The Frontend Design Skills Pack

The five Claude Code skills I use to take AI slop out of a front end, with every install command and the prompts I run them with.

Why I put this pack together

Every interface a coding agent builds on its own looks like every other interface a coding agent builds on its own. Purple gradients. Glows on everything. Emoji standing in for icons. Spacing that drifts by four pixels per section. The code works, the page loads, and it still reads as machine output the second someone looks at it.

The reason is simple once you see it. Claude Code writes markup and CSS well, and it has no opinion about which of a thousand valid layouts you actually wanted. With no direction it picks the average of everything it has read. The average is the slop.

A design skill fixes that by handing the agent a vocabulary before it writes a line. These are the five I keep installed: Impeccable, Taste, Unslop, Design Extract and Playwright CLI. They are all free and open source as of November 2026, and between them they cover detecting the problem, rewriting the screen, stealing a look you like, and checking the result in a real browser.

01

How it actually works

1. A skill is a folder of instructions Claude reads before it starts work. It sits in a skills directory on your machine, and the agent loads it when the task matches what the skill says it is for. Nothing runs in the cloud and nothing phones home.
2. The design skills in this pack each add a named vocabulary. Instead of asking for 'make it look better', you say bolder, or quieter, or distill, and the skill already knows what each of those means in spacing, weight and colour terms. That is why the output stops being average.
3. Two of them go further and open a real browser. Impeccable's live mode runs your dev server with an overlay on top so you click an element and watch variants swap in. Playwright CLI lets Claude take a snapshot of the page, read every element as a short reference like e15, and then click and fill against those references instead of guessing at selectors.

02

The one thing almost everyone gets wrong

You install a skill, type the command, and Claude acts like it has never heard of it. Nine times out of ten this is one of two things, and neither of them is an error message.

First: Impeccable is a two-step install. Running npx impeccable install only puts the files in place. You then have to reload your editor and run the init command inside the agent. Skip the reload and the skill is on disk but not in the session, so Claude genuinely cannot see it.

Second, for any skill you install by copying a folder, like Unslop: the folder name has to match the name field inside that skill's SKILL.md exactly. If the repo gives you a folder called unslop, it has to stay called unslop. Rename it to unslop-skill or my-unslop and Claude will not load it, and it will not tell you why.

> **WATCH OUT**
>
> After any install, open a brand new terminal and start a fresh Claude Code session before you test. Skills are read when the session starts. Installing one into a session that is already running does nothing, and that single detail accounts for most of the 'it does not work' messages I get.

---

03 The five skills and how each one installs

> **SOURCE CODE**
>
> Skill one, Impeccable, by Paul Bakaus. The design vocabulary system with 24 commands and the live in-app browser. Every repo in this pack is listed at the bottom of this section.
>
> **https://github.com/pbakaus/impeccable**

1. IMPECCABLE: DETECT AND STEER

```
# run from your project root
npx impeccable install

# then reload your editor and, inside the agent:
/impeccable init          # Claude Code
$impeccable init          # Codex

# pin a team version instead, via git submodule:
git submodule add https://github.com/pbakaus/impeccable vendor/impeccable
npx impeccable link --source=vendor/impeccable --providers=claude,cursor

# scan without the agent:
npx impeccable detect src/
npx impeccable detect https://example.com
npx impeccable detect --json .
```

2. TASTE: BUILD THE PAGE FROM SCRATCH

```
# as a skill
npx skills add https://github.com/Leonxlnx/taste-skill --skill "design-taste-frontend"

# or as a Claude Code plugin
/plugin marketplace add Leonxlnx/taste-skill
/plugin install taste-skill@taste-skill

# or as a Codex plugin
codex plugin marketplace add Leonxlnx/taste-skill
codex plugin add taste-skill@taste-skill

# or symlink a local clone
./install.sh

# start a build (page kinds: software, gaming, tech, product, service, place)
node scripts/start.mjs deal --page software

# verify it in a real browser when the build is done
node scripts/consistency.mjs <url or file> --page <kind>
```

3. UNSLOP: REWRITE THE GENERIC PARTS

```
# Claude Code
git clone https://github.com/heydamir/unslop
mkdir -p ~/.claude/skills && cp -r unslop/skills/unslop ~/.claude/skills/

# Codex
$skill-installer install https://github.com/heydamir/unslop/tree/main/skills/unslop
# or manually:
mkdir -p ~/.agents/skills && cp -r unslop/skills/unslop ~/.agents/skills/

# Google Antigravity
mkdir -p ~/.gemini/antigravity/skills && cp -r unslop/skills/unslop
~/.gemini/antigravity/skills/

# optional, needed for live site snapshots and the story videos
npm install -D playwright
npx playwright install chromium
```

4. DESIGN EXTRACT: PULL A DESIGN SYSTEM OFF ANY URL

```
# global install
npm i -g designlang
designlang install-browser

# or per project, no install
npx designlang https://stripe.com

# as an agent skill
npx skills add Manavarya09/design-extract

# as a Claude Code plugin
/plugin install Manavarya09/design-extract
/extract <url>
/grade <url>
/battle <urlA> <urlB>

# useful flags
npx designlang <url> --dark          # include dark mode
npx designlang <url> --responsive    # crawl several breakpoints
npx designlang <url> --full          # turn every capture on
npx designlang grade <url>           # letter-graded report card
npx designlang clone <url>           # Next.js starter
```

5. PLAYWRIGHT CLI: GIVE CLAUDE A REAL BROWSER

```
# this project only
npx skills add microsoft/playwright-cli --skill playwright-cli

# every project
npx skills add microsoft/playwright-cli --skill playwright-cli -g

# the commands Claude will reach for
playwright-cli snapshot          # read page structure, each element gets a ref like
e15
playwright-cli find              # search the snapshot by text or regex
playwright-cli route             # mock a network response
playwright-cli recording-start   # capture actions as Playwright TypeScript
playwright-cli recording-stop
playwright-cli attach            # connect to a paused debug session
playwright-cli state-save        # keep cookies and storage
playwright-cli state-load
```

EVERY LINK FROM THE VIDEO

- Impeccable, by Paul Bakaus. Design vocabulary, 24 commands, live in-app browser.
  https://github.com/pbakaus/impeccable
- Taste, by Leonxlnx. Builds full pages with a design system and real photographs.
  https://github.com/Leonxlnx/taste-skill
- Unslop, by Damir Sokolovsky. Nine-pass audit that rewrites vibecoded UI. CC BY 4.0, credit required.
  https://github.com/heydamir/unslop
- Design Extract, by Manavarya09. The designlang CLI that pulls tokens out of a live site.
  https://github.com/manavarya09/design-extract
- Playwright CLI skill write-up on Collective Brain.
  https://collectivebrain.de/en/skills/playwright-cli/

Set it up in Claude Code or Codex

1. Work out where your skills live. In Claude Code, a skill you want everywhere goes in ~/.claude/skills/ with one folder per skill, so Unslop ends up at ~/.claude/skills/unslop/. A skill you only want in one project goes in that project's own skills folder. In Codex the equivalent global path is ~/.agents/skills/, and if you keep a skill inside a project you add a pointer to it in AGENTS.md so the agent knows to read it.
2. Clone the repos you want. Impeccable is at https://github.com/pbakaus/impeccable, Taste at https://github.com/Leonxlnx/taste-skill, Unslop at https://github.com/heydamir/unslop and Design Extract at https://github.com/manavarya09/design-extract. For Unslop you clone and copy: git clone https://github.com/heydamir/unslop then cp -r unslop/skills/unslop ~/.claude/skills/. The other three have their own installers, listed in the section above.
3. Check the folder name against the skill file. Open the SKILL.md inside the folder you just copied, look at the name field at the top, and make the folder name match it character for character. This is the step that silently breaks installs.
4. Get Node sorted before anything else. Taste needs Node 22 or newer. Design Extract needs Node 20 or later. Impeccable needs Node only for its npx commands, and it downloads a self-contained engine binary once into ~/.impeccable/bin/ the first time you run it. If you are unsure which version you have, ask Claude to check node --version and tell you whether you need to upgrade.
5. Point the tools at a browser. Taste wants Chrome, Edge or Chromium, and if yours is in a non-standard location you set CHROME_PATH to it. Ask Claude to add that line to your shell profile, then open a new terminal so it takes effect. Design Extract ships bundled Chromium through Playwright, so it handles its own. Unslop only needs a browser if you want it to look at a live URL, which is what the optional playwright install covers.
6. Run Impeccable's init. From your project root: npx impeccable install, then quit and reopen your editor, then run /impeccable init inside the agent. It writes a .impeccable/ folder into the project holding PRODUCT.md, DESIGN.md and config.json. Commit those three, because they are the shared design brief everyone on the project is working from.
7. Do a first test that cannot fail quietly. Open a fresh terminal, start Claude Code in a project that has some UI in it, and run /impeccable critique the hero section. A working install answers with a scored review of that specific element. If you get a generic design opinion with no reference to your actual markup, the skill did not load and you are back at step three.
8. Add the other four one at a time. Install, open a new session, run one command, confirm it answers from the skill rather than from general knowledge, then move on. Doing all five in one go means that when something does not respond you have no idea which install went wrong.

No API keys are needed for any of the five. Taste runs offline with its photographs bundled in, and Design Extract only needs a network connection because it has to load the site you point it at.

> 💡 **QUICK TIPS**
>
> - Run these on a branch. Impeccable writes accepted changes straight back to your source files, so you want a clean place to compare against and an easy way out.
> - Unslop is CC BY 4.0, which means if you ship something built with it you credit Damir Sokolovsky and link the repo. Worth knowing before it ends up in client work.

- Design Extract writes everything into its own ./design-extract-output/ folder and never touches your project, so it is the safest of the five to try first on a repo you care about.

04

The prompts I actually run

FIND OUT HOW BAD IT IS

```
/impeccable critique the [SECTION NAME, e.g. pricing cards] in [FILE OR ROUTE]. Score it, then list the three changes that would move the score most, in order.
```

FIX IT LIVE, WATCHING EVERY CHANGE

```
/impeccable live

Start the dev server with the overlay. I want to work on [SECTION NAME]. Show me a quieter variant and a bolder variant side by side before you write anything to the files.
```

REWRITE A SCREEN THAT LOOKS AI-MADE

```
/unslop the [SCREEN NAME, e.g. dashboard] at [URL OR FILE PATH]

Propose the design direction first and wait for me to pick one. Then run the spacing, colour, typography and icon passes. Leave the copy alone.
```

STEAL A LOOK YOU LIKE, LEGALLY

```
npx designlang [URL OF SITE YOU LIKE] --dark --responsive

Then: read [SITE]-design-language.md and [SITE]-design-tokens.json from ./design-extract-output/ and rebuild my [COMPONENT NAME] using those tokens. Use their spacing scale and type scale. Do not copy their copy, their logo or their photographs.
```

MAKE THE AGENT CHECK ITS OWN WORK

```
Use playwright-cli. Open [URL], take a snapshot, and tell me what the page structure actually is by element reference. Then check [SPECIFIC THING, e.g. the signup form] on desktop and again with --mobile, and screenshot both. Tell me what breaks before you tell me what works.
```

BUILD A NEW PAGE INSTEAD OF FIXING AN OLD ONE

```
node scripts/start.mjs deal --page [software | gaming | tech | product | service | place]

Build the page for [WHAT THE PRODUCT IS, one sentence]. When it is done run scripts/consistency.mjs against the output and show me the result before I open it.
```

When it breaks

ERROR	FIX
Claude ignores the slash command and answers with generic design advice	The skill is not loaded. Quit the session, open a new terminal, start Claude Code again. For Impeccable, confirm you ran /impeccable init after reloading the editor, not just npx impeccable install.
Unslop installed but never triggers	The folder name does not match the name field in its SKILL.md. It must stay ~/.claude/skills/unslop/. Also check you copied unslop/skills/unslop and not the whole repo.
npx impeccable install fails, or the engine never appears	Node is missing or too old. Impeccable needs Node for npx, and downloads its engine binary once to ~/.impeccable/bin/. Delete that folder and run the install again to force a clean download.
Taste exits immediately, or cannot find a browser	Node must be 22 or newer, and Taste needs Chrome, Edge or Chromium. If your browser is in a non-standard location, set CHROME_PATH to the binary in your shell profile and open a new terminal.
designlang install-browser fails or the extraction returns nothing	Node must be 20 or later. If the page is behind a login, pass credentials with --cookie or --cookie-file, because an unauthenticated load gives it an empty DOM to read.
/impeccable live opens but no variants swap in	Live mode supports Vite, Next, SvelteKit, Astro, Nuxt and plain HTML. On any other setup the overlay loads and the hot swap does not. Use /impeccable critique and the steering commands instead.
Unslop cannot see the live site you pointed it at	The optional Playwright dependency is missing. Run npm install -D playwright then npx playwright install chromium in that project.
playwright-cli commands are not found	The skill is installed but the playwright-cli binary itself is not on your PATH. It has to be callable from your shell before the skill can use it. Install it globally, then open a new terminal.

Ready to go deeper?

Want the videos made for you? Paste any link or idea into Content Engine and get a published video in under 3 minutes, with DM automation armed to capture leads.

Meet Content Engine →

https://builds.digicuratoragency.com/content-engine/