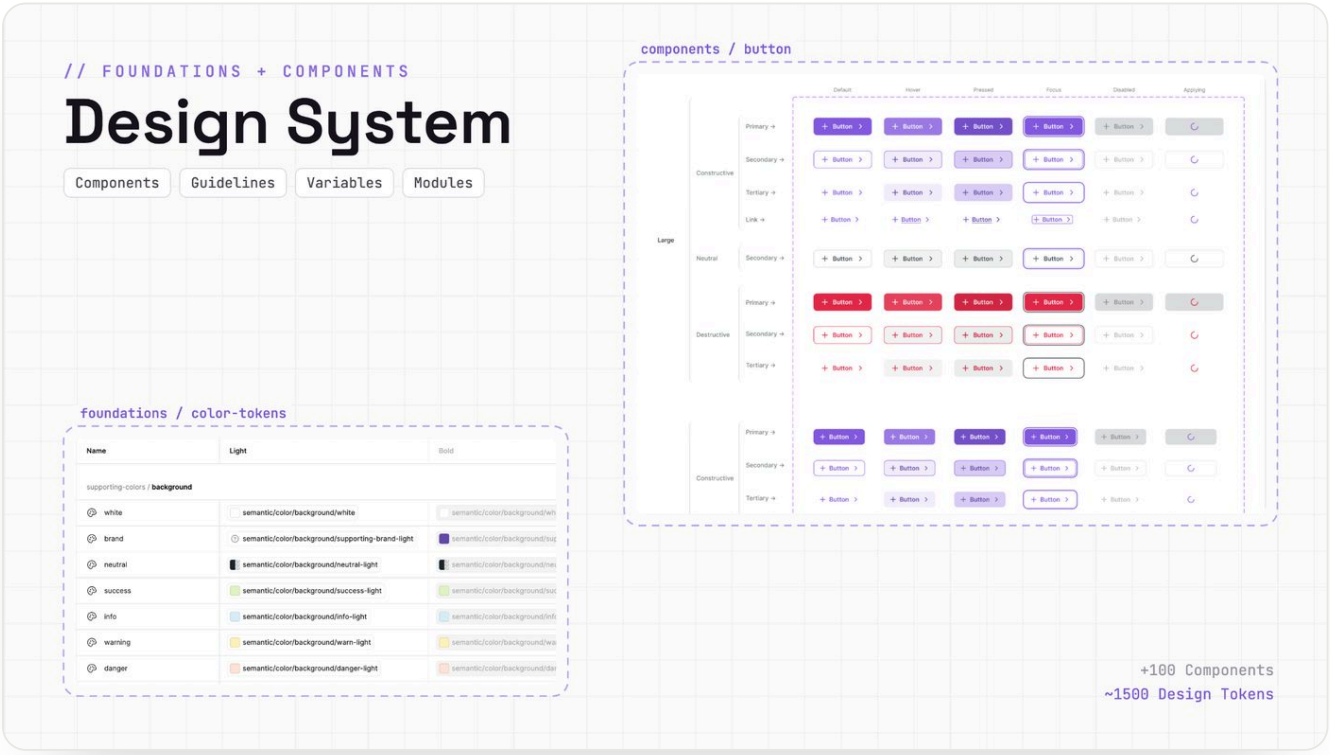
CASE STUDY · DESIGN SYSTEMS

Design System

Rebuilding a component library into tokenized infrastructure for three products, so a whole team could contribute safely instead of queueing behind one owner.

ROLE	TIMELINE	CONTEXT
Product Designer	12 months	B2B SaaS, confidential

~1,400	30+	3	24%
tokens, up from ~300	components built or rebuilt	products themed, one-click	fewer tokens across forms



Foundations and components at a glance: token layers feeding a full button system across states and variants.

Overview

This is the private design system behind three B2B SaaS products at the company. When I inherited it, it was a partially built library that only one person could change, so every component request queued behind that person, sometimes for weeks. Over the next year I rebuilt it into shared infrastructure. It now runs on a three tier token architecture that grew from roughly 300 to about 1,400 variables, more than 30 accessible components, one click theme switching across all three products, and a custom pipeline that exports design tokens straight to code. By the end, three designers were contributing safely under versioned releases, and design and development were finally building from the same source of truth.

Context and my role

The company builds scheduling and business management software for a set of related service industries. Its three products share one design language, and this system is what carries it.

A senior designer started the system and owned it end to end. During that period I worked mostly on product design and, like everyone else, requested component changes and waited. When that designer left, I took ownership. I completed what existed, rebuilt what did not hold up, and extended it into a full token system. I worked solo for the first five months. Over the following seven, two other designers contributed components and audits while I led the architecture and all of the tokenization, reviewing key decisions with the engineering lead, who was also my manager.

The problem

By the time I took over, four problems had compounded into one slow, inconsistent workflow.

One owner, everyone blocked

Any change at the component level went through a single person who was also busy with product work, so even small adjustments could stall a project for weeks.

Inconsistent by default

Coverage was partial, and many components lived in individual files instead of the shared library. Two designers working on the same feature could produce two different versions of the same form, because nothing pushed them toward a common part.

Design and code had drifted

Engineering maintained its own component library, built by eye from mockups. The components were close but not exact, and accessibility varied from one to the next.

Behaviour was undocumented

States, interactions, and accessibility expectations were never written down, so they were renegotiated at every handoff. That produced bugs for the team and confusion for end users.

For the business, this meant slow delivery, an inconsistent experience across three products that are meant to feel like siblings, and senior people spending their time rebuilding checkboxes.

DECISION 1

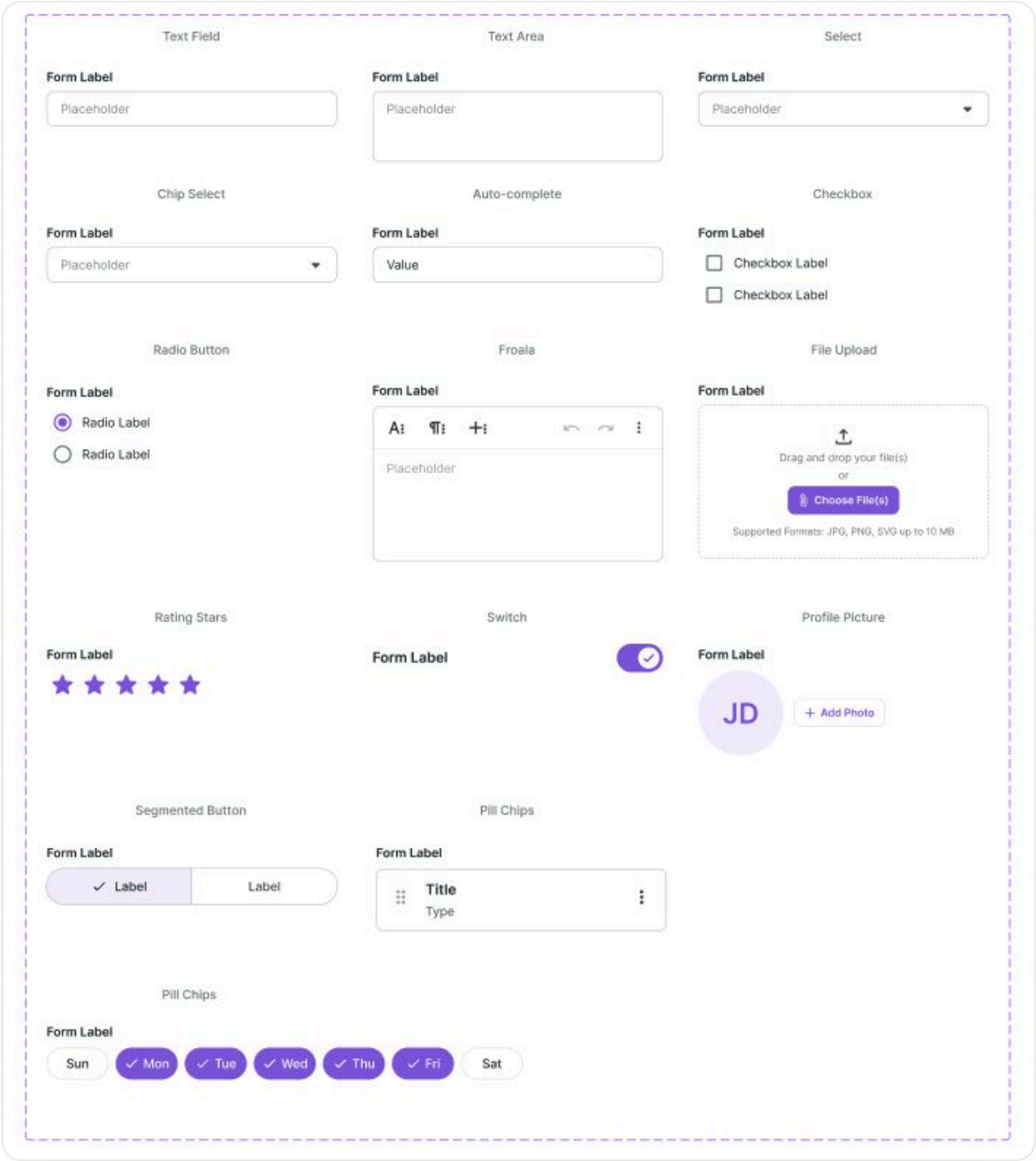
One form component instead of fifteen

Forms were the worst offender, so I started there. Building a form used to mean assembling separate components, each with its own tokens and its own quirks: a single line input, a multi line input, a checkbox, a radio button, and so on.

Working with the engineering lead, we landed on a single Form Field Container. It is one component with nested layers, where you pick the control you need from one

place: text field, text area, checkbox, radio, autocomplete, switch, chip select, file upload, and more.

The numbers made the case on their own. The old approach was 15 separate components at roughly 25 tokens each, about 375 variables in total. The container is one component with 14 variants and 284 tokens, a 24 percent reduction with more capability rather than less. Because engineering mirrors the same container structure in code, design and development stopped diverging at the exact place they used to diverge most.



One Form Field Container, fourteen controls: every form input in the system assembled from a single component.

DECISION 2

Three tiers of tokens

The heart of the system is its token architecture, organized in three layers across six variable collections. Primitives hold raw values such as the full colour ramps, spacing steps, radii, and sizes. Semantic tokens give those values meaning and intent. Alias tokens map semantics onto specific component properties.

LAYER 1

Primitives

Raw values: the full colour ramps, spacing steps, radii, and sizes. Everything resolves down to these.

LAYER 2

Semantic tokens

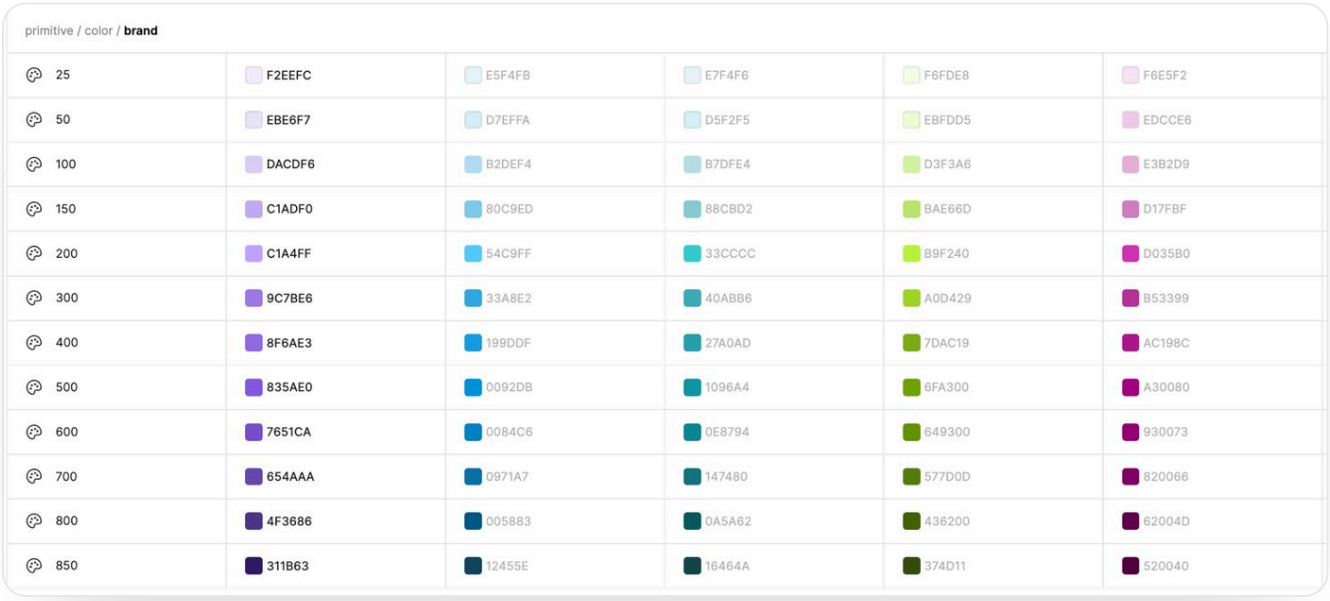
Give those raw values meaning and intent, e.g. a hover state or a background role.

LAYER 3

Alias tokens

Map semantics onto specific component properties, so a single component can carry an exception without moving anything else.

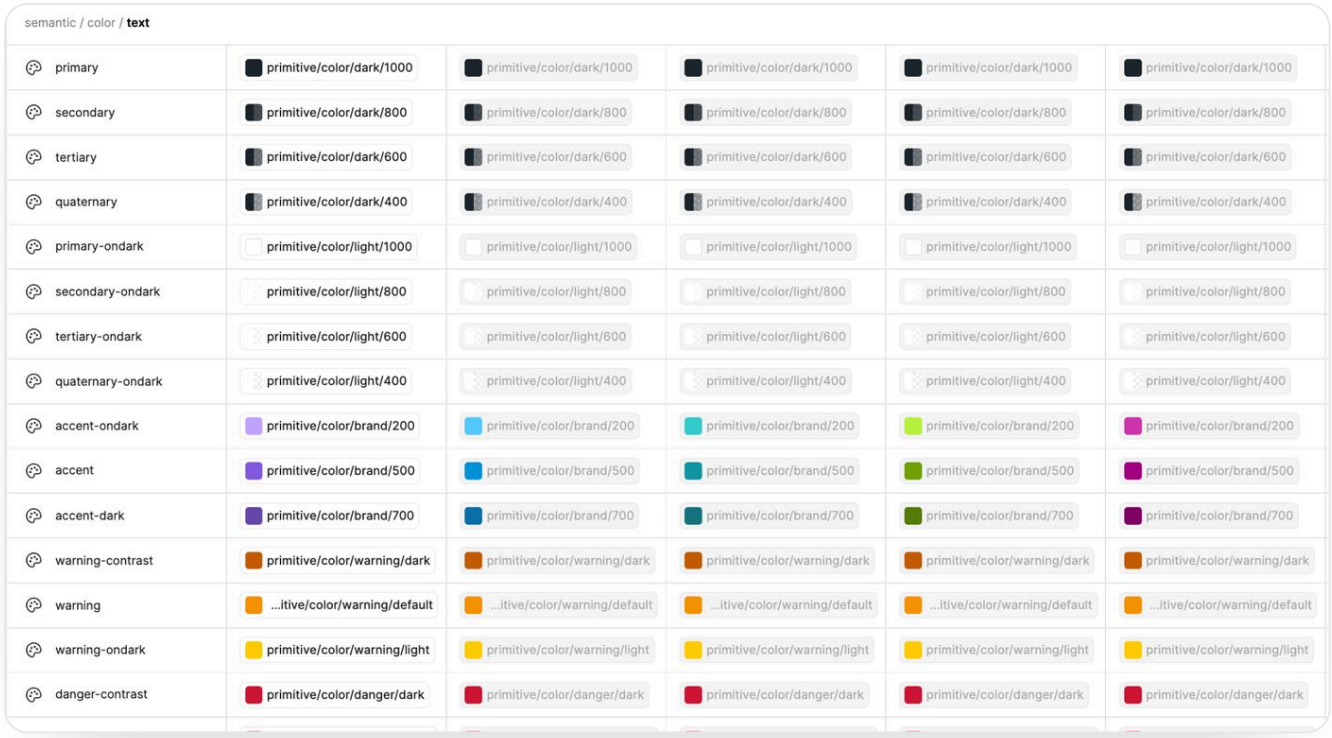
The point of the layering is that change happens at the right altitude. If we decide border colours need better contrast, we adjust one semantic token and it propagates through every component that uses it. If a single component needs an exception, we change its alias token and nothing else moves. That is what makes the system safe to scale.



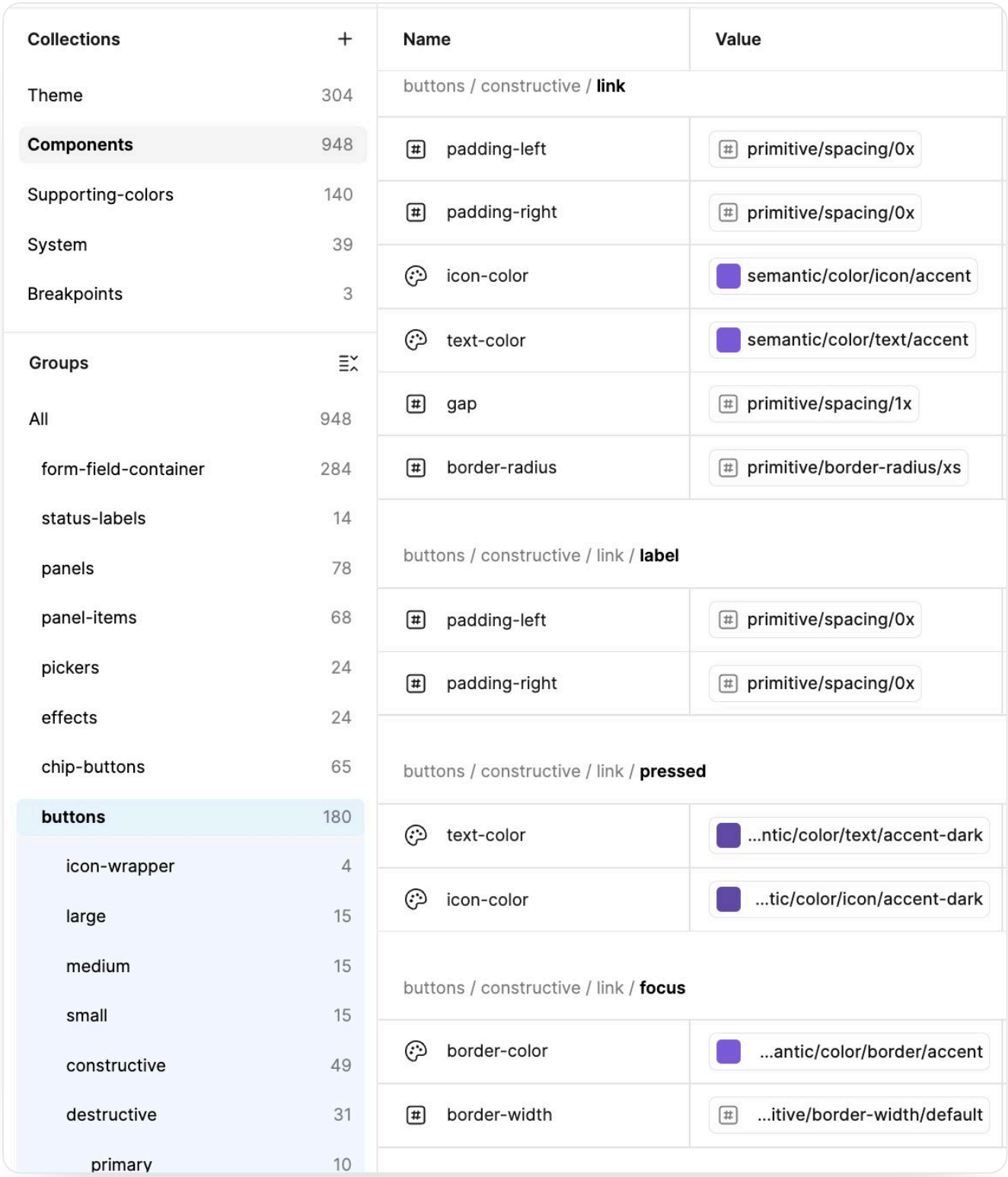
primitive / color / **brand**

25	F2EEFC	E5F4FB	E7F4F6	F6FDE8	F6E5F2
50	EBE6F7	D7EFFA	D5F2F5	EBFDD5	EDCCE6
100	DACDF6	B2DEF4	B7DFE4	D3F3A6	E3B2D9
150	C1ADF0	80C9ED	88CBD2	BAE66D	D17FBF
200	C1A4FF	54C9FF	33CCCC	B9F240	D035B0
300	9C7BE6	33A8E2	40ABB6	A0D429	B53399
400	8F6AE3	199DDF	27A0AD	7DAC19	AC198C
500	835AE0	0092DB	1096A4	6FA300	A30080
600	7651CA	0084C6	0E8794	649300	930073
700	654AAA	0971A7	147480	577D0D	820066
800	4F3686	005883	0A5A62	436200	62004D
850	311B63	12455E	16464A	374D11	520040

Primitive colour ramps: the raw values everything resolves to.



Semantic state tokens mapping intent onto primitives, per theme.



Component level alias tokens: the layer that lets one component carry an exception without touching the rest.

I grew the token system from roughly 300 variables to about 1,400, and connected type, sizing, and spacing styles to variables so nothing floats outside the tokens. Theming sits on top of this: one theme per product, plus dedicated staging and development modes, each with its own colour scheme, so new tokens are proven on safe themes before they reach a product. Because every component resolves through the token chain, switching a product's entire theme takes one click.

DECISION 3

A naming system the team could predict

Naming sounds cosmetic until several people are contributing to the same 1,400 variables. We spent several working sessions converging on a grammar for the alias level: `component, sub component, state, device, then property`. Each alias points at a semantic state token, which resolves to a primitive value, so the chain is traceable from component back to raw value.

WHY IT MATTERS

Tokens are an interface. If a designer or a developer can guess a token's name before searching for it, the system teaches itself. That predictability paid off directly when new contributors joined.

DECISION 4

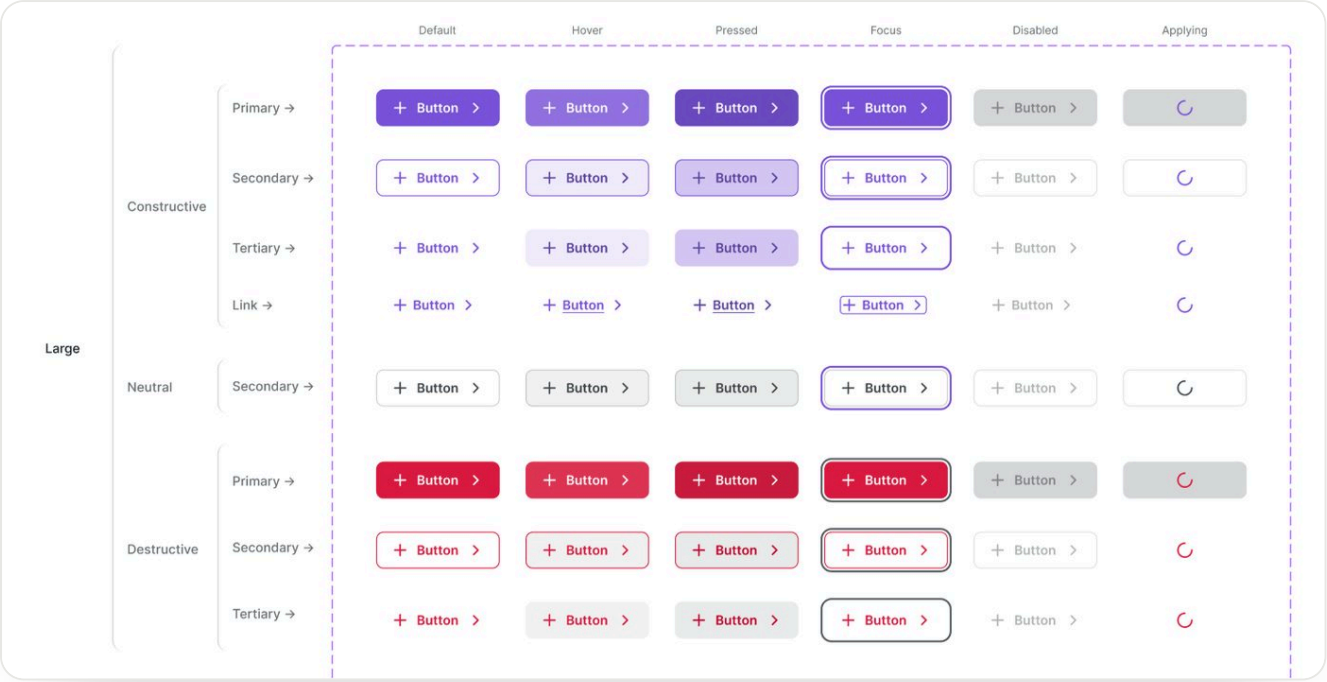
A handoff built for the team's constraints

One constraint shaped the handoff work. The team had decided against relying on live spec tooling, reasoning that static specs tempt developers to override shared components to match a mockup. I offered a different view, since automating spec upkeep reduces manual work for designers, but the call stood, so I designed the strongest pipeline I could within it.

Component specs. I built a dedicated spec file with redlined documentation for every component, covering all variants and exactly which tokens are assigned to which properties, so engineers implement from the system rather than from screenshots.

Tokens straight to code. I audited export options end to end, testing several output formats against how our stack actually consumes them. Off the shelf tools came close but could not meet our specifics, so we built our own export tooling with the engineering lead, using AI assistance for the underlying logic and

interface. A token change now reaches code without manual translation, which is what keeps a token system honest.



The button system: six states across nine variants, every combination resolving through the token chain.

DECISION 5

Governance for more than one owner

The system's original failure mode was single ownership, and I did not want to recreate it with myself in the middle. As the two other designers began contributing, I introduced versioning and release notes on every library publish. Each release records what changed and who changed it, so when a bug appears we trace it to a release instead of guessing. The system that had been one person's territory became infrastructure the whole team can change safely.

Accessibility as a default, not a pass

Every component I added or rebuilt follows WCAG 2.2: colour contrast checked, states exposed to screen readers, full keyboard navigation, alt text, and touch targets sized for real hands across devices. Where a pattern had a strong precedent in Material 3 or Apple's Human Interface Guidelines, I aligned with it so

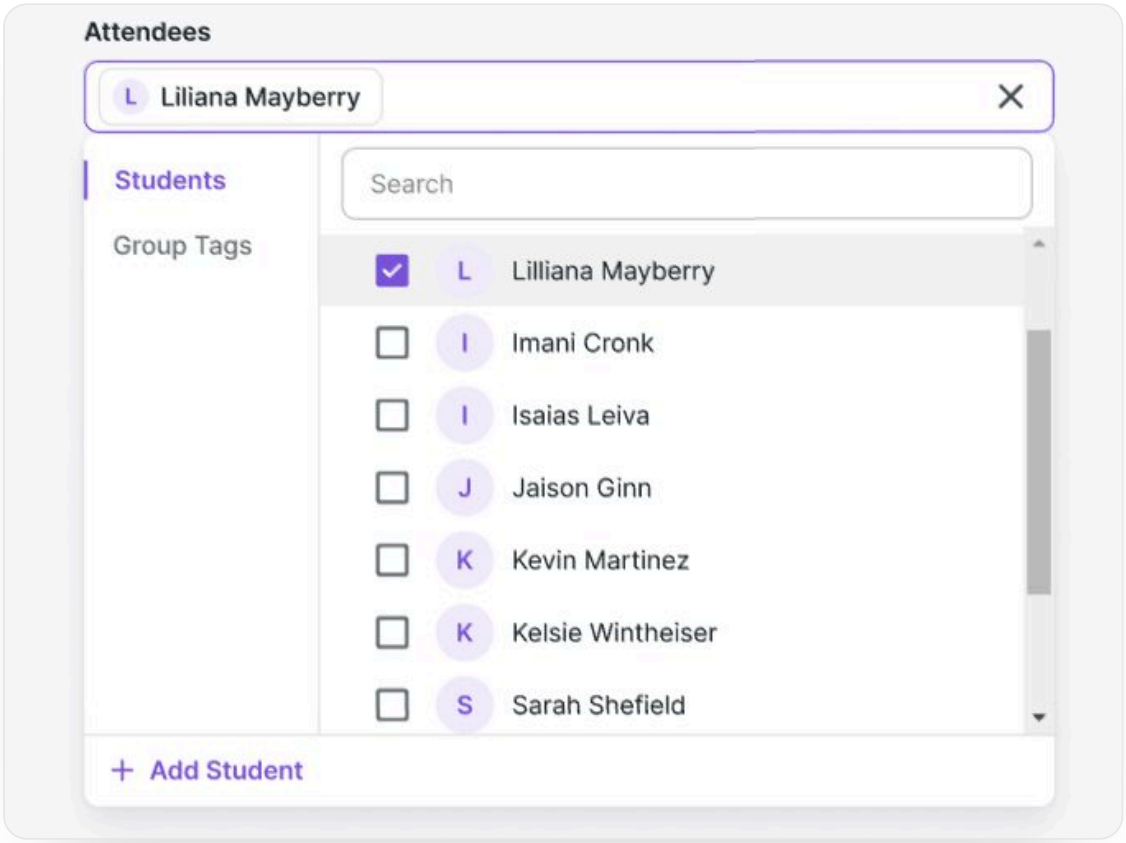
behaviour feels familiar rather than clever. Documented, consistent states also fixed the old handoff problem, since interaction behaviour is now defined once in the system instead of renegotiated on every project.

The system in use

A few highlights from the 30 plus components I designed or rebuilt.

A combined control panel

A single panel that brings together a sidebar, search, and an action bar. It exists to keep users in flow, so a user partway through creating a record who realizes a related record does not exist yet can add it without abandoning the task they are halfway through.

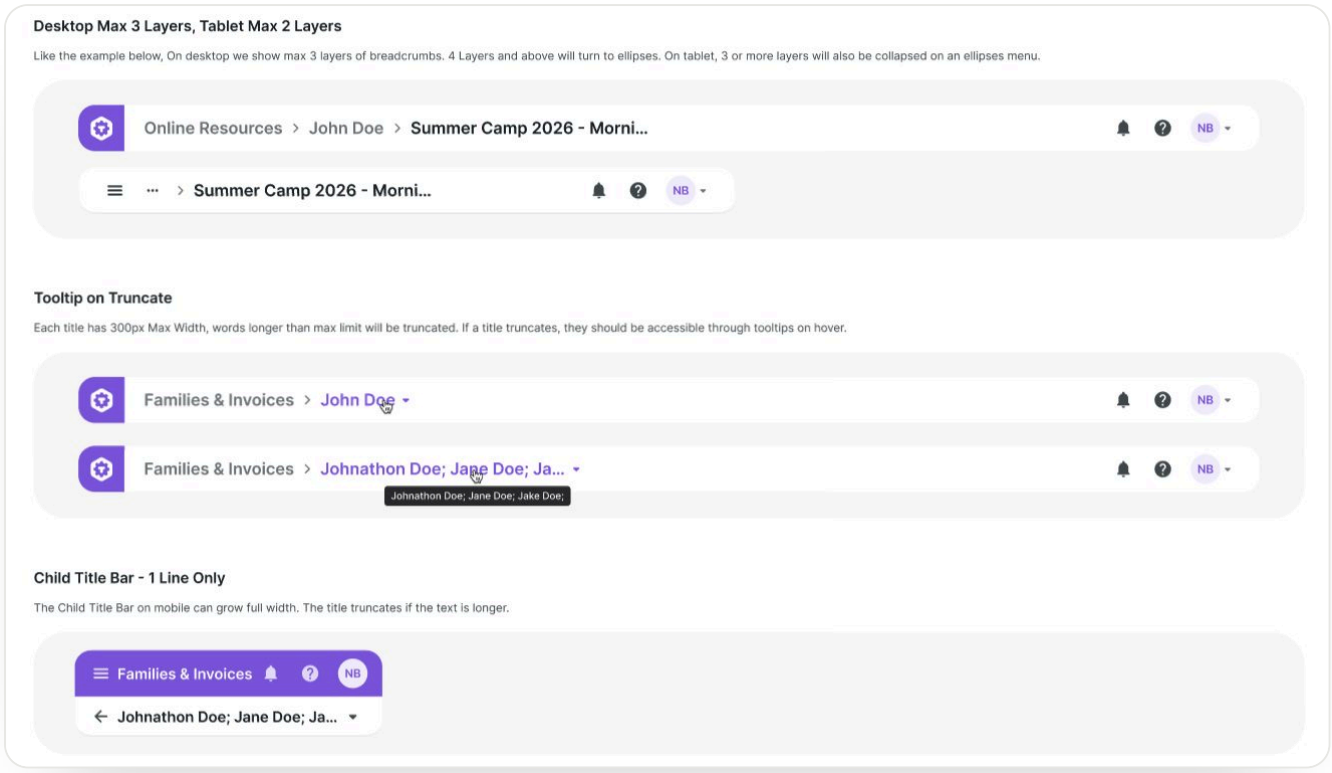


The combined control panel: sidebar, search, and inline record creation in one surface.

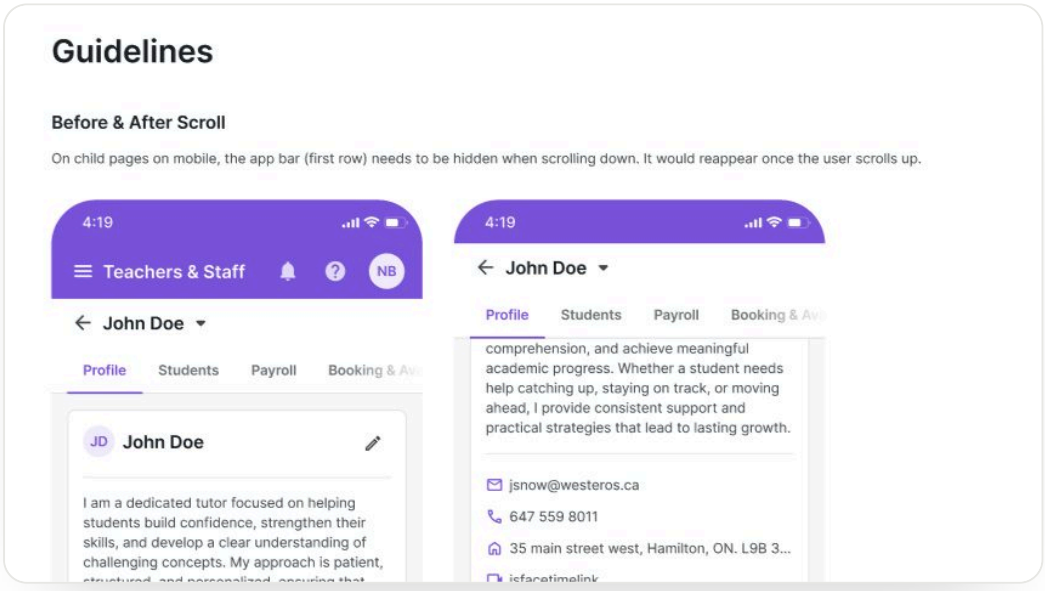
App bar and breadcrumbs

I redesigned the app bar and added breadcrumbs, so people navigating deep into the products always know where they are and how they got there. This mattered

most for new users during onboarding, who previously got lost in the deeper layers.



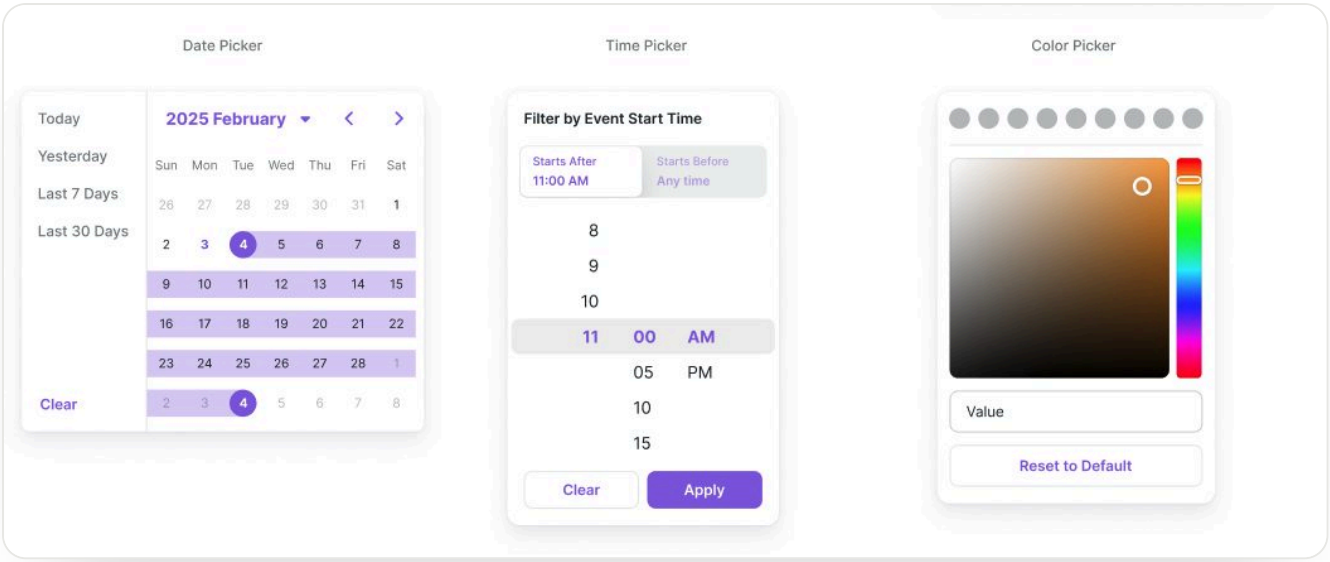
Breadcrumb behaviour across desktop, tablet, and mobile: truncation, tooltips, and collapse rules.



App bar scroll behaviour on mobile child pages.

Pickers

Date, date range, and colour pickers, all built on the same tokens and state model as everything else.



Date, date range, and colour pickers sharing one token and state model.

Impact

- The token system grew from roughly 300 to about 1,400 variables across six collections, all wired to components.
- Three products can switch complete themes in one click, with dedicated staging and development modes for safe testing before release.
- Forms went from 15 components to one Form Field Container with 14 variants, cutting related tokens by 24 percent.
- More than 30 components were designed or rebuilt to WCAG 2.2.
- The library went from one gatekeeper to three contributors publishing under versioned, documented releases.
- Engineering consumes tokens through an automated export rather than rebuilding values by hand, so design and code stay in sync by default.

The change you feel day to day is quieter than any number. Two designers picking up the same feature now produce the same form, and nobody waits weeks for a checkbox.

Reflection

1

A design system is a product

Its users are designers and developers. The visual craft matters, but the parts that made this system stick were the governance pieces: a naming grammar people can guess, versioned releases, and specs that live with the system.

2

Made with engineering, not for it

The best architectural decisions were made with engineering, not handed to it. The Form Field Container and the export tooling both came out of working sessions with the engineering lead, and both landed because they matched how the code is actually structured.

3

Constraints are design inputs

I still believe live spec tooling is the right long term call, and I said so at the time, but working without it pushed us to build an export pipeline that fits our stack more precisely than anything off the shelf would have.

4

Start governance early

If I began again, I would introduce versioning and release notes on day one rather than waiting for contributors to arrive. Traceability is cheap as a habit and expensive as a retrofit.

Let's talk through the details

I'm glad to walk through the token architecture, the export pipeline, and the governance decisions behind this system in a live conversation.

Get in touch

Behrad Bayati · Product Designer · thisisbayati@gmail.com