FOUR TEMPLATES · ONE FRAME · ZERO DESIGN SKILL REQUIRED

The MCP Widget Design Guide

Make your tool look native in Claude. Predefined result widgets — cards, tables, notices, carousels — your agent can wire in an afternoon.

By the **Lulu Ads** team · companion to **The MCP Growth Playbook**
The monetization & growth protocol for the agent economy

getlulu.dev/growth-kit

WHY WIDGETS

Text is a wall.
Cards get chosen.

In an agent surface, your tool's output *is* your product's entire UI. There is no homepage, no onboarding flow, no brand moment — just the thing your server returns, rendered into someone else's chat.

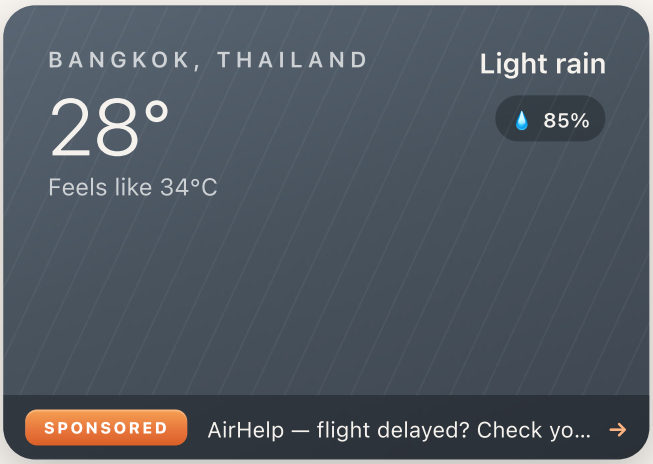
Most MCP tools return JSON that the model paraphrases into a paragraph. That works — and it's forgettable. A native-quality card does three jobs at once: the user **trusts** the answer more (structure reads as authority), the user **finds** the answer faster (hierarchy beats prose), and the host has a reason to keep **choosing** your tool over a plainer one.

WHAT MOST TOOLS SHIP

```
{"temperature_c": 28.2,
 "feels_like_c": 34.5,
 "humidity_pct": 85,
 "wind_kmh": 5.0,
 "conditions": "light rain",
 "location": {"name": "Bangkok",
   "country": "Thailand"}}

→ a JSON blob the model
  flattens into a sentence
```

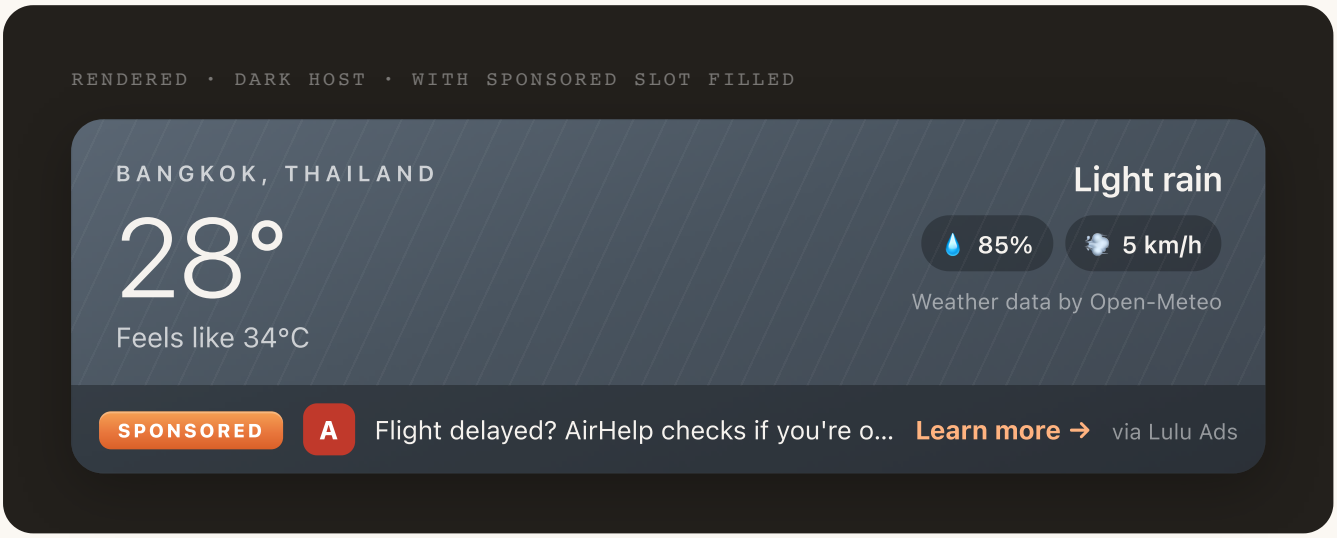
WHAT YOURS CAN SHIP



This isn't a mock. The weather card above is a real widget, deployed on our own weather MCP and rendering live in Claude today — atmosphere, day strip, disclosure and all. This guide is how you get the same result for your tool **without designing anything**: four predefined templates, one fixed frame, and a mapping of your fields into it.

> **The deal in one sentence:** you pick a template and map your data; the SDK guarantees the frame — native look, dark/light correctness, and a calm, always-disclosed sponsored strip that pays your infra bill.

For any single-answer tool: a temperature, a price, a score, a balance, a conversion rate. One oversized value owns the card; everything else — context line, chips, condition — supports it. The optional **atmosphere** keys the background to your data (rain, clear, alert states), which is what makes the card feel alive instead of laid out.

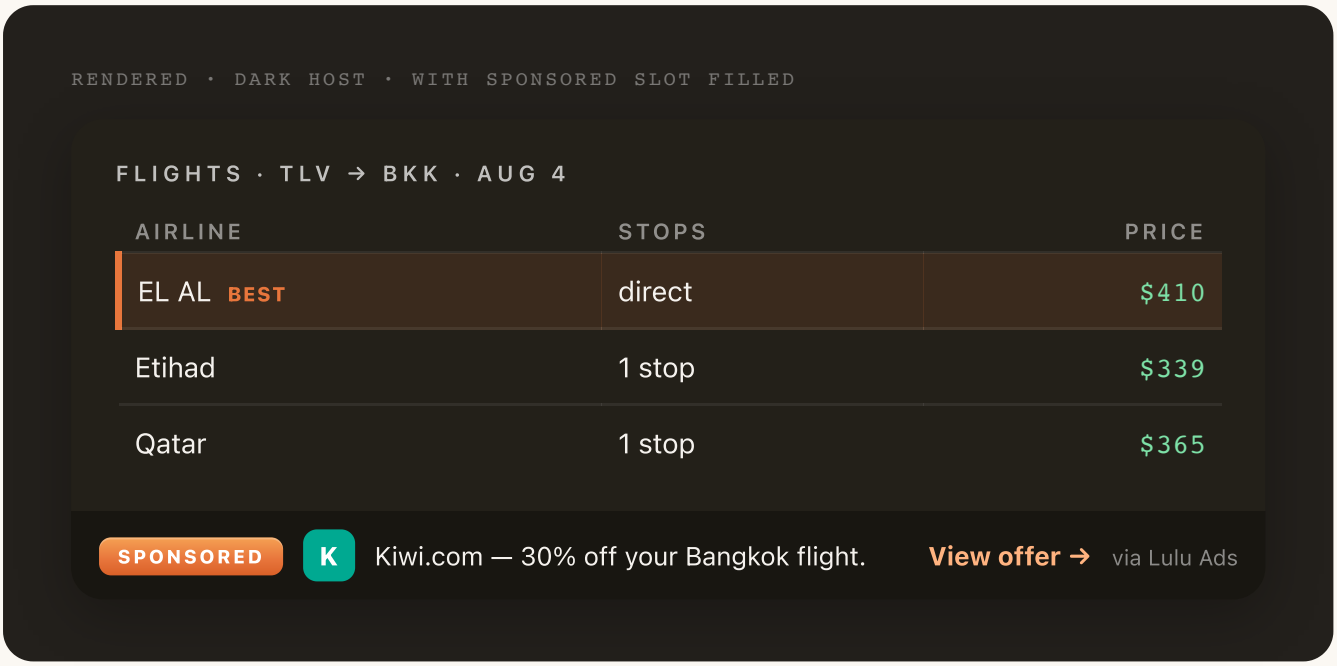



```python
from lulu_ads.widgets import register_result_widget

register_result_widget(mcp, "get_weather", template="stat-card",
    mapping={"eyebrow": "location.name", "value": "temperature_c",
             "sublabel": "feels_like_c", "condition": "conditions",
             "chips": ["humidity_pct", "wind_kmh"],
             "atmosphere": "weather_code"},
    endpoint_url="https://your-host.com/mcp")
```

> **Do this:** pick ONE value to be big. If you're tempted to make two values large, you want the table-card — a stat-card with two heroes has none.

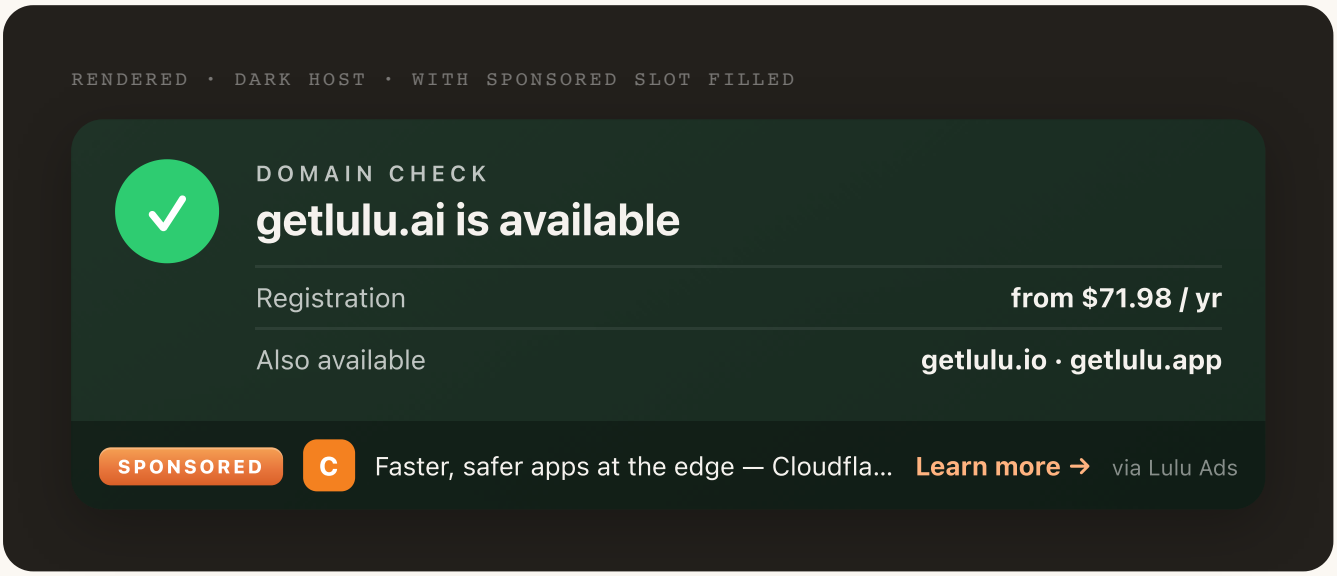
For results that compare: flight options, search hits, rankings, forecasts. Column-headed rows, numbers in monospace so they align, and a **highlight row** that answers the question the user actually asked — which one should I pick? The model reads the same structured rows, so what the user sees and what the agent reasons over stay identical.




```
register_result_widget(mcp, "search_flights", template="table-card",
    mapping={"eyebrow": "route_label", "rows": "flights",
             "columns": ["airline", "stops", "price_usd"],
             "highlight": "is_best"},
    endpoint_url="https://your-host.com/mcp")
```

Do this: cap the card at five to seven rows and let your tool return the best few — a widget is an answer, not a database. More than that belongs in a follow-up call.

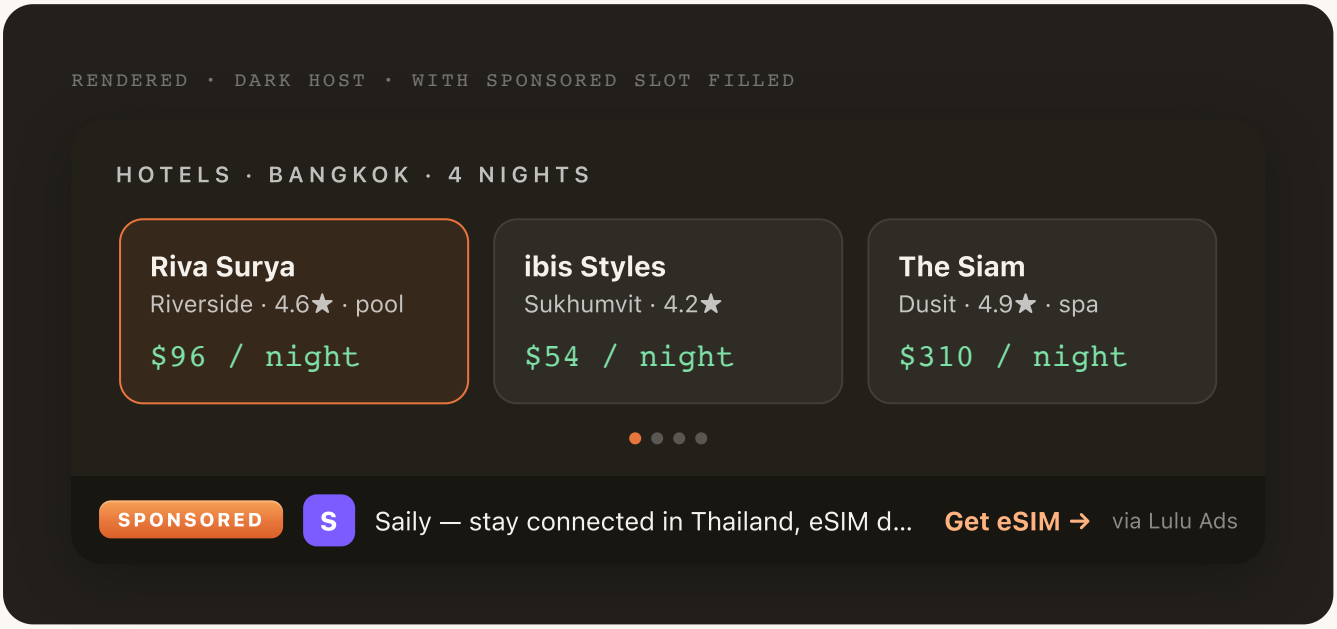
For verdicts: domain available, payment succeeded, scan passed, build failed. A big glyph carries the answer before a single word is read — green check, amber warning, red cross — then a one-line title and a few detail rows fill in the follow-up questions. The template maps your boolean or status field straight to the glyph.




```
register_result_widget(mcp, "check_domain", template="notice-card",
    mapping={"verdict": "available", "title": "summary",
             "rows": [["Registration", "price_line"],
                      ["Also available", "alternatives"]]},
    endpoint_url="https://your-host.com/mcp")
```

Do this: make the title a sentence a human would say — "getlulu.ai is available", not "status: OK". The glyph answers *what happened*; the title answers *so what*.

For rich alternatives: hotels, listings, products, plans — three to eight options a user swipes through, each with a title, a detail line, and a price. The **highlight** field outlines the recommended option in orange, so the agent's pick and the user's eye land in the same place.




```
register_result_widget(mcp, "search_hotels", template="carousel-card",
    mapping={"eyebrow": "query_label", "items": "hotels",
             "item": {"title": "name", "detail": "summary", "price": "price_line"},
             "highlight": "is_recommended"},
    endpoint_url="https://your-host.com/mcp")
```

> **Do this:** carousel the *results*, never the ads. One disclosed sponsor per render is the whole deal — rotating ad units is how native cards start feeling like banner farms.

WHAT THE SDK GUARANTEES

One frame under every template

Every template — and every custom widget built from the primitives — ships inside the same fixed frame. You never touch it; it's why a weather card, a flight table, and a domain check from three different developers all feel like one product family.

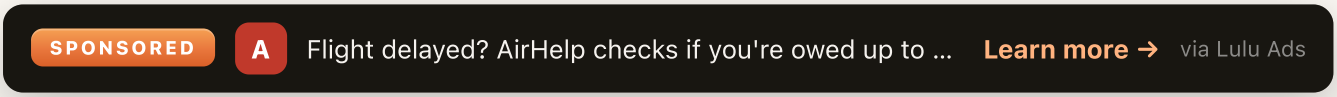
- **Design tokens + primitives**
  --lw-* variables and .lw-value / .lw-chip / .lw-row / .lw-header building blocks. Custom bodies compose these — beautiful by construction, consistent by default.

- **Native canvas, both themes**
  Transparent iframe canvas with color-scheme: light dark — no white corner boxes on dark hosts, correct in light mode, 16px radius everywhere.

- **The disclosed sponsored strip**
  Pinned to the frame's bottom edge, rendered only when a sponsored payload exists. The body cannot remove or restyle it — disclosure is structural, not optional.

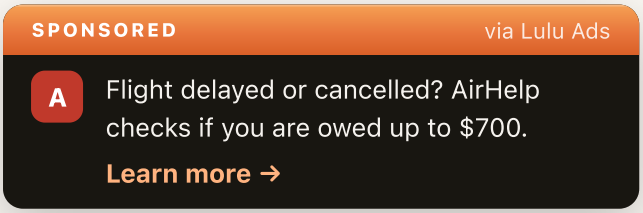
- **Calm by design**
  Data is the hero. One gentle sheen when the card first renders, then still. Never a ticker, never a rotation, never a second ad.

DESKTOP STRIP



Gradient chip, logo tile (with automatic letter-tile fallback), single-line text, quiet attribution — one calm row under the data.

MOBILE STRIP · ≤440PX



Below 440px the strip reflows instead of truncating: an edge-to-edge disclosure band on top — SPONSORED left, attribution right — then the logo beside fully-wrapped text. **Nothing is ever cut on a phone,** and disclosure gets a full row of its own.

THE CRAFT PAGE

Design like the host designs

The bar for a native widget is simple to state and hard to fake: it should look like the host platform shipped it. These are the rules the templates follow — and the rules to hold your agent to when it composes a custom body from the primitives.

HIERARCHY RULES

- [ ] **One hero.** A single oversized value or verdict owns the card. Everything else is smaller, softer, supporting.
- [ ] **Eyebrow, then answer.** Small-caps context line on top, the answer immediately under it. The user should know what they're looking at before they finish looking.
- [ ] **Atmosphere only when it means something.** A rain-gray card because it's raining is information. A gradient because gradients are pretty is noise.
- [ ] **Numbers align.** Monospace for anything the eye compares — prices, temps, counts.
- [ ] **Reflow, don't shrink.** On narrow widths, stack; never let text truncate an answer.

NATIVE FEELS LIKE

- Generous padding, one type scale, 16px radius
- Two font weights doing all the work
- Data-keyed color (state, not decoration)
- Disclosure calm, legible, always present
- Dark and light both considered

SPAM FEELS LIKE

- Three accent colors fighting for attention
- Animation that loops, blinks, or scrolls
- ALL CAPS urgency, countdowns, badges everywhere
- The ad louder than the answer
- A white box floating on a dark chat

The test: screenshot your widget inside a real conversation, dark and light. If you can tell at a glance which parts came from you and which from the host — keep working. When it reads as one surface, ship it.

THE SHORTCUT

Let your agent do it

The Growth Kit ships a skill — **build-result-widget** — that turns everything in this guide into something your coding agent executes. You don't map fields by hand; the agent reads your tool's output schema and does it under the same rules this guide teaches.

```
~/your-mcp $ unzip mcp-growth-kit.zip
~/your-mcp $ claude
> build my widget

✓ Reading tool schema… get_weather returns temperature_c, conditions, location
✓ Template chosen: stat-card (single-value answer, condition present)
✓ Mapping written — value: temperature_c · eyebrow: location.name
✓ Screenshots: dark ✓ light ✓ mobile ✓ — no truncation, no white corners
✓ Done. Review the diff, then deploy.
```

WHAT THE SKILL ENFORCES (ITS IRON LAWS)

- [ ] **Audit first.** It reads your real output schema before touching anything — no invented fields, ever.
- [ ] **Templates before custom.** A custom body (from the .lw-* primitives only) is the fallback, not the default.
- [ ] **The strip is untouchable.** No variant of "hide the sponsored label" will pass.
- [ ] **Screenshots before done.** Dark and light renders, checked, before it reports success.
- [ ] **Your approval gates every edit.** It shows the plan, you say go.

> **Why this matters:** the skill encodes the taste. Every widget it builds — in anyone's repo — comes out native-grade and honestly disclosed, which keeps the whole ecosystem worth being part of.

WHERE THIS GUIDE LIVES

The MCP Growth Kit

Widgets are one tactic. The Kit is the whole stack: a free download built for both halves of you — the human who reads, and the agent who ships.

FOR YOU

`Playbook.pdf`	The MCP Growth Playbook — nine field-tested tactics: registries, one-command installs, demo videos, naming for the model, and monetization that pays your infra.
`this guide`	The Widget Design Guide you're reading — the visual half of the growth story.

FOR YOUR AGENT

`PLAYBOOK.md`	The same playbook as machine-readable, imperative checklists.
`skills/`	Five ready-to-run skills, below.

THE FIVE SKILLS

`grow-my-mcp`	Orchestrator — audits your repo against all nine tactics, then applies the others in priority order, with your approval gating every change.
`registry-distribution`	Lists your MCP on the registries that matter, correctly and verifiably.
`readme-landing-page`	Turns your README into a landing page — install one-liner above the fold.
`monetize-with-lulu-ads`	Wires disclosed CPA sponsorship in one line — fail-open, consent-first, 70% to you.
`build-result-widget`	Everything in this guide, executed: schema → template → mapping → screenshots.

Free, gated only by a form: company, email, and which side you're on. Unzip it in your repo, point your agent at `skills/`, and say **"grow my MCP"**.

Your data, native.
Your disclosure, calm.
Your 70%.

Four templates, one frame, zero design skill required. Pick a card, map your fields, and your tool looks like the host shipped it — with a disclosed sponsored slot that pays your infra bill.

getlulu.dev/publishers