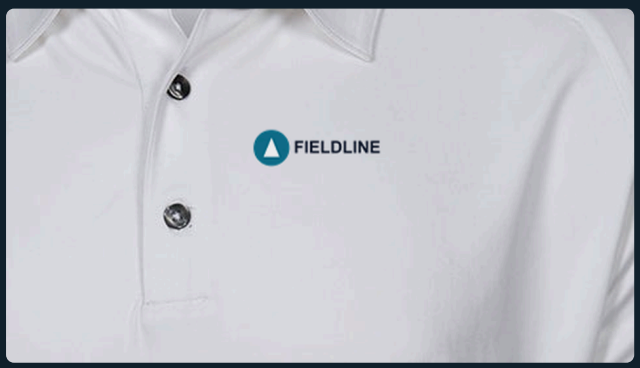
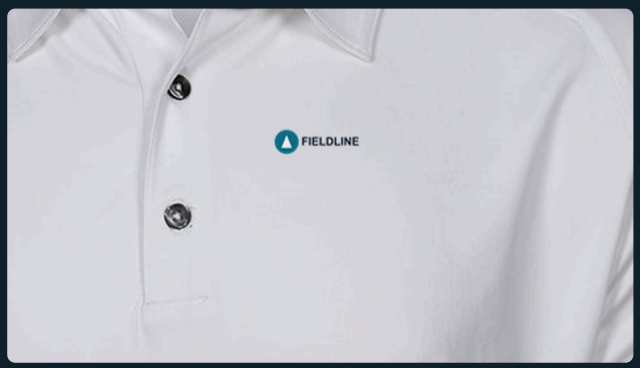
WHITE PAPER · LEAD FLOW AUTOMATION

MCP servers as a product channel

Putting a customer's real logo on real blanks inside Claude — lessons from ScalePOD's decorator server





Gergely Racz
AI architect · PhD Engineering, Cambridge

November 2026
Pilot findings, Oct–Nov 2026

merch.scalepod.ai
ScalePOD decorator server

Executive summary

ScalePOD (merch.scalepod.ai) is a Model Context Protocol (MCP) server for merchandise decorators: the print and embroidery shops that put a company's logo on polos, hoodies, caps and bags. MCP is the open connector standard that lets an AI assistant such as Claude call tools on an outside server. A decorator adds ScalePOD to their own Claude, uploads an end customer's logo, picks blanks from their own catalogue, and gets photo-real mockups, a private review page, a customer page to forward, and print-ready files for the items the customer accepted.

We built it in October 2026 and ran it with one pilot customer, a UK merch decorator. In 14 days that decorator made 298 tool calls, 51 renders, 16 internal and 15 customer review pages, without help from us. That is a pilot with one user, on free credits, with card payments built but not switched on. It is not market proof. It is enough to show which design decisions hold up when a real user drives the tools from a chat window.

Five decisions mattered most, and each one came from a failure:

1. **The client's Claude reasons; the server only renders and computes.** No language model runs on our side.
2. **Nothing paid runs without a quote and a human yes.** Credits are held before a call and settled after it, and paid tools refuse to run without a `go_token` from the quote.
3. **The procedure is served by the server, with hard STOP points.** The steps travel with the connector, and the gates that cost money or reach a customer are enforced in code.
4. **AI never touches a size-critical logo.** The model may suggest placement and lighting. The final logo is the customer's exact file, placed in millimetres by deterministic code. Its worst width error was 0.56 mm over 32 builds.
5. **Review pages are the product surface.** The chat is where the work gets done. The decision itself happens on a page the decorator, and then the decorator's customer, can open on a phone.

Why an MCP server instead of a web app

A decorator's day is already conversational. Customers send a logo by e-mail, ask for "something like last year's polos but in navy", change their minds about the size. A web app would make the decorator translate that conversation into forms. An MCP server lets the decorator keep talking. Claude asks the intake questions, chooses blanks, writes the pitch sentence for each item, and reads back what the customer decided.

The economics follow from the same split. The decorator's own Claude subscription pays for every word of reasoning and writing. Our server pays for what only it can do: image edits on a hosted model (list price $0.08 per edit), logo matching and compositing, print files, review pages and e-mail

notices. The server does contain a small language-model helper, but no decorator tool calls it. We pay nothing for model reasoning.

There is a cost to this choice. Everything the client's model sees can be printed on request: the procedure text, tool descriptions and results. The value has to sit in what never leaves the server. That means the render prompts, the logo-lock algorithm and thresholds, the print-pack rules, the catalogue mapping and the pricing logic. Our rule now is to put the *how* into server code and data, and keep the visible procedure to *what, in which order, with which STOP points*.

Architecture

Figure 1 shows the request path. The decorator talks to Claude. Claude calls tools on the ScalePOD server over HTTPS, after an OAuth sign-in. OAuth is the standard that lets a user grant an app access without sharing a password. The server is one Python process per service, with its own data folder, credit ledger and sign-in issuer. Each decorator gets an isolated workspace with its own brands, blanks, kits, ledger and uploads.

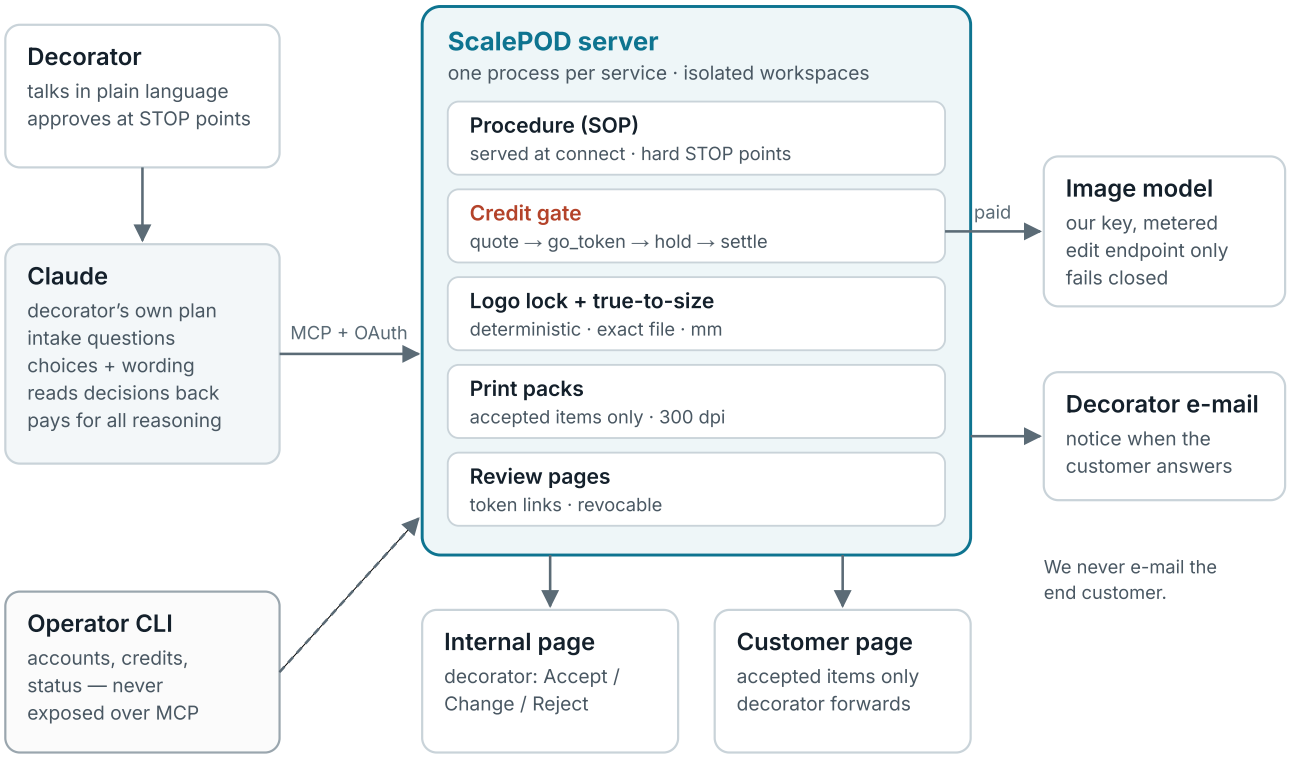


Figure 1. Request path. The decorator's Claude does the reasoning; the server only renders, computes, meters and hosts pages.

Paid work goes to the image model under our key, behind the credit gate. Deterministic work stays in our code: logo lock, true-to-size placement and print packs. Review pages are served from unguessable token links. When the end customer answers, the server e-mails the decorator. We never e-mail the end customer.

Operator actions are not exposed over MCP at all. Creating accounts, granting credits and checking status run only as a command line on the server.

The five design decisions that mattered

I The client's Claude reasons; the server only renders and computes

FAILURE THAT TAUGHT IT The first acceptance run, with the decorator's real logo on his real blanks, scored 19 pass and 3 fail. One failure was the logo itself. Colour drifted by ΔE 12–17 on four of six renders (ΔE is the standard measure of visible colour difference; under about 2 is invisible), and one cap of four had its logo rearranged. On a closer look, the model had also redrawn the letter shapes of a small mark. An earlier "11 of 13 renders kept the logo" count had only checked spelling, order and colour.

WHAT WE BUILT We stopped asking a model to be careful with brand assets. Tools now return facts the client's Claude can act on: a `logo_check` status of passed, failed or skipped, measured millimetres, and credit balances. The client's Claude carries the conversation and handles judgement and wording. Our code does anything that has to be exact.

2 Nothing paid runs without a quote and a human yes

FAILURE THAT TAUGHT IT An early independent review of the build found four blockers. One was a dry-run render that could still be sold. Another was a race between the daily cap and duplicate detection. When a model drives a paid tool in a loop, a retry or a misread instruction costs real money.

WHAT WE BUILT Every metered tool is wrapped the same way. It holds the estimated credits before the call, settles the actual cost afterwards, releases the hold on any error, and removes duplicates by `request_id`. `quote_renders` returns the credit total and a `go_token`. Paid tools refuse without that token, so Claude has to show the cost and wait for the user's yes. Daily caps sit on top, independent of credits: 40 provider calls per workspace by default, 20 renders a day on a trial, and a global cap of 200 across the instance. Rendering fails closed. If a reference image cannot be uploaded to the edit endpoint, the call errors and the hold is released. There is no fallback to a generic text-to-image picture. The acceptance test for a short balance checks the provider's request log as well as the error message: no call reached the provider.

Trials come from invite codes: 60 credits for 14 days, one trial per e-mail domain, with disposable domains refused. Sign-in uses a six-digit e-mail code or a workspace key. Card top-ups (any amount from $50) are built, with a webhook that credits each payment once and holds mismatched payments for review. They are not switched on yet.

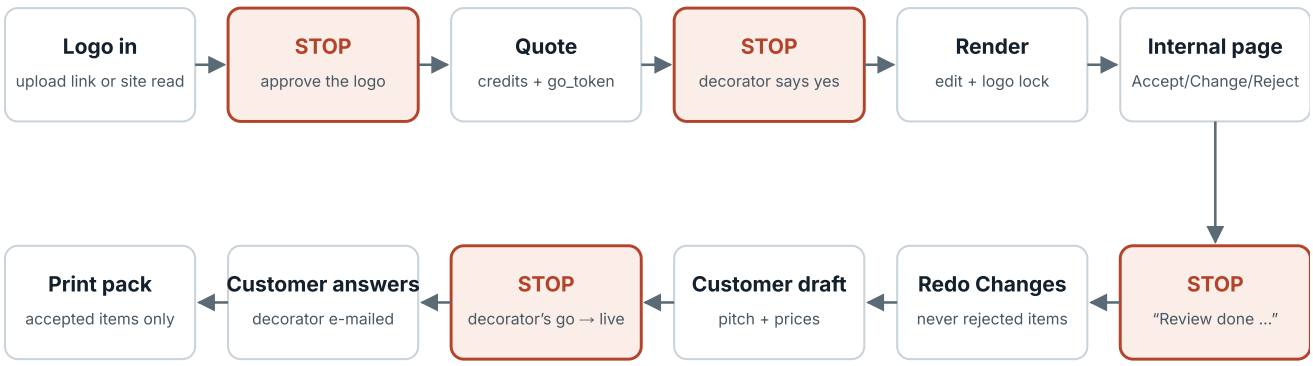
3 The procedure is served by the server, with hard STOP points

FAILURE THAT TAUGHT IT In the first specification, the single review page went straight to the end customer, and an early tool could send it to them. That left the decorator with no control over what his customer saw. A review of the build also flagged a second risk: text the customer typed on that page ("client notes") came back into Claude's context, where it could carry instructions.

WHAT WE BUILT The server sends its procedure, an SOP (standard operating procedure), in two places. One is the instructions block delivered at connect, about 700 words. The other is a fuller `get_sop('d1-decorator-kit')` document of 862 words, also exposed as an MCP prompt and resource. The flow has hard STOP points (Figure 2):

- the logo is shown and approved before anything renders;
- the cost is quoted and confirmed before any paid call;
- the decorator marks every mockup Accept, Change or Reject on a private page, then types, word for word, `Review done for kit <kit_id>`;
- only renders with a sent Change are regenerated, never rejected ones;
- the customer page starts as a draft, and only the decorator's go makes it live;
- print files are made only for items the customer accepted.

The server returns the customer's own words in a field named `client_note_untrusted`, and the procedure tells Claude to quote that text, never to follow it. Gates that cost money or reach a customer are enforced in code. The prose only explains them.



STOP = Claude must wait for the decorator. The money and customer-facing gates are also enforced in server code.

Figure 2. The decorator procedure (SOP) served by the server, with its four hard STOP points.

4 AI never touches a size-critical logo

FAILURE THAT TAUGHT IT In early November the pilot decorator gave detailed feedback. The mockups looked right, but he needed the logo at its true size in millimetres, because that is what his embroidery machine or press will produce. We built a deterministic placement engine and tested it, including one tempting shortcut: placing the exact logo first, then letting an AI model add realism inside a 3 mm mask around it. The shortcut failed badly. Logo overlap with the original fell to an IoU of 0.76 and 0.47 (IoU, intersection over union, is 1.0 for a perfect match). The logos grew by 2.8 and 4.9 mm. A wordmark was rewritten as gibberish, and a pictogram was redrawn.

WHAT WE BUILT For mockups, a *logo lock* step runs after the model's render. It finds where the model drew the logo, first by feature matching and then by a colour-region fit for small marks. It checks the layout (IoU must be at least 0.55), removes the model's version and pastes the customer's exact file back with the fabric's shading. A failed check gets one automatic re-render that we pay for. If that also fails, the result is marked "check before sending". When the fixed lock was re-run on the same raw renders, logo colour error fell to ΔE 0.8–2.7 on two caps, a hoodie and a polo. A tote failed the check, so in live use it would get the free retry. The raw model render is stored next to every version for audit.

For true-to-size work there is no AI on the logo at all. Code converts millimetres to pixels from a per-blank calibration and places the logo, which sits as a smart object (a layer that keeps the original file inside it). It then writes a layered Photoshop document (PSD) the decorator can edit, plus a preview, 300 dpi artwork and a placement record. On the test suite:

- width error was at most 0.56 mm over 32 builds, and position error at most 0.36 mm;
- five reruns were pixel-identical, and a Linux build matched pixel for pixel;
- the embedded logo was byte-identical to the approved file, and a vector trace scored IoU 0.993;
- whole-logo colour error was ΔE 0.27;
- a build took about 1 second and 181 MB of memory, and four parallel jobs ran cleanly.

Realism layers (photo matching, thread texture, fabric shading) are clipped to the logo. A dedicated test showed they change its width by 0.000 mm.

Figure 3 shows the same engine placing a fictional logo at 60 mm and at 90 mm on a white polo blank.



60 mm wide · 17.1 mm tall

90 mm wide · 25.6 mm tall

Figure 3. The deterministic placement engine putting a fictional logo on a white polo blank at two widths, left chest, centre 100 mm from the placket, top 180 mm below the shoulder point. The store photo resolves only about 1.07 pixels per mm, which is why the logo looks soft here; the 300 dpi artwork and the layered PSD from the same run carry the original logo file. Garment calibration (540 mm chest) is assumed until the size chart is read.

5 Review pages are the product surface

FAILURE THAT TAUGHT IT We built an in-chat panel as an MCP App: cards with buttons inside Claude. We attached it to the render and review tools, but we have not yet confirmed that it renders in the real claude.ai host. In one of our own test sessions Claude never called the panel tool. In another, a request for a client brief produced a text document with no mockups. The chat window is good for doing the work. It is not reliable for showing the work.

WHAT WE BUILT Two pages per kit, both on token links that expire or can be revoked:

- an **internal page** where the decorator marks each mockup Accept, Change (a note is required) or Reject;
- a **customer page** that shows only the accepted items, with the decorator's pitch sentences and prices (Claude is told to ask for prices, never to invent them). The page carries the decorator's name and logo and a small "made with ScalePOD" line.

The customer answers on their phone. The server e-mails the decorator, and Claude reads the decisions back on request. Print packs (300 dpi artwork, screen-print films, a digitiser brief for embroidery, a proof PDF) are made only for items the customer accepted.

Results from the pilot

Measure (one UK merch decorator, 14 days)	Value
Tool calls	298
Renders (`render_on_blank`)	51
Quotes before paid steps	25
Internal review pages / customer pages	16 / 15
Blanks in the decorator's workspace	125
Kits	2
Provider (image-model) calls	55
Credits remaining	880 of 1,000 granted

Engineering results behind it:

- 629 automated tests passing at the handover point.
- Logo capture without paid logo services, tested on 38 public sites: 26 correct, 4 mark only, 8 refused and 0 wrong company. The capture never tries to get past bot protection; a challenge page ends the attempt and returns an upload link.
- True-to-size placement: 18 tests pass, 2 fail and 9 are open. Both failures are about realism, not geometry.

What the numbers mean: one decorator, unprompted, used the full loop, from logo to quote to renders to internal review to customer page, many times over. The ratio of 25 quotes to 51 renders shows the quote gate did not stop him from working. The 31 review pages show that pages, not the chat, were where decisions happened.

Limits, and what we would do differently

The design reviews we commissioned against our own work were blunt, and they were right:

- **n = 1, unpaid.** One decorator, on 1,000 free credits, used 120 credits in 14 days. Card payments are built but not connected, and nobody has paid yet. The pilot proves the workflow, not willingness to pay. Next time we would switch payments on before the first user arrives and agree a trial end date.
- **Estimates slipped.** Phase A was specified at about three working days. It took from 14 to 23 October before the customer could use it, and fixes continued into November. Most of the extra

time went into real-host checks, reviews and deploy cycles, not code.

- **The client's Claude is not under our control.** Claude in the browser, the desktop app, Cowork and Claude Code follow instructions differently, and model updates change behaviour. Paid gates are enforced in code; everything else is best effort. We would test in the real host first, not last.
- **Datacenter IPs are blocked.** The online-store platform refused our server's IP address for every store we checked during product import, while the same pages loaded from an office PC. We moved store reads to the client's own Claude and only fetch photos on the server.
- **Long jobs need a job model.** Image edits return in seconds. A minute-long video render or a large crawl would hit tool-call timeouts. A future service needs a start-and-check-back pattern from day one.
- **Inputs limit realism.** Store photos came at about 1 pixel per millimetre, so a 70 mm logo was 66 pixels wide. Upscaling only the blank gave 2.8–3.2 px/mm. Some blanks were line drawings or side views, and chest widths for calibration were assumed until size charts arrive. Non-AI embroidery realism on black fabric still scored below our bar in blind tests.
- **We see tool calls, not the prose.** Quality problems in what the client's Claude writes cannot be diagnosed from our logs.

Checklist for teams building their own MCP server

1. Decide what the client's model does and what your server does. Give language to the client and anything exact or paid to the server.
2. Put a quote step in front of every paid tool, and make the paid tool refuse without the quote's token.
3. Hold before, settle after, release on error, and remove duplicates by request id. Add daily caps independent of credits.
4. Fail closed. Never return a cheaper or fake result when the real one fails.
5. Serve the procedure from the server, with STOP points. Enforce every gate that costs money or reaches a third party in code, not prose.
6. Keep operator functions off MCP entirely.
7. Treat anything a third party typed as untrusted. Label it in the field name and tell the model to quote it, not follow it.
8. Keep the *how* in server code and data. Assume the visible procedure will be copied.
9. Never let a generative model alter an asset with legal or physical meaning: logos, sizes, prices.
10. Ship a review page. Decisions need a URL a non-user can open on a phone.
11. Test outputs, not status codes: provider logs, pixel and millimetre measurements, ledger rows.
12. Get an independent adversarial review of every build stream before it is deployed, and test in the real host before you call it done.

About the author

Gergely Racz is an AI architect and developer based in Budapest, with a PhD in Engineering from Cambridge and more than ten years in AI. He started in machine learning and now builds AI agent systems, MCP servers and full-stack automation for sales, marketing and operations teams through Lead Flow Automation. ScalePOD is his own product. This paper describes it as built and as used, including where it fell short.

All figures come from internal build records, acceptance-test evidence and design reviews dated 14 October to 4 November 2026. The pilot customer is not named. The logo in Figure 3 is fictional.