Building ZUI: Why We Needed a UI Library and What It Took to Get It Right

The Problem Wasn’t Styling — It Was Entropy

On the surface, the issue looked like inconsistency:

- buttons that looked almost the same but weren’t
- spacing that varied by a few pixels
- typography implemented slightly differently in each app

But inconsistency wasn’t the root problem. Entropy was.

Each team solved the same UI problems independently:

- form fields
- loading states
- empty states
- modals
- tables

Every solution made sense locally. Collectively, they made the system harder to reason about.

ZUI wasn’t about aesthetics. It was about *reducing entropy*.

The First Rule: Don’t Build a Design System in Abstraction

One of the easiest mistakes to make is starting with theory:

- tokens before components
- exhaustive variants before real usage
- future-proofing everything

ZUI avoided that by being built directly from production needs.

Every component added to ZUI had to satisfy three constraints:

1. It was already being duplicated across apps
2. It had real usage, not hypothetical usage
3. It reduced code, not just moved it

If a component didn’t simplify something immediately, it didn’t belong.

Headless First, Opinionated Where It Matters

ZUI is intentionally not a “batteries included” UI framework.

Layout, composition, and accessibility are handled at the library level. Visual expression is kept flexible.

That balance mattered.

Too headless, and every consumer reinvents the same patterns. Too opinionated, and teams fight the library instead of using it.

ZUI draws a clear line:

- **behavior is shared**
- **structure is shared**
- **styling is constrained but extensible**

Tailwind made this possible. Instead of fighting CSS abstractions, ZUI embraces utility composition while enforcing consistency where it actually matters.

Tokens Are Useless If They Don’t Match Reality

Design tokens are often treated as a source of truth.

In practice, they’re only useful if they reflect how the product is actually built.

ZUI treats Figma as the visual source of truth and code as the enforcement mechanism. Tokens exist to bridge that gap—not to introduce a second system teams have to keep in sync manually.

When tokens drift from reality, engineers stop trusting them. When engineers stop trusting tokens, consistency collapses.

Avoiding the “Wrapper Hell” Trap

Another lesson came from restraint.

It’s tempting to wrap everything:

- `ZuiButton`
- `ZuiInput`
- `ZuiModal`

But excessive abstraction creates new problems:

- leaky props
- unclear ownership
- difficult escape hatches

ZUI avoids unnecessary wrappers. When a primitive already does the job well, ZUI composes it instead of replacing it.

The goal isn’t to own everything. It’s to make the *right things boring*.

Versioning Is a Social Contract

Breaking UI libraries don’t fail because of code. They fail because of trust.

ZUI treats versioning as a contract:

- breaking changes are explicit
- migrations are documented
- consumers are never surprised

If upgrading the UI library feels risky, teams will stop upgrading. Once that happens, the library is dead.

Stability is a feature.

What ZUI Actually Changed

After ZUI:

- new features shipped faster
- UI regressions decreased
- PRs became easier to review
- designers and engineers spoke a more shared language

None of this showed up as a single metric. It showed up as *less friction everywhere*.

That’s the hallmark of good infrastructure.

The Real Value of a UI Library

ZUI didn’t make the product prettier. It made the product *calmer*.

Calmer to work on. Calmer to change. Calmer to reason about.

UI libraries aren’t about control. They’re about leverage.

And like most leverage in engineering, their value compounds quietly over time—long after the excitement of building them has passed.

That’s when you know you built the right thing.