Design Systems & Design Tokens — Notes in Blog Style

A structured, “panel-style” write-up of your original notes (everything preserved, just organized).

2□ What problem design systems solve (core idea)

Jina makes one very important clarification early:

> Design systems are NOT just code or UI libraries.
>
> They solve three big problems:
> □ Consistency — same buttons, spacing, colors everywhere (no “why does this screen look different?”)
> □ Maintainability — change once → update everywhere (no copy-paste chaos)
> □ People & process problems — designers & developers misunderstand each other; poor docs cause friction.
> □ Design systems fix how people work together, not just how things look.

4□ What design systems look like in practice (important clarification)

Most people think:

> “Our design system = a website with components”
>
> Jina says:
> □ That website is NOT your design system
> □ It is just a representation of it

The real design system is:

- How things are named
- What values exist
- Why decisions were made
- How changes flow through the system
- How teams use it

Mindset shift

A design system is a living operating system for decisions — not a component gallery.

5 Design systems ≠ only web

She reminds us: design systems existed long before web apps:

- NASA graphic standards
- Print & physical products
- Branding systems

Digital just made them more visible.

6 Where design tokens come in (THIS IS THE CORE)

Now we reach the most important part. Her problem at Salesforce:

- She designed for web, iOS, Android
- Web was easy → she could code
- Native apps were painful → endless redlines
- Every spacing, font, color had to be manually specified
- Changes meant redoing EVERYTHING

This is the pain that created design tokens.

Tokens exist because cross-platform scaling without shared decisions is chaos.

7 What design tokens actually are (simple explanation)

Jina explains it indirectly through experience:

Definition

Design tokens are design decisions stored as data, not visuals.
Examples: Colors, Spacing, Typography, Radius, Motion values.
Stored once → transformed automatically for: Web (CSS), iOS, Android, any framework/language.
Tokens are not just variables — they are cross-platform shared truth.

9□ The BIG superpower of design tokens

Tokens make enterprise-level design possible

Accessibility fixes (contrast issues)
Theming
White-labeling
Dark mode
Density modes (compact / cozy / comfortable)
All by changing tokens — not components.
That’s enterprise-level design.

1□1□ Why W3C exists (standards part)

She explains this very clearly:

The key point

No one wants to force: JSON vs YAML, Sass vs CSS vs XML — products are too different.
So W3C focuses on: □ Concepts □ Relationships □ Structure □ APIs
This allows Figma, Adobe, InVision, Dev tools… to all speak the same language.

------ that was intro to tokens

Now we shift from understanding tokens → how to use them strategically.

3□ Why NOT start with a component library?

This is subtle but very important:

The article says

Full component libraries are hard — even experienced teams struggle.
They require alignment, tooling, documentation, governance.
So instead of “Let’s build a massive component system” you start with:
“Let’s agree on our design decisions first.”
➡ That’s what tokens are: Incremental • Low-risk • High-impact.

4□ Jina Anne’s core idea (bottom-up, not top-down)

Crucial line

“With Design Tokens, a design system can be built from the bottom up and in a technology-agnostic manner.”

What that really means:

□ Old way (top-down)

Pick framework → build components → lock decisions into code → hard to change later.

□ Token way (bottom-up)

Define decisions as data → feed them into any framework.
Web, iOS, Android, React, Vue — doesn’t matter.
□ Tokens don’t care about technology. Technology consumes tokens.

7□ Why tokens force better teamwork (VERY IMPORTANT)

What happens with tokens

Designers must think clearly about decisions and understand some code logic (no “just eyeballing”).
Developers learn why decisions exist and stop guessing intent.
➡ Tokens force consensus.
Key line: “They bridge the semantic gap between designers’ intentions and technical implementation”
That’s S-type thinking.

8□ “Powerful levers” (this is the lever concept)

Tokens give you different levels of control:

- Brand level → affects everything
- Theme level → affects groups
- Component level → affects specific UI

- Instance level → affects one element

This is why tokens are often visualized as layers or trees.

9⃣ Tree hierarchy explanation (the SCSS example)

This part explains how tokens are structured.

Layer 1 — Subatomic (raw values)

```
$color-cornflower: #5784FF;
```

This is just a fact. No meaning.

Layer 2 — Semantic meaning

```
$color-primary: $color-cornflower;
```

Now we give intent: “This is our primary color”.

Layer 3 — UI mapping

```
$color-button-cta: $color-primary;
```

Now we define usage: “This is what CTA buttons use”.

 Why this matters

This is exactly the multi-tier thinking you were confused about earlier.

Why this hierarchy is powerful (the “levers”)

The border-radius example explains this beautifully

Change root token → entire UI changes
Change mid-level token → group changes
Change component token → single element changes
The closer you are to the root → bigger impact
The closer you are to the leaf → safer change
That’s intentional design.

1️⃣1️⃣ “Semantic guardrails” (VERY IMPORTANT TERM)

Professional-level insight

Because tokens are semantic:
You can’t misuse them easily
Bad usage becomes obvious
Example: Using color-button-cta for paragraph text feels wrong
If a token doesn’t exist, it signals a system gap
➡ The system self-governs (this is why large teams love tokens)

1️⃣2️⃣ Cross-functionality & collective ownership

Final takeaway

Design Tokens are not owned by “design” or “dev”.
They are shared infrastructure, encourage collaboration, and reduce silos.
This line sums it up: “Design systems are for people.”
Not tools. Not Figma. Not React.

One-paragraph simplified summary

Summary

Design Tokens are structured, semantic design decisions stored as data.
They allow teams to build design systems bottom-up, scale across platforms, and collaborate more effectively.
By separating raw values from meaning and usage, tokens create consistency, flexibility, and safe ways to change design without breaking products.
They are the DNA of a lean design system.

Reference link

```
https://www.figma.com/design/0eUdyUjldFV6lJhjneqAqi/Shopify-Polaris-Design-System-Exerci
se--Community-?node-id=84679-15&t;=1vLCgFfEbFF3cLtP-1
```

More important things (your extended notes)

Great question

You’re exactly at the point where most designers mess up—so slowing down here is the right move.
The real systems question: “How do I name and structure tokens so this scales across hundreds of screens?”
That’s S-type designer territory.

Reset your mental model

□ Tokens are NOT for screens
□ Tokens are NOT for components
□ Tokens are for decisions
If you map tokens to screen/feature names, your system will collapse.

3-layer hierarchy

1. Primitive (raw values)
2. Semantic (meaning)
3. Component / Mapped (usage)

Primitive tokens

□ color.gray.100 = #FFFFFF
□ space.4 = 4
□ radius.2 = 2
□ Never: white / buttonBlue / cardRadius
Rule: describe what they are, not where they’re used.

Semantic tokens

□ color.text.primary / secondary / inverse
□ color.surface.default / elevated / warning
□ color.border.default / subtle / strong
Example: color.text.primary → color.gray.900
Rule: no screen/feature/component names.

Mapped / component tokens

▯ color.button.primary.text / background
▯ color.input.text / background / border
▯ color.alert.warning.text / background
Example: color.button.primary.text → color.text.inverse
Screens should consume tokens, not define them.

Naming convention cheat sheet

Structure: {category}.{entity}.{variant}.{property}
Examples: color.text.primary • color.button.primary.background • space.layout.pagePadding • radius.card.default
Keep it predictable. Keep it boring. Boring = scalable.

Retrofit steps

1) Audit colors (text/surface/border/feedback)
2) Define semantics first (~10–15 tokens)
3) Replace screen-specific styles gradually
4) Add mapped tokens only when repetition appears

Three-tier approach: Brand → Alias → Mapped

Brand tokens

brand.color.blue.500 / brand.space.8 / brand.radius.4
Rules: numeric scales only; no UI words; no semantic words.
Notes: rarely used directly; changes mainly during rebranding.

Alias tokens

alias.color.text.primary / alias.color.surface.default / alias.color.border.subtle
Brand → Alias examples:
alias.color.text.primary → brand.color.gray.900
Alias never references components; changes with theming/dark mode/accessibility.

Mapped tokens

mapped.button.primary.background / mapped.input.border / mapped.alert.warning.text
Alias → Mapped examples:
mapped.button.primary.background → alias.color.surface.brand
Mapped evolves with UI; can be deleted; absorbs churn safely.

End-to-end example

brand.color.blue.500 = #3B82F6
alias.color.surface.brand → brand.color.blue.500
mapped.button.primary.background → alias.color.surface.brand
Change brand? update brand token. Change meaning? update alias. Change component? update mapped.

Branded House vs House of Brands

Branded House: brand+alias shared; mapped per product → consistency & governance.
House of Brands: alias/mapped stay same; brand layer swaps per brand → same UI logic, different identity.