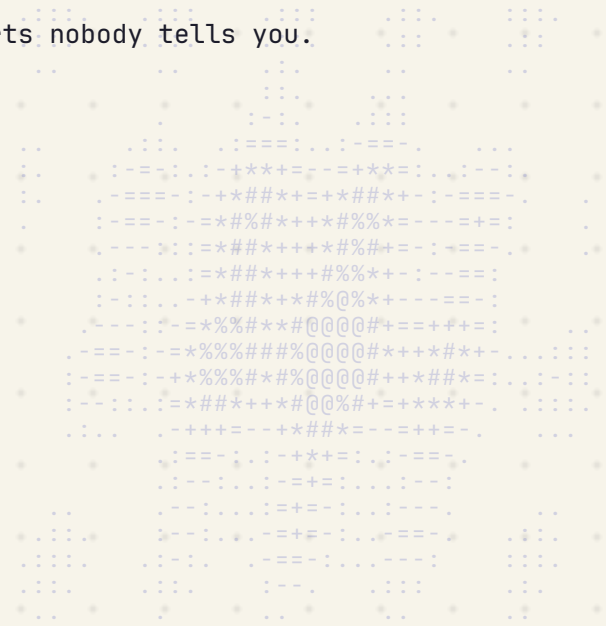
// SECTION 01 · CLAUDE SETUP

THE DESIGN STACK

Three installs that stop Claude building websites that look like Claude built them. The design vocabulary, the reference library, and the analytics connection that lets it watch the page and fix what is not working.

```
> boot opusjake_os
> resource: design-stack
> version: v1.0 · 2026
> status: ready_
```

Claude can already write the code. What it cannot do out of the box is decide. So it reaches for the same safe defaults every time and you get the page you have seen a thousand times. Three installs fix three different halves of that. Here is what each one actually does, the exact commands, and the parts nobody tells you.



THE THREE

One problem, three causes. Install all three or you only fix a third of it.

#	INSTALL	WHAT IT FIXES
1	`impeccable`	No design vocabulary. It cannot name what is wrong.
2	`taste`	No references. It averages the internet instead.
3	`posthog` MCP	No feedback. It never finds out if the page worked.

// WHY IT IS THREE AND NOT ONE

Generic output is not one failure. It is three.

The first is language. Ask for "make it look better" and Claude has nothing to reason with, so it adds a gradient. Impeccable hands it the words a designer would use, as commands you can run.

The second is reference. With nothing to point at, the model produces the average of everything it has seen, which is the definition of generic. Taste gives it specific things to point at, chosen by you.

The third is feedback. Design is a guess until someone uses it. PostHog closes that loop so the next pass is informed instead of decorative.

ONE: IMPECCABLE

Design fluency for AI harnesses. Built by Paul Bakaus. Apache 2.0, free, version 3.9.1 at time of writing.

```
claude plugin marketplace add pbakaus/impeccable
claude plugin install impeccable@impeccable
```

Inside a session use the slash version instead:

```
/plugin marketplace add pbakaus/impeccable
/plugin install impeccable
```

// THE TWENTY THREE COMMANDS

COMMAND	WHAT IT DOES
craft	Shape then build a feature end to end
shape	Plan the UX before any code
init	Capture PRODUCT.md and DESIGN.md
document	Write DESIGN.md from code that exists
extract	Pull tokens and components into a system
critique	UX review with heuristic scoring
audit	Accessibility, performance, responsive
polish	Final pass before shipping
bolder	Amplify a safe, bland design
quieter	Calm an overstimulating one
distill	Strip to the essence
harden	Errors, i18n, edge cases
onboard	First run, empty states, activation
animate	Motion with a purpose
colorize	Color into a monochrome UI
typeset	Typography hierarchy and fonts
layout	Spacing, rhythm, hierarchy
delight	Personality and memorable touches
overdrive	Past the conventional limit
clarify	UX copy, labels, error messages
adapt	Other devices and screen sizes
optimize	Diagnose and fix UI performance
live	Pick elements in the browser, get variants

// THE FOUR YOU WILL ACTUALLY USE

`/impeccable init` once per project. It writes PRODUCT.md and DESIGN.md, and every other command reads them. Skip it and the rest run blind.

`/impeccable craft` to build. `/impeccable audit` before you ship. `/impeccable polish` when it works but feels cheap.

The rest are precision tools. Reach for `typeset` when the type is flat, `layout` when the spacing is arbitrary, `quieter` when you overdid it.

THE SLOP DETECTOR

The part that earns the install. Impeccable ships a local detector with a registry of named anti-patterns, forty six of them in 3.9.1, and it reads your actual HTML and CSS. No network, no npx, no upload.

These are real rule names out of that registry:

```
ai-color-palette      hero-eyebrow-chip
gradient-text         icon-tile-stack
cream-palette         nested-cards
dark-glow             oversized-h1
flat-type-hierarchy   overused-font
em-dash-overuse       single-font
low-contrast          tiny-text
marketing-buzzword    tight-leading
monotonous-spacing    wide-tracking
```

// READ THAT LIST AGAIN

Every one of those is something an AI reaches for when it has no opinion. The purple to blue gradient headline. The little pill above the h1. The cards inside cards. One typeface doing every job. Body text at 13px because it looked tidier.

You have shipped half of these. I have shipped most of them. That is the point: the detector names the defaults, and a default you can name is a default you can delete.

`gpt-thin-border-wide-shadow` is in there as its own rule. So is `em-dash-overuse`. Somebody was paying attention.

TWO: TASTE

A Claude Code plugin for building with taste, in interfaces and in prose. Built by obakeng-develops. MIT, free.

```
claude plugin marketplace add obakeng-develops/taste
claude plugin install taste@taste
```

COMMAND	WHAT IT DOES
`/taste`	The skill. Auto loads on design and frontend work.
`/taste-learn`	Save a reference, with the reason.
`/taste-audit`	One screen or passage, ranked findings, no edits.
`/taste-sweep`	The whole codebase, ranked by scope.
`/taste-help`	The reference card.

// THE PART NOBODY TELLS YOU

The library is not full when you install it. It seeds a template at `~/.claude/taste/references.md` and then it is yours to grow. Install it, ask for a premium page, and on day one it is still mostly guessing.

So do the twenty minutes. Open ten sites you actually admire and run `/taste-learn` on each one. Studio sites, product pages, whatever makes you jealous.

That file is the whole product. Everything else is plumbing.

// WHAT TO WRITE DOWN

A bare link teaches nothing. Three fields per reference:

```
WHAT   what you are looking at
WHY    the specific reason it works
STEAL  the part you would take
```

"Nice site" is useless. "The hero is one sentence at 72px with nothing else above the fold, so the page has one job" is a rule Claude can apply to a page that looks nothing like it.

Steal structure, not pixels. Copy the pixels and you have made a different generic page.

THREE: POSTHOG MCP

The first two make the page look designed. This one tells you whether it worked. PostHog is product analytics, and the MCP is the official connection between it and Claude.

It is in the official marketplace, so this one is a single line:

```
claude plugin install posthog
```

Two other routes, same result:

```
npx @posthog/wizard mcp add

claude mcp add --transport http posthog \
  https://mcp.posthog.com/mcp -s user
```

You will be asked to log in to PostHog the first time you use it. The region routes off the account.

// WHAT YOU CAN ASK IT

Once it is connected you stop opening dashboards and start asking questions.

```
Which section do people scroll past on the
landing page?

What are the top 5 errors this week, with the
stack trace for the worst one?

Create a feature flag for the new hero and
roll it out to 20%.
```

Error tracking, feature flags, experiments, HogQL queries, insights, session data. It reads and it writes.

// THE LOOP

This is the reason it is in the stack.

Build with impeccable and taste. Ship. Ask PostHog where people leave. Feed that answer back into `/impeccable craft` as the brief for the next pass. The second version is informed. Every version after that is evidence.

Without this step you are redesigning on taste alone, which is fine right up until your taste and your buyers disagree.

THE ORDER

Paste this block, restart Claude Code, and you are set up.

```
claude plugin marketplace add pbakaus/impeccable
claude plugin install impeccable@impeccable

claude plugin marketplace add obakeng-develops/taste
claude plugin install taste@taste

claude plugin install posthog
```

// THEN, IN THIS ORDER

```
/impeccable init
```

Answers a handful of questions, writes PRODUCT.md and DESIGN.md. Ten minutes, once, and every later command is better for it.

```
/taste-learn
```

Ten sites, three fields each. Do this before you ask for anything to be built, not after you dislike what you got.

```
/impeccable craft the landing page
/taste-audit
/impeccable audit
/impeccable polish
```

Craft builds it. Taste judges it against your references. Audit catches the accessibility and performance problems. Polish is the last pass.

Then ship it, let it run a week, and ask PostHog what happened.

WHAT IT WILL NOT DO

The honest page. Every one of these is a real limit, not a disclaimer.

// IT DOES NOT GIVE YOU A BRAND

These three raise the floor. They do not raise the ceiling. They will reliably stop you shipping something embarrassing and they will not invent a point of view for you. Colors, name, voice, the thing you are actually saying: still yours.

// TASTE IS ONLY AS GOOD AS WHAT YOU FEED IT

Said once already, worth saying twice, because it is the step everyone skips. An empty references file makes the plugin an expensive no-op.

// THE MCP CAN WRITE

PostHog's MCP supports write operations across its products. It can create flags, change them, resolve issues. That is genuinely useful and it is also production. Read what it proposes before you approve it.

// IT COSTS SOMETHING

Connecting the MCP is free. The AI powered tools inside it bill against PostHog AI spend, and your organization needs AI data processing turned on before they run at all. Check both before you promise a client a live analytics loop.

// NONE OF IT CAN SEE YOUR SITE BY DEFAULT

Impeccable's `audit` and the detector read your files. `live` needs a dev server running to iterate in the browser. Taste judges what you point it at. If you never point them at the real rendered page, all three are reviewing your intentions rather than your work.

GET IT

Three links. All free.

impeccable by Paul Bakaus, Apache 2.0 github.com/pbakaus/impeccable · impeccable.style

taste by obakeng-develops, MIT github.com/obakeng-develops/taste

PostHog MCP, official posthog.com/docs/model-context-protocol

// THE SHORT VERSION

Impeccable gives Claude the words. Taste gives it something to point at. PostHog tells it whether any of it worked.

Install all three, spend twenty minutes on your references, and stop accepting the first thing it builds.