Elton Carvalho

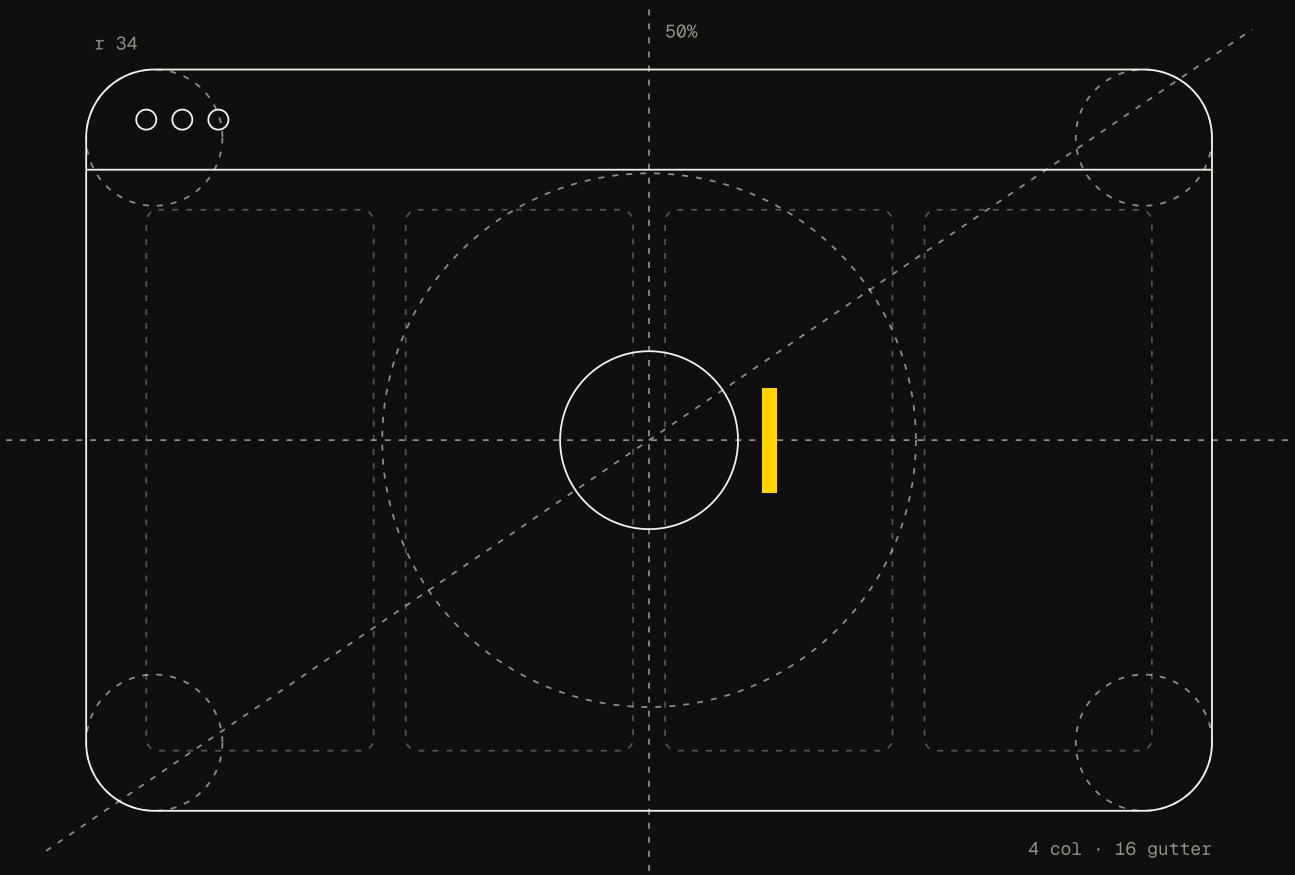
GUIDE · V1.0

Frontend with Claude Code

A PROCESS
FOR DESIGNERS

v1

2026



10 STEPS · 4 PHASES · 2 CHECKPOINTS

ELTONCARVALHO.COM.BR/EN/PROCESSO

FREE · OPEN SOURCE

> Interactive version, with the step-through flow, the simulator, the prompt generator and the downloadable skill: https://eltoncarvalho.com.br/en/processo/

This is the process and method behind 30+ design system and dark mode PRs now in production, and what I learned along the way. The designer decides product, visuals and patterns. Claude acts as the senior developer and handles code, tests and evidence. Engineering reviews and merges.

The idea in one sentence

Each card becomes **one Claude session, in its own worktree, with a clear kickoff prompt and two checkpoints**: the plan and the evidence. Claude handles everything else (code, checks, screenshots, tests, PR) and shows you the result. Commits and PRs happen only when you ask.

Who decides what

You decide	Claude decides and executes
Card scope and what's left out	Technical organization (files, commits, order)
Visuals, patterns, behavior the user sees	Which design system component to use, tokens, safe refactors
Approve the plan before any edit	Checks (lint, types, tests, E2E, performance)
Approve the evidence (before and after screenshots)	Screenshots, comparisons, evidence log
Ask for the commit and PR	Open the PR in the team's format and move the card
Lead the conversation with reviewers and decide what changes	Fix the review bot's findings and apply what you agree on

What changes the outcome most is making the **decision boundary** explicit: Claude settles anything technical and reversible on its own, and checks with you when the decision is about product or visuals, involves someone else, or can't be undone.

What you need

- **Claude Code**, ideally in the desktop app (sessions, worktrees, browser and terminal in one place).
- **Access to the repository**, and someone from engineering to review and merge.

- **A design system** with components and semantic tokens, or at least the tokens.
- **The local environment running** on your machine. Setting it up the first time is the hardest part: it's worth doing together with someone from engineering.
- **Connectors**, if your team uses them: task tracker, Figma, GitHub, and the app's browser (or Playwright) for screenshots and E2E.
- **Repo rules** (`AGENTS.md`, `CLAUDE.md`). Claude already reads them; your job is to call it out when it drifts.

Adapt it to your team

The method is the same in any project. What changes is what each team already has. Define it once and put it in the skill and in the kickoff prompt.

Item	Example
Main branch	`main`
Design system	the package, and where its rules live
Alternative library	what to use when the design system lacks a component
Task tracker	Linear, Jira, GitHub Issues; which status means "in review"
Branch format	`username/card-id-title-in-kebab-case`
PR format	title, labels, ready or draft
Setup for a new worktree	install and build the internal libraries
Start the local environment	the team's command or script
Checks	lint, package types, tests, dead code
Documentation	whether user-visible changes need a docs page
Review bot	its name, if any
Review request	channel and message format

The flow of a PR

Ten steps in four phases.

Prepare

1. **Small card.** A short title (about 40 characters), a clear scope and a list of what **not** to touch. One PR per card. If the scope grows midway, it becomes another card.
2. **Dedicated session and worktree.** One session per card, named after the card so you can find each one in the sidebar (e.g. `DS-14 PR 3: sign-up fields`), in a fresh worktree off an up-to-date main branch. Never an old branch: it shows screens that no longer exist. Claude runs the worktree setup.
3. **Kickoff prompt.** Instead of explaining the task from memory every time, you paste a prompt with the goal, the visual reference, what's out of scope, the rules and the checkpoints. Template at the end.
4. **Checkpoint 1: the plan.** Before editing, Claude sends the plan: files, the components it will use, what stays the same, risks. You approve or correct it. It's the step that saves the most rework.

Build

5. **Building, with fixed rules.**
    - **Design system first.** Use an alternative library only when the design system lacks the component; never use a component marked as draft.
    - **Don't rebuild what already exists.** If you point to a screen as a reference, Claude uses the same component and adjusts it only through props and classes. Rewriting it "cleaner" breaks the platform's identity.
    - **Semantic tokens, light and dark.** No hardcoded colors; every screen checked in light and dark mode.
    - **A visual swap doesn't change behavior.** Validation, copy, field order, the payload sent to the API and `data-testid` stay the same. A change in behavior is a bug.
    - **The laziness ladder.** Does it need to exist? Does it already exist in the repo? Does the platform or an installed dependency handle it? Only then write the minimum.
    - **Bugs: fix the root cause.** Find every place that uses the function and fix it once, at the source.

Prove

6. **Checkpoint 2: the evidence.** No PR is opened without it.
   - Before (main branch) and after (your branch) screenshots, light and dark, including error and disabled states.
   - E2E for the area on both branches. A failure that already happens on main is pre-existing, not the PR's fault.
   - Performance of main versus your branch: LCP, CLS and long tasks, measured **back to back, in the same time window**. Measurements hours apart once showed a false 21% regression.
   - Everything recorded in an `evidence-<card>.md` file, on the card and in the PR description.
7. **Checks and docs.** The team's checks, only on the packages touched. If the team requires documentation for visible changes, it goes in the same PR.
8. **Independent review.** A separate reviewer (a subagent or another session acting as reviewer) looks only at the diff, hunting for bugs. If anything changes afterward, it runs again: the second round has caught side effects of the first round's fix.

Deliver

9. **Commit and PR only when you ask.** PR in the team's format, card moved to "in review" with the link, review request in the team's format.
10. **After the PR is open.** Review bot findings: Claude fixes what's in scope and replies in the thread. Comments from people: the conversation is yours; Claude brings the technical context for each point when you ask and applies what's agreed.

Lessons

The method didn't come out fully formed. Each rule came from something that went wrong, or that worked better than expected.

1. **Before and after is the deliverable.** Without the comparison, fix-up work goes unnoticed: reviewers can't see what changed, and users never knew there was a problem. That's why evidence is a checkpoint, not an attachment.
2. **The scoreboard lives in code, not in a spreadsheet.** A script counts, on every merge, how many uses of the old component remain, and a lint rule blocks new ones. Without that guardrail, five new files using the old component showed up in four days.
3. **Measure back to back, in the same window.** A performance run done hours apart showed a 21% regression that didn't exist. Once both branches were measured back to back, the difference disappeared.
4. **Swapping components is an audit.** The most serious bugs were on screens nobody had asked me to look at. Opening each screen to swap the component exposes what was already broken around it.

5. **AI executes; design judgment decides.** The AI rewrote a set of tabs to make them "cleaner" and broke the platform's identity. That became a rule: don't rebuild what already exists, and reuse the component from the reference screen.
6. **Independent review catches what the author misses.** A second session, looking only at the diff, found a bug that was already in production. Since then, the review runs again whenever something changes after it.
7. **Measure before you promise, one area per PR.** The number comes before the commitment. And each PR touches a single area, so it can be rolled back without undoing the rest.
8. **A fixed mistake becomes memory.** When Claude makes a mistake that must not happen again, the fix goes into the project memory, along with the reason.

Evidence

The comparison is always the same grid: before on the left, after on the right; light on top, dark below. When the swap is invisible to users (a forms library, for example), the evidence shows the error state: the same messages, in the same place.

Performance criteria: no LCP regression above 10%, CLS doesn't go up, no new long task above 50 ms.

Log template:

```
# Evidence · <ID> <title>

## Screenshots
Before (main) on the left, after (branch) on the right; light on top, dark below.
- <screen>: default, error, disabled

## E2E for the area
- Main: <x of y> · Branch: <x of y>
- Pre-existing (already failing on main): <test and reason>

## Performance (both back to back, in the same window)
| Screen | LCP before | LCP after | Difference | CLS before | CLS after | New long tasks |
|-|-|-|-|-|-|-|

## Behavior
- Validation, copy, payload and data-testid unchanged: <how it was checked>

## Checks
- <lint, types, tests>: <result>
- Didn't run, and why: <...>

## Independent review
- Round 1: <findings and what changed>
- Round 2: <if anything changed afterward>
```

Several PRs at once: the orchestrator session

The problem

Each PR runs in its own session and worktree, but there's **usually only one local environment**: fixed ports, one database. Only one worktree can be running at a time, and every PR needs it for screenshots, E2E and performance. Switching the environment by hand, PR by PR, doesn't scale: every switch depends on someone being available right then, and PRs sit waiting. The fix is to hand the environment to a single session, the **orchestrator**, which decides who gets it and does the switching on its own.

What it does and doesn't do

Does	Doesn't
Is the sole owner of the environment: starts, switches and stops it	Write the PRs' code
Keeps the queue of who's next for the environment	Decide visuals or product
Checks the machine's load before starting anything heavy	Commit or open PRs for other sessions
Maintains `STATUS.md`, the status board for every PR	Talk to reviewers on the PRs' behalf
Flags PRs that depend on each other	
Prepares the card and prompt for the next PR	

Design decisions, commits and PRs still go straight to you.

Setup, once per workstream

1. **A working folder outside the repo** with `HOW-WE-WORK.md` (rules every session reads), the workstream map, `STATUS.md`, the `PROMPT-<card>.md` files, the evidence and the scripts. None of it goes into the repo.
2. **An orchestrator session** opened in the repo's main folder (it doesn't need a worktree), with an easy-to-find name and pinned to the top.
3. **One command to switch the environment**, for example `start-env pr-14`: it stops whatever is running and starts the given copy.
4. **One symlink per worktree.** The worktrees the app creates live in a hidden folder (`.claude/worktrees/`); each PR session creates a symlink named after its card, and the start command uses that name.
5. **Extra processes.** If the environment has processes that don't start on their own, the script takes their names as arguments.
6. **Keep the machine awake**, or the environment goes down in the middle of an E2E run.

Organization in the app

- **Orchestrator pinned to the top** of the sidebar.
- **One group per workstream**, with the orchestrator and all the PR sessions inside.
- **Every session named after its card**, with the PR number up front to show the order.
- **Delivered sessions get out of the way**: marked done and, with your OK, archived.

The orchestrator does this itself through the app's tools, as long as its prompt tells it to.

Switching the environment

1. **The PR session requests its turn** in one line: card, symlink, extra processes and what it will run.
2. **Before asking, it has the worktree ready.** Without the setup, the environment starts broken and the turn is wasted.
3. **The orchestrator checks** who has the environment, the queue and the machine's load.
4. **It starts the environment** in a tab of the app's terminal, with the team's command.
5. **It checks where the processes run from** before handing it over (`ps`). A screenshot taken from the wrong server is wasted work.
6. **It tells the session and you** in one line, and updates `STATUS.md`.
7. **The session hands it back** when done ("environment free"), and the queue moves.

Ground rules

- **One environment at a time, always.** Starting one can kill any extra server, including the main-branch server used for the "before" screenshots. Start that one afterward.
- **Nothing heavy while the environment is running.** Global type checks and large builds overload the machine.
- **Infrastructure is shared.** Don't start the database and queues from inside a worktree: that can create a new, empty database.
- **Two PRs in flight at most**: one builds, the other proves and documents. More than that just creates queues and conflicts with the main branch.
- **Tell the team before touching an area**, especially whoever is changing the same screens.

How sessions talk to each other

- Each PR session's prompt has a **Communication** section with the orchestrator's name and the rule: at every checkpoint, blocker or end of a stage, send it a one-line status.
- The orchestrator doesn't poll: sessions report in. It pulls everything together in `STATUS.md`.
- When a PR is merged, the orchestrator tells the remaining PR sessions what changes for them.

Memory: fix it once

Claude Code has per-project memory. When it gets something wrong that you don't want to see again, say "remember this" and explain why. Examples that apply to almost any team:

- don't commit or open a PR unless I ask;
- decide anything technical and reversible on your own, and ask me only about product, visuals, anything involving someone else or anything that can't be undone;
- reach for the design system first, without rebuilding components the platform already has;
- check every screen in both light and dark mode;
- don't run heavy builds while the environment is running.

Over time, the process lives in memories and rules files, and prompts get short.

Known pitfalls

What happens	How to avoid it
A screen in an unexpected theme: the environment was running from an outdated branch	A fresh worktree off main for each card; check where the environment is running from before taking screenshots
A fresh worktree with dozens of "module not found" errors	Run the setup (install and build the internal libraries) before running any checks
A file regenerated by the local server ended up in the PR	Stage only the files in scope; check the diff before pushing
A global check with the environment running overloads the machine	Check only the packages touched; anything heavy runs with the environment stopped
Frontend written against an endpoint the backend hasn't shipped yet	Confirm it exists in the contract or the generated types before starting
A component rewritten from scratch drifted from the visual identity	Reuse the component from the reference screen
Performance measured at different times showed a false regression	Both branches back to back, in the same time window
Switching the environment by hand becomes a waiting line	The orchestrator takes over the switching

Getting started: two paths

Path A: minimal version

One PR, and you drive the session. No orchestration, evidence folder or scripts needed.

1. Install the skill and fill in the team configuration.
2. Pick a small, visual card. A dark mode bug on a single screen is perfect.
3. Open a new session in the app, in the repository folder, named after the card.
4. Paste the filled-in kickoff prompt.
5. Approve the plan before any edit.
6. Follow along in the app's browser, in light and dark mode.
7. Approve the evidence: before and after screenshots, E2E and independent review.
8. Only then ask for the commit and PR.

Path B: automated version

For when the work is bigger than one PR: a whole area, a migration, a set of screens. You talk to **a single session**, the orchestrator, and it does the rest.

What you hand over at the start: the workstream's goal, the parent card, the references (Figma, screens that already have the pattern), what's out of scope and who needs a heads-up.

What it handles on its own:

1. Maps the affected screens and components and writes the map (inventory, PR order, risks, dependencies).
2. Splits the work into small PRs and creates the sub-cards.
3. Writes each PR's `PROMPT-<card>.md`, already including what earlier PRs learned.
4. Opens each PR's session, each in its own worktree. Two in flight at most.
5. Organizes the sidebar.
6. Manages the environment: queue, switching, checks and handback.
7. Pulls the status together in `STATUS.md` and sums up only what's waiting on you.
8. When a PR is merged, tells the remaining PRs what changes and lets the next one in the queue start.

Where you step in:

Moment	What you do
Map and PR split	Approve or adjust the order and size
Each PR's plan	Approve it before any edit

Moment	What you do
A visual or product call along the way	Decide
Each PR's evidence	Approve it
Commit and PR	Say "go ahead, commit and open it"
Reviewer comments	Lead the conversation and decide what changes

Do Path A at least once first, so you know the steps the orchestrator will automate.

Kickoff prompt template

```
Card: <ID> <card title> (<card link>)
Rename this session after the card.

Goal: <what changes for users, in 1 or 2 sentences>
Visual reference: <Figma link and/or a screen that already has the pattern>
Out of scope: <what not to touch>
Branch: <team branch format>

How to work:
- Act as a senior developer: decide anything technical and reversible; ask me only about
product, visuals, anything involving someone else, or anything irreversible.
- Reply in English.
- Fresh worktree off an up-to-date <main branch>; run <worktree setup>.
- <Design system> first. If the reference screen already has the component, reuse the same
one and adjust it only through props and classes.
- Semantic tokens; the screen has to work in light and dark mode.
- Don't change behavior: validation, copy, data-testid and payload stay the same.

Checkpoints:
1. Before editing, send me the plan (files, components, risks).
2. Before committing, show me before and after screenshots (light and dark) and the results
of the area's E2E and checks, and run an independent review of the diff.
Commit and open the PR only when I ask, in this format: <PR format>.
```

Orchestrator session prompt template

```
You are the orchestrator session for the <workstream name> workstream (parent card <ID>).
Rename this session to "<ID> orchestration".
Reply in English. Act as a senior developer: decide anything technical and reversible;
ask me only about product, visuals, anything involving someone else, or anything
irreversible.

Workstream:
- Goal: <what changes for users>
- References: <Figma, screens that already have the pattern>
- Out of scope: <what not to touch>
- Notify before touching: <whoever is on the same screens>
- Working folder (outside the repo): <path>

1. Map and split
- Survey the code (up-to-date main branch) for the affected screens and components.
- Write <folder>/map.md: inventory, PR order, risks, dependencies.
- Propose a split into small PRs and STOP for my approval.
- Once approved: create the sub-cards (titles of about 40 characters), one <folder>/PROMPT-
<card>.md
  per PR (with a Communication section that names you) and a <folder>/HOW-WE-WORK.md
  with the shared rules.

2. PR sessions and organization in the app
- Open one session per PR, named after its card, in its own worktree, telling it to read
  its PROMPT. Two PRs in flight at most.
- Pin this session to the top of the sidebar; create the group "<ID> <workstream name>"
  and move this session and each PR session into it as soon as it's created.
  Once a PR is merged: mark its session as done and ask me whether it can be archived.

3. Environment (you are its sole owner)
- Sessions request their turn by message. Before starting it, check who has the
  environment, the queue and the uptime; if the session hasn't run its setup, send the
request back.
- Start it in a tab of the app's terminal with <environment start command>, check with
  ps that the processes run from the right worktree, and update the local database if it's
  behind. Tell the session, and tell me in one line.
- One environment at a time; nothing heavy while it's running; never start the database
  and queues from inside a worktree.

4. Tracking
- Maintain <folder>/STATUS.md: a table of PRs (card, session, stage, what's waiting
  on me, next step), who has the environment, and a dated log.
- When a PR is merged, tell the remaining PR sessions what changes for them and
  let the next one in the queue start.
- At the end of each stage, send me a short summary with only what depends on me.

You don't write the PRs' code, don't decide visuals, and don't commit or open PRs for them.
```

The skill

The skill packs this process into a single file, written for Claude to read. Save it to `~/.claude/skills/processo-frontend/SKILL.md`, fill in the "Team configuration" section and open a new session. From then on, the kickoff prompt only needs the card, the goal and the reference.

```
mkdir -p ~/.claude/skills/processo-frontend && curl -fsSL
https://eltoncarvalho.com.br/en/processo/SKILL.md -o ~/.claude/skills/processo-
frontend/SKILL.md
```

Elton Carvalho · November 2026