. That'll make it easier to explain to developers too.



from A to) provide one component for each use like
`canvas`, `base` etc

Got it — I'll go **A to N** and give you **one clear example component** for each surface role, so you can see how they map in a real SaaS UI.

This keeps it **Tier 2** (role-based), but tied to something concrete so you can visualize it.

A. App Foundations

- `canvas` → The global app background (e.g., the white/gray area behind the entire dashboard).

Example component: **Login Page background.**

- `base` → The main working area background.

Example component: **Main dashboard content area** (where widgets and tables live).

B. Sections & Panels

- `section` → A grouped block of related content inside the base area.

Example component: **Analytics summary section** in a dashboard.

C. Cards & Wells

- `card` → Isolated content container.

Example component: **Account card** showing balance and transactions.

- `well` → Inset area with subtle styling.

Example component: **Form field group background** inside a settings page.

D. Navigation

- `navbar` → Top navigation bar.

Example component: **Web app header with search & profile menu.**

- `sidebar` → Vertical navigation area.

Example component: **Main left navigation with icons and links.**

- `footer` → Footer strip.

Example component: **Admin console footer with version number.**

- `tabbar` → Horizontal tab navigation.

Example component: **Mobile bottom tab navigation.**

E. Toolbars & Headers

- `toolbar` → Horizontal bar with actions inside a section or card.
Example component: Table toolbar with filters and export button.
-

F. Tables & Lists

- `table` → Table container.
Example component: Transactions table background.
 - `table.header` → Table header row.
Example component: Column header row with sorting controls.
 - `table.row` → Table row background.
Example component: Individual transaction row in the table.
-

G. Forms & Inputs

- `field` → General input surface.
Example component: Text input background in a sign-up form.
-

H. Overlays

- `overlay.scrim` → Dim layer behind a modal.
Example component: Background dim when editing a record in a modal.
 - `modal` → Floating dialog.
Example component: "Add new payment method" popup.
 - `drawer` → Sliding panel from side.
Example component: Right-side settings drawer.
 - `popover` → Contextual bubble.
Example component: User avatar menu dropdown.
 - `tooltip` → Small info bubble.
Example component: Tooltip on hover over "?" icon.
-

I. Feedback Surfaces

- `toast` → Temporary message.
Example component: "Saved successfully" toast at bottom right.

- `banner` → Persistent info strip.

Example component: **Subscription expired banner** at top of app.

J. Chips / Tags / Pills

- `pill` → Small inline container.

Example component: **"Active" status tag** in a user table.

K. Selection / Highlight / Focus

- `selection` → Highlight for selected items.

Example component: **Selected row background** in a data table.

- `highlight` → Marked text or area.

Example component: **Search result highlight** in a document viewer.

- `focusRing` → Outline around focused element.

Example component: **Blue ring around a focused input.**

L. Inverse & Contrast

- `inverse` → Inverted color scheme.

Example component: **White text on dark navbar background.**

- `contrast-high` → High contrast for accessibility.

Example component: **High-contrast theme toggle for visually impaired users.**

M. Data/Chart Containers

- `chart` → Chart container background.

Example component: **Bar chart panel** in analytics dashboard.

- `chart.plot` → Inner plotting area background.

Example component: **Grid background inside the chart.**

N. Emphasis / Depth System

- `emphasis.0-3` → Background intensity/depth levels.

Example component: **Dashboard widget stacking where each layer is slightly darker/lighter.**

If you want, I can make a **two-layer mapping table** that shows:

Tier 2 Surface Role → Example Component (Tier 3 mapping) → Light/Dark Theme Core Token Reference.

That would basically be your **design token Bible** for surfaces in SaaS.



i think that base and canvas could be one instead two
because they are almost thte same

You're right — in a lot of SaaS UIs, `canvas` and `base` end up being identical in color and usage, so splitting them can feel redundant.

The only reason to keep both is if you want **future flexibility** for visual separation, like:

- `canvas` slightly tinted or textured while
- `base` stays clean for the working area.

If you **don't** see a real design or theming use case for that, you can safely merge them into **one foundational token**:

Updated App Foundation

csharp

 Copy code

```
color.surface.base.background  
color.surface.base.border  
color.surface.base.text  
color.surface.base.icon  
color.surface.base.divider
```

- **Mobile app** → This would be the main screen background.
- **Web app** → This would be the main content area background and also act as the global backdrop.

Pros of merging

- **Simpler token set** — fewer concepts for designers/devs to learn.
 - **Less mapping in Tier 3** — you don't need to decide when to use `canvas` vs `base`.
 - **Keeps naming cleaner** — "base" becomes the go-to default for anything without a specific surface role.
-

Cons of merging

- If in the future you want a **different background behind the base** (like a patterned frame, or a contrasting app shell), you'll have to reintroduce `canvas`.
-

If you want, I can **restructure the full A–N surface list** so `canvas` is removed and `base` is the single global foundation. That way it's cleaner now, but still easy to expand later if your design evolves.



instead of `section` could be a container that wraps other stuff?

Yes — renaming `section` to `container` actually makes a lot of sense for a SaaS design system.

Why "container" works better

- **More flexible meaning** — A "container" can wrap any grouped content: dashboards, forms, feature areas, etc.
 - **Cross-platform clarity** — On mobile and web, "container" reads as a *layout element*, while "section" can sound like just a heading/content block.
 - **Keeps Tier 2 neutral** — You're not locking it to a visual pattern like a "card" or a "panel" — it's just a role that can be styled.
-

Updated Tier 2 token

CSS

 Copy code

```
color.surface.container.background
color.surface.container.border
color.surface.container.text
color.surface.container.icon
color.surface.container.divider
color.surface.container.raised.background
color.surface.container.outlined.border
shadow.surface.container.raised
```

How it's used

- **Web app:** Could be the main wrapper for a dashboard grid, wrapping multiple cards.
 - **Mobile app:** Could be a scrollable content area that groups related blocks.
 - **Variant roles:**
 - `.raised` → Brings the container forward with shadow
 - `.outlined` → Bordered, no shadow
 - `.subtle` → Almost blends with base
-

If you want, I can **rebuild the full surface role list** with:

- `canvas` **merged into** `base`
- `section` **renamed to** `container`
- Everything structured so you have the **cleanest possible Tier 2 surface system**.



what difference between outlined or border?

The difference is **subtle but important** — it's about **how the border is treated in your token system and the design intent**.

1. Border

- A **border** token is just a **color role** (Tier 1 or Tier 2) for the edge of a surface.

- Any surface can have a `border` property, even if it's *not meant* to be a visual style variant.
 - Example:
 - `surface.card.border = gray/300` → This is the default edge color for all cards.
 - It's a **property** of a surface, not a **style mode**.
-

2. Outlined

- An **outlined** variant is a **style decision** where the *border is the primary visual separator*, instead of elevation or shadow.
 - Outlined containers usually have:
 - Stronger border contrast
 - No shadow
 - Same background as the base or container
 - Example:
 - `surface.card.outlined.border = gray/400`
 - `surface.card.outlined.background = base.background`
 - It's a **variant** in Tier 2 that changes how the component feels (more flat and framed vs raised).
-

Think of it like this:

- **Border** = The *color role* for the edge (every surface can have it).
 - **Outlined** = A *variant style* where the border is the main separation and the shadow is removed.
-

Example in SaaS UI

- **Card with border (default)** → White background, subtle gray edge, small shadow.
 - **Card outlined variant** → White background, thicker dark gray border, no shadow (used in dense layouts or for minimal styling).
-

If you want, I can mark **which surface roles in your Tier 2 list** should *also* have an `outlined` variant so you don't overcomplicate and add it everywhere unnecessarily.

That way outlined becomes an intentional design choice, not something every token needs.



modern desing not use so much shadow anymore on my believe, maybe on hovers but is not like on the past? I am correct or wrong?

You're correct — in modern SaaS and product design, **shadows are used far more sparingly** than in the past, and mostly for **interaction or subtle depth cues**, not as a default separation method.

Why shadows are used less now

1. **Clean, flat aesthetic** → Modern UI trends lean toward **minimal, crisp edges** and lighter visual noise.
2. **Performance & accessibility** → Heavy shadows can create contrast issues, especially in dark mode, and can look inconsistent across screens.
3. **Layering alternatives** → Designers now rely more on:
 - **Color contrast** (light/dark surface levels)
 - **Borders & dividers**
 - **Subtle background shifts**
4. **Neumorphism backlash** → Early neumorphism overused shadows, making things look soft but causing usability issues. Many teams moved back to flatter styles.

When shadows *are* used in modern design

- **Hover states** → Slight shadow lift to signal interactivity.
 - **Focused/highlighted elements** → Emphasizing active cards or popovers.
 - **Overlays/modals** → To separate them from the background and imply elevation.
 - **Floating action buttons** → In mobile/web hybrid UIs, a subtle shadow can make them stand out.
-

Example: Modern SaaS visual hierarchy

Layer	Modern approach	Old approach
Base surfaces	Flat with slight color step	Often had subtle default shadow
Interactive hover	Adds a small shadow or color shift	Strong shadow or big lift
Modals/popovers	Strong but soft shadow	Multiple-layer shadow with high blur
Cards	Often flat or outlined	Default elevation with shadow

So yes — your instinct is in line with current design systems:
Shadows are now a **contextual tool** rather than a **default decoration**.

If you want, I can adjust your **Tier 2 surface tokens** so shadows are **optional and state-driven** (e.g., only `hover` , `active` , `overlay`) instead of being a default property for every surface. That way your design system stays modern, but you still have elevation tokens when you need them.



make the fully final surface tokens with this new
approach

Perfect—here’s the **final Tier-2 “Surfaces” token set** with your updated approach:

- **Canvas merged into Base**
- **“Section” renamed to “Container”**
- **Flat by default** (borders/contrast first), **shadows only where needed** (hover + true overlays)

Use these as **semantic roles** mapping to Tier-1 primitives per theme (light/dark/HC).

Surface tokens (Tier-2, semantic)

0) Conventions

- **Pattern:** `color.surface.<role>.<property>[.<state>|.<variant>]`
- **Properties:** `background` | `border` | `border-strong` | `divider` | `text` | `icon`
- **States:** `hover` | `active` | `selected` | `disabled` | `focus`
- **Variants (optional):** `subtle` | `raised` | `outlined` | `emphasis.[0-3]`

Shadows are **optional** and limited to:

- `shadow.surface.overlay` (modal/drawer/popover/tooltip)
- `shadow.surface.hover.card` (optional hover lift)
- `shadow.surface.hover.interactive` (generic small lift)

A) App foundation (canvas merged)

csharp

 Copy code

```
color.surface.base.background
color.surface.base.border
color.surface.base.divider
color.surface.base.text
color.surface.base.icon
```

B) Containers (formerly "section")

scss

 Copy code

```
color.surface.container.background
color.surface.container.border
color.surface.container.border-strong
color.surface.container.divider
color.surface.container.text
color.surface.container.icon

color.surface.container.subtle.background // near-base; dense layouts
color.surface.container.outlined.border // no shadow, framed look

// Optional emphasis steps via color, not shadow:
color.surface.container.emphasis.0.background
```

```
color.surface.container.emphasis.1.background  
color.surface.container.emphasis.2.background
```

C) Cards & Wells (flat by default)

SCSS

 Copy code

```
color.surface.card.background  
color.surface.card.border  
color.surface.card.border-strong  
color.surface.card.divider  
color.surface.card.text  
color.surface.card.icon  
  
color.surface.card.hover.background           // slight color step on hove  
color.surface.card.active.background  
color.surface.card.selected.background  
color.surface.card.disabled.background  
color.surface.card.outlined.border  
  
color.surface.well.background                 // inset group inside forms/  
color.surface.well.border  
color.surface.well.text  
color.surface.well.icon  
  
// Optional, modern hover shadow (keep subtle):  
shadow.surface.hover.card
```

D) Navigation (topbar / sidebar / footer / tabs)

SCSS

 Copy code

```
color.surface.navbar.background  
color.surface.navbar.border  
color.surface.navbar.text  
color.surface.navbar.icon  
  
color.surface.sidebar.background  
color.surface.sidebar.border  
color.surface.sidebar.text
```

```
color.surface.sidebar.icon  
color.surface.sidebar.active.background  
color.surface.sidebar.selected.background
```

```
color.surface.footer.background  
color.surface.footer.border  
color.surface.footer.text  
color.surface.footer.icon
```

```
color.surface.tabbar.background  
color.surface.tabbar.border  
color.surface.tabbar.text  
color.surface.tabbar.icon  
color.surface.tabbar.active.background  
color.surface.tabbar.active.text  
color.surface.tabbar.indicator
```

// underline / active marker

E) Toolbars & Headers (inside areas)

CSS

 Copy code

```
color.surface.toolbar.background  
color.surface.toolbar.border  
color.surface.toolbar.text  
color.surface.toolbar.icon
```

F) Tables & Lists

pgsql

 Copy code

```
color.surface.table.background  
color.surface.table.border  
color.surface.table.divider  
color.surface.table.text  
color.surface.table.icon
```

```
color.surface.table.header.background  
color.surface.table.header.border  
color.surface.table.header.text  
color.surface.table.header.icon
```

```
color.surface.table.row.background  
color.surface.table.row.alt.background  
color.surface.table.row.hover.background  
color.surface.table.row.selected.background  
color.surface.table.row.active.background  
color.surface.table.row.border  
color.surface.table.cell.border
```

G) Forms & Fields (surface roles, not component props)

CSS

 Copy code

```
color.surface.field.background  
color.surface.field.border  
color.surface.field.text  
color.surface.field.icon  
color.surface.field.placeholder  
  
color.surface.field.hover.background  
color.surface.field.focus.border  
color.surface.field.disabled.background  
color.surface.field.disabled.text  
color.surface.field.invalid.border  
color.surface.field.readonly.background
```

H) Overlays (true elevation → allowed shadow)

SCSS

 Copy code

```
color.surface.overlay.scrim.light.background  
color.surface.overlay.scrim.dark.background  
  
color.surface.modal.background  
color.surface.modal.border  
color.surface.modal.text  
color.surface.modal.icon  
  
color.surface.drawer.background  
color.surface.drawer.border
```

```
color.surface.drawer.text  
color.surface.drawer.icon
```

```
color.surface.popover.background  
color.surface.popover.border  
color.surface.popover.text  
color.surface.popover.icon
```

```
color.surface.tooltip.background  
color.surface.tooltip.border  
color.surface.tooltip.text
```

```
// One shared, soft shadow for all overlays:  
shadow.surface.overlay
```

I) Feedback surfaces (flat/outlined preferred)

CSS

 Copy code

```
color.surface.toast.background  
color.surface.toast.border  
color.surface.toast.text  
color.surface.toast.icon
```

```
color.surface.banner.background  
color.surface.banner.border  
color.surface.banner.text  
color.surface.banner.icon
```

J) Chips / Tags / Pills

CSS

 Copy code

```
color.surface.pill.background  
color.surface.pill.border  
color.surface.pill.text  
color.surface.pill.icon  
color.surface.pill.selected.background
```

K) Selection / Highlight / Focus

CSS

 Copy code

```
color.surface.selection.background  
color.surface.selection.border
```

```
color.surface.highlight.background  
color.surface.highlight.border
```

```
color.surface.focusRing
```

```
// can be color only; width live
```

L) Inverse & High-contrast

CSS

 Copy code

```
color.surface.inverse.background  
color.surface.inverse.border  
color.surface.inverse.text  
color.surface.inverse.icon
```

```
color.surface.contrastHigh.background  
color.surface.contrastHigh.border  
color.surface.contrastHigh.text  
color.surface.contrastHigh.icon
```

M) Data / Charts (container roles)

CSS

 Copy code

```
color.surface.chart.background  
color.surface.chart.border  
color.surface.chart.text  
color.surface.chart.icon  
color.surface.chart.gridline  
color.surface.chart.plot.background
```

N) Global dividers (surface-aware)

```
color.surface.divider.default  
color.surface.divider.muted  
color.surface.divider.emphasis
```

Notes for implementation

- Keep components **flat by default**; prefer `...outlined.border` or subtle `...background` steps for separation.
- Reserve shadows for:
 - **Overlays** (`shadow.surface.overlay`)
 - **Optional hover lift** (`shadow.surface.hover.card` or a generic `shadow.surface.hover.interactive`)
- In Tier-3, map specific components, e.g.:
 - `component.accountCard.bg` → `color.surface.card.emphasis.1.background` (if you introduce emphasis variants)
 - `component.sidebar.bg` → `color.surface.sidebar.background`
 - `component.modal.bg` → `color.surface.modal.background`

If you want, I can generate a **ready-to-ship JSON** mapping these Tier-2 roles to your **light/dark palette primitives** so you can drop it straight into your token pipeline.



lets do a overview on C) Cards & Wells (flat by default)

Alright — let's break down **C) Cards & Wells** with your *flat-by-default* approach so you can see exactly **what each token is for, when to use it, and how it fits into modern SaaS UI**.

Core principle

- **Cards** = standalone content blocks with a defined boundary.
- **Wells** = inset containers *inside* another surface, often for grouping or emphasis.

- **Flat by default** — no default shadows; rely on background contrast, borders, or subtle hover shadows for depth.
-

Card Tokens

Base properties

CSS

 Copy code

```
color.surface.card.background
color.surface.card.border
color.surface.card.border-strong
color.surface.card.divider
color.surface.card.text
color.surface.card.icon
```

- `background` → The main fill color for the card. Usually lighter than base (light mode) or darker (dark mode) for contrast.
 - `border` → Standard edge color for subtle separation from the background.
 - `border-strong` → Higher-contrast border for emphasis or high-density layouts.
 - `divider` → For horizontal/vertical separators *inside* the card (e.g., splitting header/body/footer).
 - `text` → Default text color inside the card.
 - `icon` → Icon color inside the card, tuned for background contrast.
-

State properties

arduino

 Copy code

```
color.surface.card.hover.background
color.surface.card.active.background
color.surface.card.selected.background
color.surface.card.disabled.background
```

- `hover.background` → Slightly different fill for hover state; can be paired with a small shadow if desired (`shadow.surface.hover.card`).
- `active.background` → When the card is being interacted with (click/tap hold).
- `selected.background` → For cards in selectable grids/lists.
- `disabled.background` → Muted tone for cards that are inactive/non-clickable.

Variant properties

CSS

 Copy code`color.surface.card.outlined.border`

- `outlined.border` → Used for the *outlined variant*, where the border is the primary separator instead of shadow or high background contrast.

Optional shadow

 Copy code`shadow.surface.hover.card`

- Only for **hover or drag states** to give a sense of elevation. Should be soft, subtle, and theme-consistent.

Well Tokens

CSS

 Copy code`color.surface.well.background`
`color.surface.well.border`
`color.surface.well.text`
`color.surface.well.icon`

- `well` is essentially a *secondary container* inside a card, form, or other surface.
- Used for:
 - Grouping related fields in a form
 - Highlighting optional/secondary information
 - Showing inline alerts/info inside a card
- Usually has:
 - Slightly different background from the parent card
 - Optional border for definition
 - No shadow (since it's embedded, not elevated)

Modern SaaS Usage Patterns

1. Dashboard widget

- `card.background` = light neutral
- `card.border` = subtle gray
- On hover → `card.hover.background` + `shadow.surface.hover.card`

2. Payment method card

- `card.background` = white
- `card.border-strong` for strong contrast in grid
- `card.selected.background` for active method

3. Form section well

- `well.background` = slightly tinted neutral
- `well.border` = light gray to separate from parent form background

Flat-by-default design benefits

- Keeps visual noise low
- Works well with high-density data layouts
- Lets you use **background steps** and **outlined borders** as the primary hierarchy tools
- Shadows remain special and intentional (interaction or overlays)

If you want, I can make a **visual reference grid** showing each of these card & well states/variants in **light and dark themes**, so devs and designers have a clear shared reference.

That way you lock in the *exact* look of “flat-by-default” with controlled interaction shadows.



provide a componetn exanple

Here's a **concrete example** of a **Card** component using the *flat-by-default* Tier-2 surface tokens we just defined.

I'll show you what tokens map to which parts so you can see how this works in practice.

Example: "Account Summary Card" (Dashboard Widget)

Purpose: Display account balance, account type, and quick actions inside a dashboard.

Structure & Token Mapping

php-template

 Copy code

```
<div class="card">
  <div class="card-header">
    <h3>Account Balance</h3>
    <button class="icon-button">⌵</button>
  </div>
  <div class="card-body">
    <p class="amount">$12,340.50</p>
    <span class="status-pill">Active</span>
  </div>
  <div class="card-footer">
    <a href="#">View Transactions</a>
  </div>
</div>
```

Token Application

Base styling (flat by default)

- **Background:** `color.surface.card.background`
- **Border:** `color.surface.card.border`
- **Divider (between header/body/footer):** `color.surface.card.divider`
- **Text:** `color.surface.card.text`
- **Icon:** `color.surface.card.icon`

Hover state (only on interactive cards)

- **Background:** `color.surface.card.hover.background`
- **Optional lift shadow:** `shadow.surface.hover.card`

Selected state (if selectable in a grid)

- **Background:** `color.surface.card.selected.background`

Token Usage in CSS Variables

CSS

 Copy code

```
.card {
  background: var(--color-surface-card-background);
  border: 1px solid var(--color-surface-card-border);
  color: var(--color-surface-card-text);
  border-radius: var(--radius-md);
  transition: background 0.2s ease, box-shadow 0.2s ease;
}

.card:hover {
  background: var(--color-surface-card-hover-background);
  box-shadow: var(--shadow-surface-hover-card);
}

.card-header,
.card-footer {
  border-bottom: 1px solid var(--color-surface-card-divider);
}

.icon-button {
  color: var(--color-surface-card-icon);
}
```

Well Example Inside the Card

If the card had a **grouped info area** inside:

html

 Copy code

```
<div class="well">
  <p>Spending limit: $20,000</p>
</div>
```

CSS

 Copy code

```
.well {
  background: var(--color-surface-well-background);
  border: 1px solid var(--color-surface-well-border);
  color: var(--color-surface-well-text);
}
```

```
border-radius: var(--radius-sm);  
padding: var(--space-md);  
}
```

Why this works well in a modern SaaS

- **Flat by default:** Only uses shadows for hover lift.
- **Theme-ready:** Switch light/dark mode by remapping Tier-1 primitives → Tier-2 tokens.
- **Scalable:** Any other “card” (e.g., billing card, report card) reuses the same Tier-2 surface tokens.
- **Consistent wells:** Form wells, info wells, and alert wells all share the same background/border logic.

If you want, I can make **three variations** of this card — default, outlined, and hover-lift — so you can see exactly how the **same Tier-2 tokens** produce different visual styles without changing the component’s HTML. That would make this ready for a design system spec sheet.



i am still not get why is call well?

Yeah, the name “**well**” isn’t very intuitive unless you’ve seen it in older UI frameworks — it comes from early web & desktop UI terminology.

Where the name comes from

- In older design systems (Bootstrap, Salesforce Lightning, Windows UI), a “**well**” was an **inset container** — like a shallow “dip” in the surface.
- It was used to **visually group content** *inside* another surface, often with a slightly different background and/or a border.
- The term is metaphorical: **like a shallow well in the ground** — it’s lower than the surrounding surface, so it looks inset.

Modern equivalent

Most modern design systems don't use the name "well" in their docs anymore — they use:

- Inset container
- Grouped panel
- Inline panel
- Sub-surface
- Inner section

Why keep the role

Even if we rename it, the **concept is still valuable** in SaaS UI:

- You might have a **card** with a **special area inside** for:
 - Grouping related form fields
 - Displaying a summary or warning
 - Separating secondary information from the main content
- This inset area often:
 - Has a different background from the parent
 - Uses a border or subtle color to separate it
 - Never uses shadows (because it's "inside")

Example

Without a "well":

Everything in the card looks the same, making it harder to see grouped areas.

With a "well":

yaml

 Copy code

```
[ Account Details Card ]
```

```
Name: Jane Doe
```

```
Email: jane@email.com
```

```
[ Billing Info ]
```

Card: **** 1234

Exp: 06/26

Last updated: 3 days ago

The [**Billing Info**] block would be a “well” — slightly tinted background, maybe a thin border.

If you want, we could **rename** `surface.well` in your Tier-2 tokens to something like:

- `surface.inset`
- `surface.inner`
- `surface.subsection`
- `surface.grouped`

That would make it clearer to modern designers/devs without losing the role in the system.

Do you want me to update the **final surface tokens** so “well” is replaced with a more modern, descriptive name?



so if is not in use anymore why you suggested to use?

Fair point — the only reason I suggested `well` earlier is because it’s still a *known* role name in some older design systems (Bootstrap, Salesforce Lightning, Atlassian UI kits), so if someone has worked with those, they’ll recognize it immediately.

But... in **modern SaaS design systems**, “well” is indeed **rarely used in public-facing docs** anymore, for two main reasons:

1. It’s jargon-y — new designers/devs often have no idea what it means without explanation.
 2. The same visual pattern is better described by **what it does** (e.g., “inset container”) instead of a metaphor.
-

Why I included it for you

When I built your Tier-2 list, I was trying to make sure **all possible surface roles** were represented — including embedded/inset containers inside cards or forms.

- The **concept** (an inset background area) is still used a lot.
 - The **term** “well” is just legacy naming that we can absolutely replace.
-

Better modern naming

We could rename `surface.well` → something like:

- `surface.inset` → Short, generic, clear.
 - `surface.inner` → Works well when inside a parent container.
 - `surface.group` → Implies it groups related content.
 - `surface.subsection` → A bit longer, but clear.
-

My recommendation for your design system

- Keep the **role** (because grouping/inset areas are valuable).
 - Drop the **term** “well” — pick one of the clearer names above.
 - Document it as *“A contained inset area inside another surface, used for grouping related content or emphasizing secondary info.”*
-

If you want, I can **update your final Tier-2 surface token list** so:

- `well` → `inset`
- All references are modernized
- No legacy UI jargon remains

Would you like me to rewrite that list with this update?



provide a better card surfaces final tokens

Alright — here's a **clean, modern, and final Tier-2 set** just for **Card surfaces**, based on your *flat-by-default* approach and without any legacy terms like "well."

This focuses on clarity, flexibility, and easy theming.


Role: `color.surface.card.*`

Cards are **standalone content blocks** with a defined boundary.

Flat by default — separation is created with **background contrast** and **borders**, not shadows (except on interaction).

Base

SCSS

 Copy code

```
color.surface.card.background
color.surface.card.border
color.surface.card.border-strong // Optional: high-contrast border fo
color.surface.card.divider      // For inner separators (header/body,
color.surface.card.text
color.surface.card.icon
```

States

arduino

 Copy code

```
color.surface.card.hover.background // Slight lift via color shift (may
color.surface.card.active.background // While pressed/clicked
color.surface.card.selected.background // For selected cards in grids/list
color.surface.card.disabled.background
color.surface.card.disabled.text
```

Variants

SCSS

 Copy code

```
color.surface.card.outlined.border // Outlined style (no shadow, strong
color.surface.card.subtle.background // Very light/dark tint; minimal vis
```

```
color.surface.card.emphasis.background // Stronger color for featured/impo
```

Optional Shadows (only when needed)

cpp

 Copy code

```
shadow.surface.card.hover // Light lift shadow on hover
shadow.surface.card.drag  // Slightly larger shadow for drag/d
```

Role: color.surface.inset.*

"Inset" replaces "well" — an **inner grouping container** inside a card or other surface, used to group related content or emphasize secondary information.

Base

css

 Copy code

```
color.surface.inset.background
color.surface.inset.border
color.surface.inset.text
color.surface.inset.icon
```

Variants

arduino

 Copy code

```
color.surface.inset.subtle.background // Minimal separation
color.surface.inset.emphasis.background // Highlighted grouping (e.g., warn
```

Example usage in SaaS UI

Account Card (default)

- background → color.surface.card.background
- border → color.surface.card.border

- On hover → `color.surface.card.hover.background` + `shadow.surface.card.hover`

Featured Offer Card

- `background` → `color.surface.card.emphasis.background`
- `text` → `color.surface.card.text`
- Border removed or set to `border-strong`

Grouped Billing Details (inside a card)

- Outer → `color.surface.card.background`
- Inner grouping → `color.surface.inset.background` + `color.surface.inset.border`

This keeps:

- **Modern naming** (no "well")
- **Flat by default** (shadows only for interaction)
- **Clear roles** (card = outer container, inset = inner grouping)
- **Theme-ready mapping** (light/dark/high-contrast)

If you want, I can now **map these card & inset tokens** to example **light and dark theme primitives** so you can see how they actually render in both modes.

That way you'll have a ready-to-ship design token spec for your cards.



lets go to and check if you need to improve or adapte this
E) Toolbars & Headers (inside areas)

Alright — let's review **E) Toolbars & Headers (inside areas)** and see how we can modernize/adapt it so it fits with the **flat-by-default** direction we've been using for surfaces like cards.

Current intent

"Toolbars & headers" are **functional horizontal strips** inside an area (like inside a card, table, container, modal) that:

- Contain **actions, filters, or titles**
 - Separate themselves visually from the content area
 - Do **not** take on the same role as global navigation bars
-

Where they appear in SaaS

- Toolbar at the top of a **table** with search/filter/export
 - Toolbar at the top of a **modal** with title and close button
 - Header bar in a **detail view** with breadcrumbs + actions
 - Panel or drawer **action strip**
-

How they differ from navigation surfaces

- Navigation (navbar, sidebar) = app-level context
 - Toolbars & headers = **component-level context**
 - They often live *inside* a surface like `container` or `card` and should inherit its overall shape
-

Improved Tier-2 tokens

scss

 Copy code

```
color.surface.toolbar.background
color.surface.toolbar.border
color.surface.toolbar.divider           // between groups of actions
color.surface.toolbar.text
color.surface.toolbar.icon

color.surface.toolbar.subtle.background // minimal separation from parent
color.surface.toolbar.emphasis.background // strong visual weight (highlight)
```

Optional states

arduino

 Copy code

```
color.surface.toolbar.hover.background // for interactive areas in toolbar
color.surface.toolbar.active.background
```

Interaction cues

Since we're keeping **flat-by-default**:

- Use `border` or `divider` to separate toolbar from its content area
 - Avoid default shadow — only use for overlays (e.g., sticky/fixed toolbars on scroll)
-

Example mapping

Table toolbar (light mode)

- Background → `color.surface.toolbar.background` = light neutral
- Border → `color.surface.toolbar.border` = subtle gray
- Divider → `color.surface.toolbar.divider` between filter and action groups

Modal header (dark mode)

- Background → `color.surface.toolbar.emphasis.background` = darker neutral
 - Text/Icon → `color.surface.toolbar.text` and `color.surface.toolbar.icon`
-

Potential improvement

We might want to make **two sub-roles**:

1. `toolbar` = for action strips with buttons/filters
2. `header` = for title-only top bars inside surfaces

This avoids overloading "toolbar" to mean both things.

If you want, I can **split E) into "Toolbar" and "Header" roles** and rewrite the tokens so each one has exactly what's needed for modern SaaS, without overlap or confusion. That would make it more future-proof when your design system scales.



provide a components example

Here's a **clear component example** for **E) Toolbars & Headers** showing how the Tier-2 tokens apply in a modern, flat-by-default SaaS design.

I'll give you two examples — one for a **toolbar** and one for a **header** — so you can see the difference.

1. Toolbar Example — Table Action Strip

Scenario: Inside a `container` or `card` holding a data table, there's a toolbar at the top with filters and action buttons.

HTML:

html

 Copy code

```
<div class="table-toolbar">
  <div class="toolbar-left">
    <button class="btn">+ Add Item</button>
  </div>
  <div class="toolbar-right">
    <input type="search" placeholder="Search..." />
    <button class="icon-btn">⌕</button>
  </div>
</div>
```

Token Mapping:

- **Background:** `color.surface.toolbar.background`
- **Border:** `color.surface.toolbar.border` (bottom border to separate from table content)
- **Divider:** `color.surface.toolbar.divider` (between left and right groups, optional)
- **Text:** `color.surface.toolbar.text`
- **Icon:** `color.surface.toolbar.icon`
- **Hover background for buttons:** `color.surface.toolbar.hover.background`

CSS (token usage):

```
.table-toolbar {
  background: var(--color-surface-toolbar-background);
  border-bottom: 1px solid var(--color-surface-toolbar-border);
  color: var(--color-surface-toolbar-text);
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: var(--space-sm) var(--space-md);
}

.table-toolbar button {
  color: var(--color-surface-toolbar-icon);
}

.table-toolbar button:hover {
  background: var(--color-surface-toolbar-hover-background);
}
```

2. Header Example — Modal Title Bar

Scenario: At the top of a modal, we have a header with a title and a close button.

HTML:

html

 Copy code

```
<div class="modal-header">
  <h2 class="modal-title">Edit Profile</h2>
  <button class="icon-btn" aria-label="Close">x</button>
</div>
```

Token Mapping:

- **Background:** color.surface.header.background
- **Border:** color.surface.header.border (optional bottom border)
- **Text:** color.surface.header.text
- **Icon:** color.surface.header.icon
- **Hover background for close button:** color.surface.header.hover.background

CSS (token usage):

```
.modal-header {  
  background: var(--color-surface-header-background);  
  border-bottom: 1px solid var(--color-surface-header-border);  
  color: var(--color-surface-header-text);  
  display: flex;  
  justify-content: space-between;  
  align-items: center;  
  padding: var(--space-sm) var(--space-md);  
}  
  
.modal-header .icon-btn {  
  color: var(--color-surface-header-icon);  
}  
  
.modal-header .icon-btn:hover {  
  background: var(--color-surface-header-hover-background);  
}
```

Key Takeaways

- **Toolbar** = action-oriented, usually contains interactive elements (buttons, inputs, filters).
- **Header** = title-oriented, often at the top of a surface, may have a few actions (like close/edit).
- Both **inherit** from their parent surface but have **their own role tokens** so you can tweak backgrounds, borders, and text contrast without affecting the parent container or card.

If you want, I can **add "toolbar" and "header" as two distinct roles in your final surface tokens list** so it's 100% clear in the design system which one to use in each case.

That would make it easier for developers to choose the correct role without overloading "toolbar" to mean everything.



elts go G) Forms & Fields (surface roles, not component props)

Alright — let's go deep on **G) Forms & Fields** so they fit our **flat-by-default** approach and work well for both **light/dark themes** and **SaaS complexity**.

This section is all about **surface roles**, not per-component props — meaning these tokens describe **the look of the background, borders, and text of input areas** without hardcoding them to "TextField" or "Select."

Role: color.surface.field.*

These define **semantic roles** for form inputs across the system (text inputs, selects, textareas, date pickers, etc.).

This way, you can **change the field appearance globally** without touching individual component styles.

Base

CSS

 Copy code

```
color.surface.field.background // Base fill for input area
color.surface.field.border     // Default border for input
color.surface.field.text       // Main input text color
color.surface.field.icon       // Icon color inside inputs (prefix/s
color.surface.field.placeholder // Placeholder text color
```

States

SCSS

 Copy code

```
color.surface.field.hover.background // Slightly different fill for hover
color.surface.field.focus.border     // Highlighted border for active/focu
color.surface.field.active.border    // For click/drag interactions in sli
color.surface.field.disabled.background
```

```
color.surface.field.disabled.text
color.surface.field.invalid.border // For error state
color.surface.field.invalid.text
color.surface.field.readonly.background
color.surface.field.readonly.border
```

Variants

SCSS

 Copy code

```
color.surface.field.outlined.border // Stronger border instead of filled
color.surface.field.subtle.background // Minimal fill (blends with parent)
```

Why this works well

- **Theme-ready** — Switching light/dark mode is just remapping Tier-1 colors to these Tier-2 roles.
 - **Consistent across components** — Date pickers, dropdowns, and plain text inputs all share the same background/border logic.
 - **Flat-by-default** — No default shadow on inputs, just border & background changes for states.
 - **Accessible** — Tokenizing placeholder and invalid text colors makes it easy to meet contrast requirements.
-

Component Example — “Email Address” Field

HTML:

html

 Copy code

```
<label for="email">Email address</label>
<div class="input-wrapper">
  <input type="email" id="email" placeholder="you@example.com">
  <span class="icon">@</span>
</div>
```

CSS using tokens:

CSS

```
.input-wrapper {
  background: var(--color-surface-field-background);
  border: 1px solid var(--color-surface-field-border);
  display: flex;
  align-items: center;
  padding: 0 var(--space-sm);
  border-radius: var(--radius-sm);
  transition: background 0.2s ease, border-color 0.2s ease;
}

.input-wrapper:hover {
  background: var(--color-surface-field-hover-background);
}

.input-wrapper:focus-within {
  border-color: var(--color-surface-field-focus-border);
}

.input-wrapper input {
  flex: 1;
  border: none;
  background: transparent;
  color: var(--color-surface-field-text);
}

.input-wrapper input::placeholder {
  color: var(--color-surface-field-placeholder);
}

.icon {
  color: var(--color-surface-field-icon);
}
```

State Example

- **Default:** field.background + field.border
- **Hover:** field.hover.background
- **Focus:** field.focus.border (accent color)
- **Invalid:** field.invalid.border + field.invalid.text
- **Disabled:** field.disabled.background + field.disabled.text

If you want, I can also **extend this section** so we include **field grouping surfaces** (like inset containers for grouped form fields), which would make this part work seamlessly with the **inset role we defined for cards**.

That would make your form tokens fully consistent with the rest of the surface system.



what is readonly?

In forms, `readonly` means the field **can't be edited** by the user, but:

- The field is **still focusable** (you can click into it and select/copy the text).
- The value is **visible** and **looks active** enough to read clearly.
- It's not "disabled" — it remains part of the tab order and can be used for reference.

When to use `readonly`

- Showing a **generated value** (e.g., API key, invoice number) the user shouldn't change.
- Displaying data that's **fixed for now** but may become editable in another state.
- Allowing **copy** without allowing **edit**.

How it differs from `disabled`

State	Editable?	Focusable?	Selectable?	Submitted with form?
Normal	✔ Yes	✔ Yes	✔ Yes	✔ Yes
Readonly	✘ No	✔ Yes	✔ Yes	✔ Yes
Disabled	✘ No	✘ No	✘ No	✘ No

Why have `readonly` tokens

- `color.surface.field.readonly.background` — Often matches default background but with slightly lower emphasis to signal non-editability.

- `color.surface.field.readonly.border` — Usually the same or more subtle than the default border, but not as “inactive” as disabled.
 - Keeps **UX clarity**:
 - `readonly` → “You can look/copy but not change.”
 - `disabled` → “You can’t use this at all.”
-

If you want, I can show **an example of a readonly API Key field** with the tokens we defined so you can see the difference in look/feel from normal and disabled. That would make it clearer in a SaaS UI context.



provide a better example

Alright — let’s make a **clear SaaS-style example** showing **normal**, **readonly**, and **disabled** form fields side by side, using our Tier-2 `color.surface.field.*` tokens.

Example: API Key Field

HTML:

html

 Copy code

```
<div class="form-field">
  <label for="api-key">API Key (Normal)</label>
  <input type="text" id="api-key" value="sk_live_123456789" />
</div>

<div class="form-field">
  <label for="api-key-readonly">API Key (Readonly)</label>
  <input type="text" id="api-key-readonly" value="sk_live_123456789" reado
</div>

<div class="form-field">
  <label for="api-key-disabled">API Key (Disabled)</label>
```

```
<input type="text" id="api-key-disabled" value="sk_live_123456789" disab
</div>
```

CSS with tokens:

CSS

 Copy code

```
.form-field {
  margin-bottom: var(--space-md);
  display: flex;
  flex-direction: column;
}

label {
  margin-bottom: var(--space-xs);
  color: var(--color-text-primary);
  font-weight: 500;
}

input[type="text"] {
  padding: var(--space-sm) var(--space-md);
  border-radius: var(--radius-sm);
  font-size: var(--font-size-md);
  color: var(--color-surface-field-text);
  background: var(--color-surface-field-background);
  border: 1px solid var(--color-surface-field-border);
  transition: background 0.2s ease, border-color 0.2s ease;
}

input[type="text"]:hover:not(:disabled):not([readonly]) {
  background: var(--color-surface-field-hover-background);
}

input[type="text"]:focus {
  outline: none;
  border-color: var(--color-surface-field-focus-border);
}

input[readonly] {
  background: var(--color-surface-field-readonly-background);
  border-color: var(--color-surface-field-readonly-border);
  cursor: text; /* still selectable */
}
```

}

```
input:disabled {
  background: var(--color-surface-field-disabled-background);
  border-color: var(--color-surface-field-disabled-border);
  color: var(--color-surface-field-disabled-text);
  cursor: not-allowed;
}
```

How it looks in a SaaS UI

State	Look/Feel	UX Behavior
Normal	Solid border, standard background, hover/focus effects	User can type, select, copy, paste
Readonly	Slightly muted background, softer border, no hover background change	User can focus, select, copy , but cannot edit
Disabled	Muted background and text, cursor shows "not-allowed", no focus or selection possible	User cannot interact in any way

✔ Why this example works:

- It shows clearly **how readonly is different** from disabled — especially important in SaaS apps where fields often contain API keys, IDs, or generated tokens.
- Uses **Tier-2 surface tokens** so the styling is consistent across all forms.
- Keeps the "flat-by-default" look, relying on border & background states rather than heavy shadows.

If you want, I can also add a **readonly field with a "Copy" button** beside it (common in developer portals), which would make it even more realistic for SaaS use cases. That's a typical pattern for API key or account ID fields.



lets go and od: H) Overlays (true elevation → allowed shadow)

Awesome—here's a tight, production-ready spec for **H) Overlays** that fits your **flat-by-default** system (shadows allowed for true elevation only).

H) Overlays (true elevation → allowed shadow)

Roles & intent

Overlays sit **above** the app to present transient UI: **scrims**, **modals**, **drawers**, **popovers**, **tooltips**.

They use **their own surface roles** + **one shared overlay shadow** for consistency.

Token set (Tier-2, semantic)

1) Scrims (backdrop layers)

arduino

 Copy code

```
color.surface.overlay.scrim.light.background // e.g., rgba(0,0,0,0.25)
color.surface.overlay.scrim.dark.background  // e.g., rgba(0,0,0,0.6) fo
```

Use: Dim app behind modals/drawers; choose `.light` for subtle, `.dark` for high focus or HC mode.

2) Modal

css

 Copy code

```
color.surface.modal.background
color.surface.modal.border
color.surface.modal.text
color.surface.modal.icon
```

```
color.surface.modal.divider  
shadow.surface.overlay
```

Notes:

- Flat fill + border for clarity; use the **single overlay shadow**.
 - Divider separates header/body/footer.
-

3) Drawer (side panel)

CSS

 Copy code

```
color.surface.drawer.background  
color.surface.drawer.border  
color.surface.drawer.text  
color.surface.drawer.icon  
color.surface.drawer.divider  
shadow.surface.overlay
```

Notes:

- Shares the same shadow token as modal.
 - Border aligns with edge it slides from.
-

4) Popover (menus, dropdowns, context menus)

CSS

 Copy code

```
color.surface.popover.background  
color.surface.popover.border  
color.surface.popover.text  
color.surface.popover.icon  
color.surface.popover.divider  
shadow.surface.overlay
```

Notes:

- Border is important for contrast in both themes.
 - Keep radius tight; rely on border + shadow for separation.
-

5) Tooltip

CSS

 Copy code

```
color.surface.tooltip.background
color.surface.tooltip.border
color.surface.tooltip.text
shadow.surface.overlay
```

Notes:

- Border may be transparent in light mode if contrast is already adequate.
 - Use the same shadow (reduced blur via component styles if needed).
-

6) Shared overlay shadow (one token to rule them all) Copy code

```
shadow.surface.overlay
```

Guideline: A single, soft, color-aware shadow that works for modal/drawer/popover/tooltip.

Implementation detail (example only):

- Light: medium blur, low spread, dark alpha
 - Dark: lower blur, **lighter** shadow color (avoid heavy glow), possibly add a faint outline via `border` tokens for contrast
-

Optional states (keep minimal)

For interactive overlay bodies (menus, selectable list items), reuse existing state tokens:

- Use the **list/table row** or **card hover/selected** background tokens inside popovers/menus for consistency (don't invent overlay-specific row states).
-

Accessibility & behavior guidelines

- **Contrast:** Ensure `modal.text` & `popover.text` meet WCAG 4.5:1 on their backgrounds.
- **Scrim opacity:**
 - `.light` \approx 20–30% dim (let context show)
 - `.dark` \approx 50–70% dim (high focus tasks)

- **Keyboard focus:** Trap focus inside modal/drawer; return focus on close.
- **Motion:** Respect "reduce motion." Use your `motion.duration.*` + `motion.easing.*` tokens.
- **Dismiss:**
 - Modal: close via X button, ESC, and scrim click (if non-destructive)
 - Drawer/Popover: close on outside click, ESC, or selection
- **Z-index layering:** Define a scale (Tier-1) and assign: tooltip < popover < drawer < modal.

Example components

Modal (HTML/CSS with tokens)

html

 Copy code

```
<div class="scrim"></div>
<div class="modal">
  <div class="modal-header">
    <h2>Invite teammate</h2>
    <button class="icon-btn" aria-label="Close">x</button>
  </div>
  <div class="modal-body">
    <!-- content -->
  </div>
  <div class="modal-footer">
    <button class="btn-primary">Send</button>
    <button class="btn-ghost">Cancel</button>
  </div>
</div>
```

css

 Copy code

```
.scrim {
  background: var(--color-surface-overlay-scrim-dark-background);
  position: fixed; inset: 0;
}

.modal {
  background: var(--color-surface-modal-background);
  color: var(--color-surface-modal-text);
  border: 1px solid var(--color-surface-modal-border);
}
```

```

    box-shadow: var(--shadow-surface-overlay);
    border-radius: var(--radius-lg);
    max-width: 640px; width: 100%;
  }

.modal-header, .modal-footer {
  border-bottom: 1px solid var(--color-surface-modal-divider);
  padding: var(--space-md) var(--space-lg);
}

.modal-footer { border-bottom: 0; border-top: 1px solid var(--color-surfac

```

Drawer (right side)

html

 Copy code

```

<div class="scrim"></div>
<aside class="drawer drawer-right">
  <header class="drawer-header">
    <h3>Filters</h3>
    <button class="icon-btn" aria-label="Close">x</button>
  </header>
  <div class="drawer-body">...</div>
</aside>

```

css

 Copy code

```

.drawer {
  background: var(--color-surface-drawer-background);
  color: var(--color-surface-drawer-text);
  border-left: 1px solid var(--color-surface-drawer-border);
  box-shadow: var(--shadow-surface-overlay);
  width: 420px; max-width: 90vw;
}

.drawer-header {
  border-bottom: 1px solid var(--color-surface-drawer-divider);
  padding: var(--space-md) var(--space-lg);
}

```

Popover (menu)

html

 Copy code

```
<div class="popover" role="menu">
  <button class="menu-item">Profile</button>
  <button class="menu-item">Settings</button>
  <hr class="divider" />
  <button class="menu-item destructive">Sign out</button>
</div>
```

CSS

 Copy code

```
.popover {
  background: var(--color-surface-popover-background);
  color: var(--color-surface-popover-text);
  border: 1px solid var(--color-surface-popover-border);
  box-shadow: var(--shadow-surface-overlay);
  border-radius: var(--radius-md);
  padding: var(--space-xs) 0;
}
.popover .divider { border-top: 1px solid var(--color-surface-popover-divi
.menu-item:hover { background: var(--color-surface-card-hover-background);
```

Tooltip

CSS

 Copy code

```
.tooltip {
  background: var(--color-surface-tooltip-background);
  color: var(--color-surface-tooltip-text);
  border: 1px solid var(--color-surface-tooltip-border);
  box-shadow: var(--shadow-surface-overlay);
  padding: var(--space-2xs) var(--space-xs);
  border-radius: var(--radius-sm);
  font-size: var(--text-label-sm); /* semantic typography token */
}
```

Mapping & theming tips

- Light theme:

- Modal/Popover bg = near-white; border = neutral-300; text = neutral-900
 - Tooltip bg = neutral-900; text = white; border = transparent or neutral-800
 - **Dark theme:**
 - Modal/Popover bg = neutral-900/950; border = neutral-700; text = neutral-100
 - Tooltip bg = neutral-100; text = neutral-900; border = neutral-200
 - **High contrast:** Increase border and divider contrast, prefer `.dark` scrim.
-

Minimal motion recipe (tokens)

- **Modal/Drawer:** `motion.duration.medium`, `motion.easing.decelerate`
 - Modal: scale 0.98 → 1 & fade
 - Drawer: `translateX/translateY` from edge & fade scrim
 - **Popover/Tooltip:** `motion.duration.fast`, `motion.easing.standard`
 - Small translate (4–8px) + fade
-

If you like, I can generate a **token JSON scaffold** for H) with light/dark sample values that reference your existing palettes, so you can drop it straight into your pipeline.



lets do a J) Chips / Tags / Pills

Alright — let's lock in **J) Chips / Tags / Pills** so they work seamlessly with the **flat-by-default** philosophy, remain theme-friendly, and scale across SaaS use cases.

Role: `color.surface.pill.*`

Covers **small, inline surfaces** used for status, categorization, filters, and labels.

Base

CSS

 Copy code

```
color.surface.pill.background  
color.surface.pill.border  
color.surface.pill.text  
color.surface.pill.icon
```

- `background` → Main fill for the pill.
 - `border` → Defines edge; may be subtle in filled style or strong in outlined variant.
 - `text` → Primary text label color.
 - `icon` → Icon color for status or action (e.g., "x" remove icon).
-

States

arduino

 Copy code

```
color.surface.pill.hover.background  
color.surface.pill.active.background  
color.surface.pill.selected.background  
color.surface.pill.disabled.background  
color.surface.pill.disabled.text
```

- `hover` → Small color step for interactive pills (filters, clickable tags).
 - `active` → Applied when pressed or toggled.
 - `selected` → For persistent selection (filter is "on").
 - `disabled` → Muted style, non-interactive.
-

Variants

CSS

 Copy code

```
color.surface.pill.outlined.border  
color.surface.pill.subtle.background  
color.surface.pill.emphasis.background
```

- `outlined` → Transparent or minimal background, strong border.
 - `subtle` → Minimal contrast background, good for status indicators.
 - `emphasis` → High contrast fill for priority or call-to-action tags.
-

Usage patterns in SaaS UI

1. Status Label

- e.g., Active , Pending , Suspended
- background = subtle role color (green, amber, red)
- text = contrasting foreground
- Non-interactive

2. Filter Tag

- Clickable pill to apply/remove a filter
- Has `hover.background` and `active.background` states
- May use `outlined` for unselected state

3. Category Chip

- Tag for grouping (e.g., "Marketing", "Finance")
- Often colored for category identification
- Might allow removal via icon

Component Example — Filter Pill

HTML:

html

 Copy code

```
<button class="pill selected">
  <span class="label">Active Users</span>
  <span class="icon">x</span>
</button>
```

CSS with tokens:

CSS

 Copy code

```
.pill {
  display: inline-flex;
  align-items: center;
  padding: 0 var(--space-sm);
  border-radius: var(--radius-full);
  font-size: var(--text-label-sm);
  line-height: var(--line-height-tight);
  border: 1px solid var(--color-surface-pill-border);
  background: var(--color-surface-pill-background);
  color: var(--color-surface-pill-text);
  transition: background 0.15s ease, border-color 0.15s ease;
}
```

```
.pill .icon {
  color: var(--color-surface-pill-icon);
  margin-left: var(--space-2xs);
}

.pill:hover {
  background: var(--color-surface-pill-hover-background);
}

.pill:active {
  background: var(--color-surface-pill-active-background);
}

.pill.selected {
  background: var(--color-surface-pill-selected-background);
}

.pill.disabled {
  background: var(--color-surface-pill-disabled-background);
  color: var(--color-surface-pill-disabled-text);
  cursor: not-allowed;
}
```

Example theme mapping

Light mode

- pill.background = neutral-100
- pill.border = neutral-300
- pill.text = neutral-900
- pill.hover.background = neutral-200
- pill.selected.background = brand-50

Dark mode

- pill.background = neutral-800
- pill.border = neutral-600
- pill.text = neutral-100
- pill.hover.background = neutral-700
- pill.selected.background = brand-800

If you want, I can extend this to include **status-specific pill tokens** like `pill.status.success.background`, `pill.status.error.background`, etc., so your design system has ready-to-use roles for the most common SaaS labels. That way you'd never have to hardcode green/red for statuses again. Would you like me to add those?



this is just a select chip? or a status chip usually on tables?

What I just wrote actually covers **both patterns** — but they are two very different use cases in SaaS UI.

1. Select Chip (interactive)

- **Purpose:** Clickable/toggleable element for filters, selections, tags.
- **Where:** Filter bars, multi-select inputs, search refinements.
- **Behavior:**
 - Has **hover**, **active**, and **selected** states.
 - Often includes a close/remove icon ("x") for quick removal.
 - Background changes when selected or hovered.

Example:

Show: [Active Users x] [Paying Customers x]

2. Status Chip (non-interactive)

- **Purpose:** Static label showing a **state or category**.
- **Where:** Common in **tables** (last column showing "Active", "Pending", "Suspended"), detail views, dashboards.
- **Behavior:**
 - No hover or active states.
 - Color indicates meaning — e.g., green for success, amber for pending, red for error.
 - Often small and purely informative.

Example:

[Active] [Pending] [Failed]

Token approach difference

We can keep `color.surface.pill.*` as the **base styling role**, then define **two sub-roles**:

Interactive chip (select chip)

vbnet

 Copy code

```
color.surface.pill.select.background
color.surface.pill.select.hover.background
color.surface.pill.select.selected.background
color.surface.pill.select.border
color.surface.pill.select.text
```

Status chip (table label)

lua

 Copy code

```
color.surface.pill.status.success.background
color.surface.pill.status.success.text
color.surface.pill.status.warning.background
color.surface.pill.status.warning.text
color.surface.pill.status.error.background
color.surface.pill.status.error.text
color.surface.pill.status.info.background
color.surface.pill.status.info.text
```

Why separate them

- Status chips don't need hover/active states — they should look stable and static.
 - Select chips must have clear interactive states and focus/selection feedback.
 - Having both roles avoids overloading one set of tokens with unrelated styles.
-

If you want, I can rewrite **J) Chips / Tags / Pills** in the final list so it explicitly has **two categories**:

Status chips (static) and **Select chips (interactive)** — with their own token sets and theme mapping.

That will make it much easier for devs to pick the right role in your SaaS design system.



i have 3 colors of chips red, black and white,

Got it — so in your system you basically have **three chip color styles** that get reused across contexts:

- **Red chip** → Usually means **error, danger, or destructive** status/action.
- **Black chip** → Usually **neutral / dark emphasis**, often for labels or categories.
- **White chip** → Usually **light emphasis**, often for filters, tags, or default state.

How I'd model that in tokens

Since you have only 3 colorways, I'd make **3 variants** under `color.surface.pill` and allow each to have both **static (status)** and **interactive (select)** usage.

CSS

 Copy code

```
color.surface.pill.red.background
color.surface.pill.red.text
color.surface.pill.red.border
```

```
color.surface.pill.black.background
color.surface.pill.black.text
color.surface.pill.black.border
```

```
color.surface.pill.white.background
color.surface.pill.white.text
color.surface.pill.white.border
```

For interactive chips (select chips)

Add states for each:

arduino

color.surface.pill.red.hover.background
color.surface.pill.red.selected.background

color.surface.pill.black.hover.background
color.surface.pill.black.selected.background

color.surface.pill.white.hover.background
color.surface.pill.white.selected.background

For static status chips (like in tables)

You don't need hover/selected — just background, text, and border.

Example Mapping

Light theme

Role	Background	Text	Border
Red	#FEE2E2	#991B1B	#FCA5A5
Black	#111827	#FFFFFF	#1F2937
White	#FFFFFF	#111827	#D1D5DB

Dark theme

Role	Background	Text	Border
Red	#7F1D1D	#FEE2E2	#B91C1C
Black	#000000	#FFFFFF	#1F2937
White	#F9FAFB	#111827	#D1D5DB

How they map in SaaS UI

- **Red** → "Error", "Suspended", or "Critical" labels in tables; "Remove" tags in filters.
- **Black** → Category chips like "Enterprise", "Admin"; strong contrast filter tags.

- **White** → Default filter chips; minimal neutral status chips.

If you want, I can turn this into a **final J) Chips / Tags / Pills** token spec that includes **only your 3 colorways**, but still separates **interactive** vs **status** usage so it's clear for devs when to apply hover states and when not to.



so could be primary second tryaty?

Yes — instead of thinking of them only as “red, black, white,” you can treat them as **semantic variants** like **Primary**, **Secondary**, and **Tertiary**.

That way:

- The **visual color** (red/black/white) can change in the future without renaming the token.
- Your devs & designers work with **meaningful roles** instead of hard-coded colors.
- The same role can be themed differently for light/dark modes or brand updates.

Example Mapping

Tier-2 Semantic Roles

CSS

 Copy code

```
color.surface.pill.primary.background
color.surface.pill.primary.text
color.surface.pill.primary.border
```

```
color.surface.pill.secondary.background
color.surface.pill.secondary.text
color.surface.pill.secondary.border
```

```
color.surface.pill.tertiary.background
color.surface.pill.tertiary.text
color.surface.pill.tertiary.border
```

Your Current Visual Mapping

Role	Light Mode Visual	Dark Mode Visual
Primary	Red chip (error/danger)	Darker red tone
Secondary	Black chip (neutral emphasis)	Black or deep gray
Tertiary	White chip (light emphasis)	Light gray/white

Interactive States (only for select chips)

arduinoCopy code

```
color.surface.pill.primary.hover.background
color.surface.pill.primary.selected.background

color.surface.pill.secondary.hover.background
color.surface.pill.secondary.selected.background

color.surface.pill.tertiary.hover.background
color.surface.pill.tertiary.selected.background
```

Why this is better than color names

- **Future-proof:** If tomorrow "primary" chip becomes blue instead of red, you don't have to rename tokens everywhere.
- **Theming flexibility:** Dark/light modes or brand-specific themes can assign any color to "primary," "secondary," "tertiary."
- **Consistency:** The roles can be applied across **status chips in tables** and **interactive filter chips** without confusion.

If you want, I can rewrite your **J) Chips / Tags / Pills** section so it has **Primary / Secondary / Tertiary** as the only three variants, but still supports **status** and **interactive** modes.

That will make your design system clean, scalable, and easier for devs to implement.



make the fully pills tokens, all posibilie include staud
select, hover secondry primary

Alright — here's the **final, exhaustive token spec** for **J) Chips / Tags / Pills**
with **Primary, Secondary, and Tertiary** variants —
covering **status (static)** and **selectable (interactive)** states in a clean, semantic way.

Base Role

php-template

 Copy code

```
color.surface.pill.<variant>.<property>
```

Where:

- <variant> = primary | secondary | tertiary
 - <property> = background | text | border | icon
-

1. Static Status Pills (non-interactive)

For chips in **tables, dashboards**, or anywhere they indicate a state but aren't clickable.

CSS

 Copy code

```
color.surface.pill.primary.background  
color.surface.pill.primary.text  
color.surface.pill.primary.border  
color.surface.pill.primary.icon
```

```
color.surface.pill.secondary.background  
color.surface.pill.secondary.text  
color.surface.pill.secondary.border  
color.surface.pill.secondary.icon
```

```
color.surface.pill.tertiary.background  
color.surface.pill.tertiary.text
```

```
color.surface.pill.tertiary.border  
color.surface.pill.tertiary.icon
```

2. Selectable Pills (interactive — filters, tag selectors)

Add hover, active, and selected states for interactive chips.

CSS

 Copy code

```
color.surface.pill.primary.hover.background  
color.surface.pill.primary.active.background  
color.surface.pill.primary.selected.background  
color.surface.pill.primary.selected.text  
color.surface.pill.primary.selected.icon  
  
color.surface.pill.secondary.hover.background  
color.surface.pill.secondary.active.background  
color.surface.pill.secondary.selected.background  
color.surface.pill.secondary.selected.text  
color.surface.pill.secondary.selected.icon  
  
color.surface.pill.tertiary.hover.background  
color.surface.pill.tertiary.active.background  
color.surface.pill.tertiary.selected.background  
color.surface.pill.tertiary.selected.text  
color.surface.pill.tertiary.selected.icon
```

3. Disabled State

For any variant, when the chip is shown but cannot be interacted with.

CSS

 Copy code


```
color.surface.pill.primary.disabled.background  
color.surface.pill.primary.disabled.text  
color.surface.pill.primary.disabled.icon  
  
color.surface.pill.secondary.disabled.background  
color.surface.pill.secondary.disabled.text  
color.surface.pill.secondary.disabled.icon
```

```
color.surface.pill.tertiary.disabled.background
color.surface.pill.tertiary.disabled.text
color.surface.pill.tertiary.disabled.icon
```

4. Optional Variant Styles

If you support multiple visual styles (filled vs outlined), add:

CSS

 Copy code

```
color.surface.pill.primary.outlined.border
color.surface.pill.secondary.outlined.border
color.surface.pill.tertiary.outlined.border
```

- **Outlined** → No fill or minimal background, stronger border emphasis.

Example Usage in SaaS UI

Status Chip (Static)

- `color.surface.pill.primary.background` = red tone for "Error"
- `color.surface.pill.secondary.background` = black tone for "Admin"
- `color.surface.pill.tertiary.background` = white/light tone for "Default"

Filter Chip (Selectable)

- Hover → Slight color shift: `hover.background`
- Selected → Stronger fill and text/icon inversion: `selected.background` + `selected.text`
- Disabled → Muted fill and text, cursor: `not-allowed`

Example Light Theme Mapping

Variant	Background	Text	Border
Primary	#FEE2E2 (light red)	#991B1B	#FCA5A5
Secondary	#111827 (black)	#FFFFFF	#1F2937
Tertiary	#FFFFFF (white)	#111827	#D1D5DB

Example Dark Theme Mapping

Variant	Background	Text	Border
Primary	#7F1D1D (dark red)	#FEE2E2	#B91C1C
Secondary	#000000 (black)	#FFFFFF	#1F2937
Tertiary	#F9FAFB (light gray)	#111827	#D1D5DB

If you want, I can also **add a state diagram** showing exactly which of these tokens are used for each chip type (status vs selectable) so your devs instantly know when to use `hover` and when not to.

That would make this spec completely implementation-ready.



lets go and do:

I) Feedback surfaces (flat/outlined preferred)

Alright — let’s define **I) Feedback Surfaces** so they match the rest of your **flat-by-default** system, are **semantic** (not hard-coded to colors), and work across **toast notifications, banners**, and other inline feedback.

Purpose

Feedback surfaces give **user-visible system messages** such as:

- **Positive** (success, confirmation)
- **Informational** (neutral updates)
- **Warning** (caution, risk)
- **Negative** (error, failure)

They appear as **toasts, banners**, or inline **alerts**.

Base Role Structure

php-template

 Copy code

```
color.surface.<feedbackType>.<component>.<property>
```

Where:

- <feedbackType> = success | info | warning | error
- <component> = toast | banner
- <property> = background | border | text | icon
- Optional states for interactive banners.

1. Toast Notifications

Transient messages, often floating over the app UI.

CSS

 Copy code

```
color.surface.toast.success.background  
color.surface.toast.success.border  
color.surface.toast.success.text  
color.surface.toast.success.icon
```

```
color.surface.toast.info.background  
color.surface.toast.info.border  
color.surface.toast.info.text  
color.surface.toast.info.icon
```

```
color.surface.toast.warning.background  
color.surface.toast.warning.border  
color.surface.toast.warning.text  
color.surface.toast.warning.icon
```

```
color.surface.toast.error.background  
color.surface.toast.error.border  
color.surface.toast.error.text  
color.surface.toast.error.icon
```

Shadow:

 Copy code

`shadow.surface.toast`

Small, subtle elevation so it stands out against the app.

2. Banners

Inline persistent messages at the top or inside containers.

CSS

 Copy code

```
color.surface.banner.success.background
color.surface.banner.success.border
color.surface.banner.success.text
color.surface.banner.success.icon
```

```
color.surface.banner.info.background
color.surface.banner.info.border
color.surface.banner.info.text
color.surface.banner.info.icon
```

```
color.surface.banner.warning.background
color.surface.banner.warning.border
color.surface.banner.warning.text
color.surface.banner.warning.icon
```

```
color.surface.banner.error.background
color.surface.banner.error.border
color.surface.banner.error.text
color.surface.banner.error.icon
```

3. States for Interactive Banners

If a banner contains buttons/links:

arduino

 Copy code


```
color.surface.banner.<feedbackType>.hover.background
color.surface.banner.<feedbackType>.active.background
```

Keeps hover/active feedback subtle, maintaining the main feedback color identity.

4. Variants

Flat vs Outlined — outlined uses a transparent or minimal background with strong border:

CSS

 Copy code

```
color.surface.banner.success.outlined.border
color.surface.banner.info.outlined.border
color.surface.banner.warning.outlined.border
color.surface.banner.error.outlined.border
```

5. Example Light Theme Mapping

Type	Background	Border	Text
Success	#ECFDF5	#A7F3D0	#065F46
Info	#EFF6FF	#BFDBFE	#1E40AF
Warning	#FFFBEB	#FCD34D	#92400E
Error	#FEF2F2	#FCA5A5	#991B1B

6. Example Dark Theme Mapping

Type	Background	Border	Text
Success	#064E3B	#34D399	#D1FAE5
Info	#1E3A8A	#60A5FA	#DBEAFE
Warning	#78350F	#FBBF24	#FEF3C7
Error	#7F1D1D	#F87171	#FEE2E2

Example Components

Toast

html

 Copy code

```
<div class="toast success">
  <span class="icon">✓</span>
  <span class="message">Profile updated successfully</span>
</div>
```

css

 Copy code

```
.toast.success {
  background: var(--color-surface-toast-success-background);
  border: 1px solid var(--color-surface-toast-success-border);
  color: var(--color-surface-toast-success-text);
  box-shadow: var(--shadow-surface-toast);
}
```

Banner

html

 Copy code

```
<div class="banner warning">
  <span class="icon">⚠</span>
  <span class="message">Password will expire in 5 days.</span>
  <a href="#">Update now</a>
</div>
```

css

 Copy code

```
.banner.warning {
  background: var(--color-surface-banner-warning-background);
  border: 1px solid var(--color-surface-banner-warning-border);
  color: var(--color-surface-banner-warning-text);
}
```

If you want, I can **merge this with the "status chip" colors** from J) so your chips, banners, and toasts share the exact same success/info/warning/error palette — keeping

your whole system visually consistent without re-picking colors for each surface.
Would you like me to do that?



diffrence between banner and toast?

Here’s the short answer:

Aspect	Banner	Toast
Placement	Inside the app layout (often top of a page or container)	Floating above content, typically bottom-left/right or top-right
Persistence	Persistent — stays visible until dismissed or page changes	Transient — auto-disappears after a few seconds
Purpose	Communicates important information relevant to the current screen	Notifies about events that just happened (feedback from user action)
Interaction	May have actions (buttons/links) for the user to respond	May have a close button, but actions are rare/minimal
Size	Usually full-width of its container	Compact, self-contained
Disruption	More disruptive — occupies space in layout	Less disruptive — overlays content without shifting layout
Examples	“Your subscription will expire in 3 days” (with renew button)	“Profile updated successfully” after save

Think of it like this


- **Banner** = Part of the **page** → persistent, integrated into layout.
- **Toast** = Part of the **system layer** → floating, temporary.

If you want, I can adjust I) **Feedback Surfaces** so *the exact same tokens* can theme both banners and toasts, but with **different default behavior** (persistence, position, motion). That way you don't have to maintain two separate color sets — only one shared feedback palette.



lets do overlay:

```
color.surface.overlay.scrim.light.background  
color.surface.overlay.scrim.dark.background
```

Got it  — here's the **overlay scrim** part fully defined so it works across modals, drawers, popovers, and any future elevated surfaces.

Tokens

arduino

 Copy code

```
color.surface.overlay.scrim.light.background  
color.surface.overlay.scrim.dark.background
```

Purpose

The **scrim** (a.k.a. backdrop or overlay mask) sits **between** elevated UI and the underlying app content, to:

- Dim the background
- Focus attention on the overlay
- Prevent accidental interaction with underlying elements (when modal/drawer is blocking)

Usage Guidelines

light background

- **Subtle dimming** — underlying UI still visible enough to keep context.
 - Use for:
 - **Non-blocking overlays** (e.g., context menus, popovers, side panels that aren't modal)
 - **In-product tools** where background awareness is important
 - Typical opacity: **20–30%**
 - Example: `rgba(0, 0, 0, 0.25)` in light mode; `rgba(255, 255, 255, 0.12)` in dark mode
-

dark background

- **Strong dimming** — puts all focus on the overlay, background becomes secondary.
 - Use for:
 - **Blocking modals**
 - **Critical alerts** and confirmation dialogs
 - Typical opacity: **50–70%**
 - Example: `rgba(0, 0, 0, 0.6)` in light mode; `rgba(255, 255, 255, 0.32)` in dark mode
-

Example in CSS

CSS

 Copy code

```
.scrim-light {  
  background: var(--color-surface-overlay-scrim-light-background);  
  position: fixed;  
  inset: 0;  
}  
  
.scrim-dark {  
  background: var(--color-surface-overlay-scrim-dark-background);  
  position: fixed;  
  inset: 0;  
}
```

Light/Dark Mode Mapping Example

Token	Light Mode	Dark Mode
color.surface.overlay.scrim.light.background	rgba(0, 0, 0, 0.25)	rgba(255, 255, 255, 0.12)
color.surface.overlay.scrim.dark.background	rgba(0, 0, 0, 0.6)	rgba(255, 255, 255, 0.32)

If you want, I can **extend this scrim spec** to also include `motion.overlay.scrim.fade.duration` & `motion.overlay.scrim.fade.easing` tokens so the scrim transition is consistent across all overlays in your SaaS UI. That would make it truly ready for production-level design systems.



in figma is color + opacity tokens?

Yes — in **Figma** (and in many design systems), a scrim is usually defined as **two tokens**:

- 1. **Base color token** — pure color without opacity.
- 2. **Opacity token** — percentage or decimal value controlling transparency.

Why split them

- **Reusability:** The same base color can be used for shadows, overlays, or borders, just with different opacities.
- **Theme flexibility:** You can swap the base color for dark mode without touching all opacity values.
- **Accessibility tuning:** Opacity can be adjusted independently to meet visual clarity needs.

Scrim token example in Figma

Tokens

csharp

 Copy code

```
color.base.overlay.light = #000000  
opacity.overlay.light = 0.25
```

```
color.base.overlay.dark = #000000  
opacity.overlay.dark = 0.6
```

Computed in UI (Figma style)

ini

 Copy code

```
color.surface.overlay.scrim.light.background = rgba(#000000, 0.25)  
color.surface.overlay.scrim.dark.background = rgba(#000000, 0.6)
```

How it works in practice

- **Figma color style:** "Overlay / Scrim / Light" → linked to `color.base.overlay.light`
- **Opacity token:** applied in the layer settings or as part of the color definition
- **Dev handoff:** They see it as an `rgba()` or `hsla()` value.

If you want, I can rewrite your **overlay scrim tokens** so they're **base color + opacity pairs** in Figma naming style,
but still compile down to `rgba()` for developer handoff — that's how most advanced SaaS design systems handle scrim now.



in figma i use this `color.base.overlay.light` or this
`color.surface.overlay.scrim.light.background`

It depends on **what level** of your token system you're working in inside Figma.

Two levels to think about

1. Base (Tier 1) tokens

- `color.base.overlay.light`
- This is your **raw primitive** — just the flat color, without opacity or context.
- Lives in your **foundations** page in Figma.
- Never directly tied to a specific component type — it's generic.
- You can reuse it for scrim, shadows, or anything else needing an overlay color.

Example:

```
color.base.overlay.light = #000000
```

2. Semantic (Tier 2) tokens

- `color.surface.overlay.scrim.light.background`
- This is a **role-specific token** — it tells you *what it's for*, not just what it is.
- Lives in your **component library** or UI kit pages.
- Tied to a **specific usage**: the scrim background behind overlays.
- Can pull its value from `color.base.overlay.light` + an opacity token.

Example:

```
color.surface.overlay.scrim.light.background =  
rgba(color.base.overlay.light, opacity.overlay.light)
```

How to use them in Figma

- When you're **designing a component** (like a modal scrim), you should apply the **semantic token**:
`color.surface.overlay.scrim.light.background`
→ This way, if you ever change scrim color globally, all modals update automatically.
 - When you're **building or updating foundation styles**, you work with the **base token**:
`color.base.overlay.light`
→ This keeps your foundation clean and theme-ready.
-



Rule of thumb in Figma:

- **Base tokens** = used in **foundations**.

- **Surface/semantic tokens** = used in **real components**.
-

If you want, I can give you a **full Figma naming map** showing how `color.base.overlay.light` flows into `color.surface.overlay.scrim.light.background` via opacity tokens, so your DS is consistent from primitives → surfaces → components.



lets go and do and exmapli it better:

L) Inverse & High-contrast

Alright — let's break down **L) Inverse & High-contrast** so it's crystal clear, and I'll give you a **practical example mapping** from base → surface → component (Figma-friendly).

What "Inverse" means

Inverse tokens are for when **the background is dark** and **text/icons need to be light** — the opposite of your default theme.

You see these in:

- Dark toolbars on a light app
- Dark banners on a light page
- Dark sections of a dashboard
- Media overlays (e.g., text over an image)

They aren't just "dark mode" — they can exist in *any* theme.

What "High-contrast" means

High-contrast tokens are for **accessibility and visibility in extreme situations**:

- Users with high-contrast mode enabled (OS/Browser setting)
- WCAG AAA contexts

- Components where readability must override brand colors (e.g., alerts, loading screens)

Token Structure

Inverse

SCSS

 Copy code

```
color.text.inverse           // Text on dark background
color.icon.inverse           // Icons on dark background
color.border.inverse         // Borders on dark background
color.background.inverse     // Dark background for inverse areas
```

Surface-specific inverse

CSS

 Copy code

```
color.surface.button.inverse.background
color.surface.button.inverse.text
color.surface.button.inverse.icon
```

(Same naming can apply for chips, banners, etc.)

High-contrast

CSS

 Copy code

```
color.text.highContrast
color.icon.highContrast
color.border.highContrast
color.background.highContrast
```

Surface-specific high-contrast

CSS

 Copy code

```
color.surface.banner.highContrast.background
color.surface.banner.highContrast.text
color.surface.banner.highContrast.icon
```

Figma-friendly structure (base → opacity → surface)

Tier 1 (Base tokens):

csharp

 Copy code

```
color.base.white = #FFFFFF
color.base.black = #000000
```

Tier 1 (Opacity tokens):

ini

 Copy code

```
opacity.highContrast.background = 1
opacity.inverse.overlay = 0.9
```

Tier 2 (Surface tokens):

ini

 Copy code

```
color.surface.overlay.inverse.background = rgba(color.base.black, opacity.
color.surface.banner.highContrast.background = color.base.black
color.surface.banner.highContrast.text = color.base.white
```

Example Usage

Inverse toolbar in a light theme

CSS

 Copy code

```
.toolbar.inverse {
  background: var(--color-background-inverse); /* dark bg */
  color: var(--color-text-inverse);           /* light text */
}
```

Light mode mapping:

arduino

 Copy code

```
color.background.inverse = #111827
color.text.inverse = #FFFFFF
```

Dark mode mapping:

arduino

 Copy code

```
color.background.inverse = #FFFFFF
color.text.inverse = #111827
```

High-contrast banner

css

 Copy code

```
.banner.high-contrast {
  background: var(--color-background-highContrast); /* pure black or pure white */
  color: var(--color-text-highContrast); /* pure white or pure black */
}
```

Light mode mapping:

arduino

 Copy code

```
color.background.highContrast = #000000
color.text.highContrast = #FFFFFF
```

Dark mode mapping:

arduino

 Copy code

```
color.background.highContrast = #FFFFFF
color.text.highContrast = #000000
```

Why split into two

- **Inverse** is for brand/design-driven situations (style choice).
 - **High-contrast** is for accessibility-driven situations (compliance & clarity).
 - Both need **tokens** so you can apply them anywhere without hardcoding colors.
-

If you want, I can now **extend L)** to also include **interactive states** for inverse & high-contrast surfaces

(hover, active, selected) so your buttons, chips, and banners stay usable in those modes too.



but this is necessary if I am using the property colors and I got dark theme?

If your **dark theme is already designed and tokenized properly**, you don't *have* to maintain a full set of **inverse** tokens — but they're still useful in certain cases.

When you can skip inverse tokens

- If **every surface in your system** automatically switches to correct dark/light colors based on the global theme.
- If you **never** mix dark-on-light and light-on-dark surfaces in the *same theme*.
- If all background/text combinations are already covered by your **theme mappings**.

In that case, the dark theme itself already handles “inverse” automatically.

When inverse tokens are still helpful

Even with a proper dark theme, you sometimes need a **dark surface inside a light theme** or a **light surface inside a dark theme**.

Common cases in SaaS UIs:

- **Dark toolbar** on top of a light page (brand styling)
- **Light card** inside a dark dashboard section
- **Media overlays** (white text over images)
- **Marketing or promo sections** embedded in product UI
- **High-contrast toolbars** for specialized modes (e.g., code editors)

Inverse tokens let you style those *without* hacking in “dark theme” colors while the rest of the app stays in light mode.

High-contrast is separate

Even if you have a dark theme, **high-contrast tokens** are still valuable because:

- They're tied to **accessibility mode**, not theming.
 - OS/browser high-contrast modes can override colors unexpectedly unless you plan for it.
 - Some surfaces (alerts, safety-critical notices) should always use maximum contrast regardless of theme.
-

Bottom line

- If your app never needs "mixed theme" layouts → you can skip inverse tokens.
 - If your app only has normal + dark mode → inverse tokens may be overkill.
 - High-contrast tokens are **always** good to have for accessibility compliance.
-

If you want, I can give you a **decision chart** for when to use normal theme colors, inverse tokens, or high-contrast tokens so your team knows exactly when to apply each. That way you don't add complexity unless it's needed.



lets do M) Data / Charts (container roles)

Alright — let's define **M) Data / Charts (container roles)** so they fit your **flat-by-default** SaaS design system and cover all the container-level tokens needed for analytics, dashboards, and data-heavy UIs.

Purpose

These tokens define **container-level roles** for data visualizations — the **surfaces, grids, and supporting areas** that charts live in — not the colors of the data series themselves. This ensures:

- Consistent background/border/divider colors across all chart components.
 - Theme-ready containers for light/dark/high-contrast modes.
 - Clear separation between chart *framework* and *data visualization styles*.
-

Token Structure

1. Chart Area (primary plotting surface)

CSS

 Copy code

```
color.surface.chart.background
color.surface.chart.border
color.surface.chart.grid.major
color.surface.chart.grid.minor
color.surface.chart.axis
color.surface.chart.axis.text
```

Usage:

- `background` — Fill under the plot
 - `border` — Container outline (if any)
 - `grid.major` — Major reference lines
 - `grid.minor` — Minor reference lines
 - `axis` — Axis line color
 - `axis.text` — Labels for axes
-

2. Chart Panel (outer container)

CSS

 Copy code

```
color.surface.chart.panel.background
color.surface.chart.panel.border
color.surface.chart.panel.title
color.surface.chart.panel.subtitle
```

Usage:

- Wraps the chart + controls + legend
 - Often matches `container` role but can have variant backgrounds for analytics sections.
-

3. Legend Area

CSS

 Copy code

```
color.surface.chart.legend.background
color.surface.chart.legend.border
color.surface.chart.legend.text
color.surface.chart.legend.icon
```


Usage:

- Background and borders for the legend block
 - Can be inline or floating over the chart
-

4. Tooltip / Hover Info

(Allowed to use light elevation here — true overlay)

CSS

 Copy code

```
color.surface.chart.tooltip.background
color.surface.chart.tooltip.border
color.surface.chart.tooltip.text
color.surface.chart.tooltip.icon
shadow.surface.chart.tooltip
```

Usage:

- Hover pop-ups for data points
 - Border + shadow to ensure readability over any chart color
-

5. States / Variants

For interactive chart containers:

arduino

 Copy code

```
color.surface.chart.hover.background
color.surface.chart.selected.background
color.surface.chart.disabled.background
```

Light Theme Mapping Example

Token	Value
chart.background	#FFFFFF
chart.border	#E5E7EB
chart.grid.major	#D1D5DB
chart.grid.minor	#E5E7EB
chart.axis	#9CA3AF
chart.axis.text	#374151
chart.tooltip.background	#FFFFFF
chart.tooltip.border	#D1D5DB

Dark Theme Mapping Example

Token	Value
chart.background	#1F2937
chart.border	#374151
chart.grid.major	#4B5563
chart.grid.minor	#374151
chart.axis	#6B7280
chart.axis.text	#D1D5DB
chart.tooltip.background	#111827
chart.tooltip.border	#374151

Why container tokens matter

- Prevents each chart library from making its own “light gray” guess.

- Lets you change the entire dashboard's data presentation look by remapping one set of tokens.
- Makes **theming consistent** whether you're using Chart.js, Recharts, D3, or custom SVG.

If you want, I can extend **M)** to also include **data series color tokens** so your charts have a standardized palette for lines, bars, and fills — instead of leaving it up to the library defaults. That's the next logical step after defining container roles.



and percentage on increa and decrease on card with data?

Gotcha — you're talking about the **percentage change indicators** you see on KPI/data cards,

like **+12%** in green for growth or **−8%** in red for decline.

This actually fits perfectly under **M) Data / Charts** because these percentage labels are **data-related surface/text roles**, not chart series colors.

Token structure for % change indicators

Base

SCSS

 Copy code

```
color.data.change.increase.text
color.data.change.increase.icon
color.data.change.increase.background // optional, if you put the % in a

color.data.change.decrease.text
color.data.change.decrease.icon
color.data.change.decrease.background

color.data.change.neutral.text // for 0% change or no significant
```

```
color.data.change.neutral.icon
color.data.change.neutral.background
```

Usage examples

Inline text only

Example:

Revenue: \$42,000 (+12%)

- % text uses color.data.change.increase.text (green)
- Optional ▲/▼ icon uses color.data.change.increase.icon

Badge-style

Example:

Orders: 510

[▲ 8%] (green pill)

- Background = color.data.change.increase.background (light green)
- Icon/Text = color.data.change.increase.icon / .text

Light theme mapping example

Token	Value
increase.text	#065F46
increase.icon	#065F46
increase.background	#D1FAE5
decrease.text	#991B1B
decrease.icon	#991B1B
decrease.background	#FEE2E2
neutral.text	#6B7280
neutral.icon	#6B7280

Token	Value
neutral.background	#F3F4F6

Dark theme mapping example

Token	Value
increase.text	#6EE7B7
increase.icon	#6EE7B7
increase.background	#064E3B
decrease.text	#FCA5A5
decrease.icon	#FCA5A5
decrease.background	#7F1D1D
neutral.text	#9CA3AF
neutral.icon	#9CA3AF
neutral.background	#374151

Accessibility tips

- Icons + colors for direction (▲/▼) help **colorblind users** — don't rely on color alone.
- If the percentage is clickable (e.g., to show a trend chart), also define hover/focus styles:

arduino

 Copy code

color.data.change.increase.hover.background
color.data.change.decrease.hover.background

If you want, I can make a **full KPI card spec** that ties together:

- **Surface tokens** (card background, border)

- **Data tokens** (number, label, % change)
- **State tokens** (hover/focus if interactive)

so your analytics cards look and behave consistently across the product.

< 1/2 >      ...

make the final and the most complet for data and chart tokens

Thought for a few seconds >

Absolutely—here's a **final, production-grade, Tier-2 token spec for Data & Charts**. It's split into clear sections so you can drop it into your design system. These are **semantic roles** (Tier-2) that you'll map to Tier-1 primitives (raw colors, sizes, opacities) per theme.

M) Data & Chart Tokens — Final

1) Chart Containers (surfaces around data)

CSS

 Copy code

```
color.surface.chart.background
color.surface.chart.border
color.surface.chart.panel.background
color.surface.chart.panel.border
color.surface.chart.panel.title
color.surface.chart.panel.subtitle

color.surface.chart.legend.background
color.surface.chart.legend.border
color.surface.chart.legend.text
color.surface.chart.legend.icon

color.surface.chart.tooltip.background
color.surface.chart.tooltip.border
color.surface.chart.tooltip.text
color.surface.chart.tooltip.icon
shadow.surface.chart.tooltip
```

```
color.surface.chart.hover.background  
color.surface.chart.selected.background  
color.surface.chart.disabled.background
```

2) Axes, Ticks & Grid

CSS

 Copy code

```
color.chart.axis.line  
color.chart.axis.text  
color.chart.axis.tick  
color.chart.grid.major  
color.chart.grid.minor
```

3) Core Series Palette (categorical)

(Use as your default category palette for lines/bars/pies. Expand/contract as needed.)

CSS

 Copy code

```
color.chart.series.1  
color.chart.series.2  
color.chart.series.3  
color.chart.series.4  
color.chart.series.5  
color.chart.series.6  
color.chart.series.7  
color.chart.series.8  
color.chart.series.9  
color.chart.series.10  
color.chart.series.11  
color.chart.series.12
```

Series roles (semantic mapping for common patterns)

SCSS

 Copy code

```
color.chart.series.positive // e.g., gains  
color.chart.series.negative // e.g., losses
```

```
color.chart.series.neutral
color.chart.series.highlight // callout/selected series
color.chart.series.comparison // secondary/benchmark series
color.chart.series.forecast // predicted values
color.chart.series.baseline // zero line / goal series
```

4) Quantitative Scales (sequential & diverging)

(For heatmaps, choropleths, intensity fills. Stops are semantic; map to Tier-1 scales.)

Sequential (low → high)

CSS

 Copy code

```
color.scale.sequential.0
color.scale.sequential.1
color.scale.sequential.2
color.scale.sequential.3
color.scale.sequential.4
color.scale.sequential.5
color.scale.sequential.6
color.scale.sequential.7
color.scale.sequential.8
color.scale.sequential.9
```

Diverging (negative → neutral → positive)

CSS

 Copy code

```
color.scale.diverging.neg.4
color.scale.diverging.neg.3
color.scale.diverging.neg.2
color.scale.diverging.neg.1
color.scale.diverging.neutral
color.scale.diverging.pos.1
color.scale.diverging.pos.2
color.scale.diverging.pos.3
color.scale.diverging.pos.4
```

5) Change Indicators (% change on KPI cards, badges)

(Pairs nicely with your cards.)

CSS

 Copy code

```
color.data.change.increase.text
color.data.change.increase.icon
color.data.change.increase.background // if shown as a badge/pill

color.data.change.decrease.text
color.data.change.decrease.icon
color.data.change.decrease.background

color.data.change.neutral.text
color.data.change.neutral.icon
color.data.change.neutral.background

color.data.change.increase.hover.background
color.data.change.decrease.hover.background
```

6) Data Marks (lines, bars, points, areas)

(Visual primitives across chart types.)

CSS

 Copy code

```
color.chart.mark.line
color.chart.mark.bar
color.chart.mark.point
color.chart.mark.area.fill
color.chart.mark.area.stroke
```

States:

SCSS

 Copy code

```
color.chart.mark.hover
color.chart.mark.selected
color.chart.mark.muted // de-emphasized (others highlight)
```

Quantities (Tier-2 sizing tokens referencing Tier-1 sizes):

arduino

 Copy code

```
size.chart.stroke.thin
size.chart.stroke.medium
size.chart.stroke.thick
size.chart.point.radius.sm
size.chart.point.radius.md
size.chart.point.radius.lg
```

Dash patterns (semantic):

arduino

 Copy code

```
pattern.chart.stroke.solid
pattern.chart.stroke.dashed
pattern.chart.stroke.dotted
```

Opacities (semantic):

CSS

 Copy code

```
opacity.chart.area.fill
opacity.chart.band.fill
opacity.chart.muted
```

7) Bands, Confidence & Forecasts

arduino

 Copy code

```
color.chart.band.fill           // confidence interval fill
color.chart.band.stroke
color.chart.forecast.line
opacity.chart.forecast.line
```

8) Thresholds, Reference & Targets

CSS

 Copy code

```
color.chart.reference.line
color.chart.reference.text
color.chart.threshold.low
color.chart.threshold.high
```

```
color.chart.goal.line  
color.chart.goal.text  
pattern.chart.reference.dashed
```

9) Selection, Focus & Interaction

css

 Copy code

```
color.chart.interaction.focusRing  
color.chart.interaction.brush.area // drag-to-zoom selection area  
color.chart.interaction.brush.border  
color.chart.interaction.crosshair.line
```

10) Sparklines & Micro-charts

css

 Copy code

```
color.chart.sparkline.line  
color.chart.sparkline.fill  
color.chart.sparkline.positive  
color.chart.sparkline.negative
```

11) Heatmaps & Density

css

 Copy code

```
color.chart.heatmap.cell.low  
color.chart.heatmap.cell.mid  
color.chart.heatmap.cell.high  
color.chart.heatmap.cell.border
```

12) Maps (choropleth, point maps)

arduino

 Copy code

```
color.chart.map.land  
color.chart.map.water  
color.chart.map.border  
color.chart.map.region.fill.low  
color.chart.map.region.fill.mid  
color.chart.map.region.fill.high  
color.chart.map.point          // markers  
color.chart.map.point.hover
```

13) Progress & Gauges

CSS

 Copy code

```
color.chart.progress.track  
color.chart.progress.fill  
color.chart.gauge.track  
color.chart.gauge.fill.ok  
color.chart.gauge.fill.warn  
color.chart.gauge.fill.critical
```

14) Layout & Spacing (chart padding & gaps)

pgsql

 Copy code

```
space.chart.padding.outer  
space.chart.padding.inner  
space.chart.gap.series  
space.chart.gap.category
```

15) Motion (respect "reduce motion")

 Copy code

```
motion.chart.duration.fast  
motion.chart.duration.medium  
motion.chart.duration.slow  
motion.chart.easing.standard
```

```
motion.chart.easing.decelerate
motion.chart.easing.accelerate
```

16) Accessibility & Modes

css

 Copy code

```
color.chart.a11y.highContrast.background
color.chart.a11y.highContrast.text
color.chart.a11y.highContrast.axis
color.chart.a11y.focusRing
```

Implementation Notes

- **Tier-1 mapping:** All the tokens above should **reference** your base palette (neutrals, brand, semantic status colors), size scales, and opacities. E.g.:
 - `color.chart.grid.major = color.neutral.300`
 - `size.chart.stroke.medium = size.2`
 - `opacity.chart.area.fill = opacity.40`
 - **Light/Dark:** Provide a mapping per theme; keep **axis text** sharply legible; gridlines lighter than axes; tooltip uses **overlay** rules (border + shadow).
 - **Consistency:** Use **one** forecast line style + one confidence band style across all charts.
 - **States:** Prefer **color shift + stroke width** over drop shadows for hover/selected in charts (modern, crisp).
-

Minimal JSON Skeleton (example)

(Values are placeholders; wire them to your Tier-1 tokens.)

json

 Copy code

```
{
  "color": {
    "surface": {
      "chart": {
        "background": "{color.neutral.0}",
```

```
"border": "{color.neutral.200}",  
"panel": {  
  "background": "{color.neutral.0}",  
  "border": "{color.neutral.200}",  
  "title": "{color.neutral.900}".
```