TheRecLab.

Candidate Pack Skill (Example Template)

In plain English: what this skill does

You run an intake call and your note-taker writes it up in Notion. This skill turns that one Notion page into a small private website (a "candidate pack") for that specific role and puts it on the live internet — all in one go.

Step-by-step, no jargon:

1. Find the right meeting note in your Notion.
2. Read it and pull out the role-specific bits (title, salary, who they'd report to, interview stages, etc.).
3. Make a fresh copy of the pack website template.
4. Swap in the new role's content.
5. Test that the website still builds with no errors.
6. Save the new copy to GitHub (private, so the salary stays internal).
7. Publish it live on Vercel and turn off the login wall so candidates can actually open the link.
8. Return the live URL.

Every pack is its own standalone thing — its own GitHub project, its own Vercel project. **The shared template is never touched as part of pack generation.**

Critical user identity config

> IN PLAIN ENGLISH
>
> These are the fixed details the tool needs before it can do anything: your GitHub account, the name and email stamped on every save, the master template it copies from, and the rule that every pack stays private. They're set once and sit at the top because every step after this leans on them. Get one wrong here and the error ripples through the whole run, so they're locked down first.

These never change. They're listed first because every step downstream assumes them.

- **GitHub username:** `[YOUR_GITHUB_USERNAME]`
- **Git author email** (use this for ALL commits, every time): `[YOUR_EMAIL]`
- **Git author name:** `[YOUR_NAME]`
- **Template repo to clone from:** `https://github.com/[YOUR_GITHUB_USERNAME]/[YOUR_TEMPLATE_REPO].git`
- **Repo visibility:** every generated pack repo MUST be private. Always pass `--private` to `gh repo create`. These packs include candidate-facing salary bands and internal process details — never push them to a public repo.

> ⚠️ **Why the author email matters.** Vercel's free plan ("Hobby") silently fails to publish a site when the email on a commit doesn't match the email on the Vercel account. So every commit MUST be authored as `[YOUR_EMAIL]`. Two ways to make sure:

- Set it for one command: pass `-c user.email=[YOUR_EMAIL] -c user.name="[YOUR_NAME]"` on the git command, OR
- Set it for the whole repo: run `git config user.email [YOUR_EMAIL]` inside the cloned repo before any commits.

Never invent an email.

Push policy: which repos can you ship directly to?

> IN PLAIN ENGLISH
>
> There are two kinds of project here and they get very different treatment. A single candidate pack is disposable, so its changes go live straight away, no ceremony. The master template is the opposite: it feeds every future pack, so one careless change to it would break them all. That's why the template only ever gets touched carefully, reviewed first, and never as part of a normal run. Same instinct as editing one live job ad versus redrawing your entire careers page.

Two kinds of repos are involved here. Treat them very differently.

- **Per-role pack repos** (e.g. `founding-engineer-pack` — every repo this skill creates from the template): **ship directly to `main`**. These are throwaway demo artefacts; don't open a branch, don't ask to merge — `gh repo create ... --push` lands on `main` and that's fine.
- **Template repo** (`[YOUR_GITHUB_USERNAME]/[YOUR_TEMPLATE_REPO]`): **never push directly to `main`**. One bad change here silently breaks every future pack the skill generates. Always work on a feature branch, push the branch, and ask the user to review/merge. The skill itself should not modify the template repo as part of normal pack generation; only touch it when the user explicitly asks for a template fix.

End-to-end flow

1. Find the right Notion page

> IN PLAIN ENGLISH
>
> First it has to find the right meeting note. It searches your Notion for the role you asked about and, because you might have run the same role more than once, it puts the most recently edited note at the top so it doesn't grab a stale old version by mistake. It won't just pick for you either: if there's any doubt, it shows you what it found and waits for a yes. A quick confirm is cheap insurance against building a pack off the wrong note and sending the wrong salary to a candidate.

Use the Notion MCP `notion-search` tool. Build the query from what the user said: extract the role keyword (e.g. "founding engineer", "TA partner") and any context clue (e.g. "call with [name]", "[name]'s intake").

```
notion-search query="<role keyword> intake" filters={}
```

Each result carries a `timestamp` (last-edited time). **Always sort results by `timestamp` descending (newest first) before displaying** — the most recently edited intake is almost always the one the user means. This matters most when an old role is being re-opened and there are duplicate intake notes from earlier rounds.

Behaviour by result count:

- **0 matches** — ask user to confirm the role name or paste a Notion URL directly
- **1 match** — show the page title *and the last-edited date* and ask user to confirm before proceeding ("Found: 'Intake - Founding Engineer', last edited 2026-05-06. Use this?"). The date is important in case the search lands on a stale duplicate.
- **Multiple matches** — list all matches with titles + last-edited dates, **newest first**. Ask user to pick by number.

NEVER auto-pick when there are multiple matches. Always ask, even when the top match is clearly the most recent.

2. Fetch the full page contents

> IN PLAIN ENGLISH
>
> Once you've confirmed the right note, it opens it and reads the whole thing, cover to cover. Not just the tidy summary at the top, but the raw transcript underneath too, since the details worth putting in the pack (an offhand comment about the team, the real reason the role's open) often live in the messy notes rather than the neat write-up.

Once confirmed, use `notion-fetch` with the page ID. Read the whole page — both the transcript (if present) and any summary block.

3. Extract structured role data

> IN PLAIN ENGLISH
>
> This is the heart of it. The tool reads the intake note and lifts out everything that changes from one hire to the next: the job title, the salary, who the person reports to, the interview stages, and so on. It drops each of those into the matching slot on the site, like filling in a very detailed form. Crucially, it only touches the role-specific slots. Your brand, your benefits and your interview pledge are company-wide, so they're left exactly as the template wrote them and every pack stays on-message. One thing it's strict about: the salary shows up in two places on the page, so it forces them to match, otherwise the pack could quote a candidate two different numbers.

From the page contents, extract these fields. The transcript will contain all of them — they're all role-specific. Read it carefully.

Here are the fields we need you to populate in the code:

```
hero: {
  // Eyebrow text — adapt to the role type. E.g. "Our [Role Title] Job Pack"
  eyebrow: string;
  line1: "Hi, we're hiring a"; // always
  line2: string; // the role title with gradient applied
  line3: "at [Company]."; // always
  subhead: string; // can stay as default intro text unless transcript
                   // says otherwise
  ctaPrimary: "Jump to the process";
  ctaSecondary: "Read the role";
}
role: {
  title: string; // e.g. "Founding Engineer"
  location: string; // use literal "\n" between line 1 and 2.
                    // E.g. "Farringdon, EC1V\n(Tuesday and Thursday)"
  employmentType: string; // usually "Full-time"
  reportsTo: string; // e.g. "[Name], [Title]"
  equity: string; // e.g. "You'd be employee #6". Extract the
                  // actual employee number from the transcript —
                  // do NOT default to a fixed number.
  startDate: string; // e.g. "ASAP · flexible for notice"
  holiday: string; // usually "28 days + bank holidays"
  benefits: [...] // 4 cards, see template — usually unchanged
                  // unless transcript mentions different perks
  hiringManagerFirstName: string; // first name only. Pick the person
                                  // who'd own the no-ghost SLA.
                                  // This gets substituted into the pledge.
  snapshot: string; // 2-sentence pitch from the transcript
}
comp: {
  headline: string; // e.g. "£XX,000 - £XX,000 base · meaningful equity"
  // Update comp.items[0].value (Base salary) to match the headline band —
  // these two MUST agree or the pack contradicts itself.
  // Update comp.items[4].value (Work setup) to match role.location — the
  // template default will contradict any role with mandated office days.
  // Rewrite to fit.
  // Leave the rest of comp.items alone (company-wide benefits).
}
scorecard: {
  // Rewrite each competencies[*].body to fit the role.
  // The competency `name` fields (Craft, Judgement, Collaboration, Drive)
  // are universal — leave those alone unless the transcript explicitly
  // says otherwise. Also leave scorecard.rubric and scorecard.decisionProcess
  // alone — those are company-wide.
  competencies: [
    { name: "Craft", body: string },
    { name: "Judgement", body: string },
    { name: "Collaboration", body: string },
    { name: "Drive", body: string },
  ]
}
prep: {
  // Rewrite prep item bodies to fit the role. Some default items may be
  // role-specific in the template — skim them and adjust if any phrasing
  // reads off for a different role.
  items: [
    { title: string, body: string }, // 4 cards
  ]
}
responsibilities: {
  heading: "What you'll actually do";
  items: string[]; // 6 bullets — extract from the transcript
}
youAre: {
  heading: "You'll thrive here if…";
  items: string[]; // 6 bullets — extract from the transcript
```

```
}
process: {
  heading: "The interview process";
  subhead: string; // e.g. "Four touchpoints. Roughly 2-3 weeks
                   // end to end. ..."
  stages: [ // 4 stages, each with:
    {
      step: "01" | "02" | "03" | "04";
      title: string; // e.g. "Intro call", "Technical deep-dive"
      duration: string; // e.g. "30 min", "60 min"
      format: string; // e.g. "Video call · Google Meet"
      with: string; // e.g. "[Name], [Title]" — for stages
                    // with two interviewers, "[Name] and [Name]"
      teaser: string; // 1-line preview shown when collapsed
      what: string; // 2-3 sentence description
      assessing: string[]; // 3-4 bullets of what we're looking for
      prep: string; // 1-2 sentence guidance for the candidate
      decision: string; // SLA + feedback policy, e.g. "Go / no-go
                        // within 24 hours, written feedback either way"
    }
  ]
}
```

Do NOT touch these keys (they're company-wide and stay identical between every pack):

- `brand`, `company`, `team`, `glassdoor`, `pledge`, `ai`, `next`
- Inside `scorecard`: `rubric` and `decisionProcess`
- Inside `comp`: every `items[*]` other than `[0]` (Base salary) and `[4]` (Work setup)

The pledge contains a `{hiringManager}` placeholder that the website fills in automatically using `pack.role.hiringManagerFirstName`. So as long as you set `hiringManagerFirstName` correctly in step 3, the pledge will name the right person — don't edit the pledge body itself.

4. Pick a slug

> IN PLAIN ENGLISH
>
> A "slug" is just the short, tidy, web-friendly name for the pack, all lowercase with dashes: "Founding Engineer" becomes `founding-engineer-pack`. It's the label that names the project and shapes the eventual link. Before committing to it, the tool checks the name isn't already taken, and if it is, adds a number on the end so a new pack can never overwrite an existing one. Whatever name it lands on then gets reused everywhere for the rest of the run, so nothing gets crossed.

Auto-generate from the role title:

- "Founding Engineer" → `founding-engineer-pack`
- "Senior Product Designer" → `senior-product-designer-pack`
- "VP of Sales" → `vp-of-sales-pack`

Lowercase, dashes, ASCII only, suffix `-pack`. Check the slug doesn't collide with an existing GitHub repo — if it does, append `-2` etc.

Once you've resolved the final slug (with any suffix), **use it everywhere downstream** — `<slug>` in steps 5, 6, 9, 10, 11 all means the suffixed slug, not the bare auto-generated one. A common bug is wiping `/tmp/candidate-packs/founding-engineer-pack` then creating `founding-engineer-pack-2` and losing track of which is which.

5. Create the new repo from the template

> IN PLAIN ENGLISH
>
> Now it makes a fresh copy of the master template, one brand-new project just for this role, and pulls a working copy down onto the machine to edit. Before it does, it clears out any leftover folder from a previous attempt. That housekeeping matters more than it sounds: a stray folder from a failed earlier run can secretly wire the new pack up to the wrong live site, so wiping the slate first avoids a genuinely confusing mix-up later.

The template repo is marked as a template repository on GitHub, so `gh repo create --template` creates the new repo + clones it in one step. Always wipe any leftover folder from a previous run first — a stale `.vercel/` directory from a previous attempt can silently link the pack to the wrong Vercel project.

```
rm -rf /tmp/candidate-packs/<slug>
mkdir -p /tmp/candidate-packs
cd /tmp/candidate-packs
gh repo create <slug> --private --template
[YOUR_GITHUB_USERNAME]/[YOUR_TEMPLATE_REPO] --clone
cd <slug>
git config user.email "[YOUR_EMAIL]"
git config user.name "[YOUR_NAME]"
```

The repo is created on GitHub with the template's content as a clean initial commit. No need to wipe `.git` or re-initialise — the clone is already a fresh repo on `main` tracking the new remote.

6. Edit pack.ts with the new role data

> IN PLAIN ENGLISH
>
> All the words on the site live in one file, called `pack.ts`. Think of it as a fill-in-the-blanks form: it has a labelled slot for the job title, one for the salary, one for each interview stage, and so on. The tool opens that form and types the new role's answers into the right slots, using what it pulled from the intake back in step 3. It only fills the slots that change per role and leaves the rest of the form (your brand, benefits, interview pledge) untouched. And it changes those lines one by one rather than retyping the whole form, because starting the form from scratch would wipe all the parts you wanted to keep.

The role-specific fields all live in `src/content/pack.ts`. Edit ONLY those keys (see step 3). Use the `Edit` tool — **don't use `Write`** because it wipes the whole file and loses the company-wide content. Read the file first to confirm the current shape, then make targeted edits.

7. Local build check (don't skip)

> IN PLAIN ENGLISH
>
> Before anything goes public, the tool assembles the site on its own machine and checks it holds together with no errors. Think of it as a dress rehearsal. If a stray typo has broken something, it surfaces here, where it's trivial to fix, rather than live on the shared link halfway through a demo in front of a candidate. Once the rehearsal passes, it takes a snapshot of the finished pack and shows it to you, so a human gets to eyeball the actual thing (the salary, the interviewers, the wording) and give a yes before it goes any further. Nothing moves forward until both the build passes clean and you're happy with what you see.

Install the dependencies, then run a production build:

```
npm install --silent
npm run build
```

If the build fails, fix the issue in `pack.ts` and re-run. Do NOT commit or push until the build is green.

Then show the human what was built. Once the build is green, capture a screenshot of the finished pack from this build folder and present it for review. Call out the fields most worth a human eye — the role title, the salary band, who's on each interview stage, and the snapshot — and ask the user to confirm before continuing. If they spot something off, fix it in `pack.ts`, re-run the build, and show the updated screenshot. Do NOT commit, push, or deploy until the user has approved what they've seen.

> ⚠️ **Don't try to preview the pack with `preview_start` before deploy.** The preview tool reads its config from a different folder than the one this skill builds in (`/tmp/candidate-packs/<slug>`), so any screenshot it takes will show the *template's* old content, not the new pack. Screenshot the build folder directly instead (as above). The local `npm run build` plus your approval is the verification gate; the final live URL check happens in step 11 after deploy + protection-disable.

8. Commit and push the role-specific changes

> IN PLAIN ENGLISH
>
> Now it saves the edits with a short label and uploads them to GitHub, so the finished copy lives safely online rather than only on the machine. Because the project was set private way back in step 5, all the sensitive detail (the salary bands, the internal process notes) stays inside your own account when it goes up. Nothing is exposed by this step.

The repo already exists on GitHub from step 5 — just push the edits.

```
git add -A
git commit -q -m "Pack: <Role Title>"
git push
```

The repo was created `--private` in step 5, so the role-specific copy (salary bands, process detail) stays inside your GitHub account.

9. Link to a new Vercel project and deploy

> IN PLAIN ENGLISH
>
> This is the moment the pack actually goes live. GitHub has been storing the files; Vercel is the service that turns them into a real website with a proper web address you can send to someone. The tool connects the folder to a brand-new Vercel project, publishes it, and grabs the resulting link. After this step there's a working URL, though there's still one thing standing between it and your candidate, which the next step sorts out.

```
npx --yes vercel@latest link --yes --project <slug>
npx --yes vercel@latest deploy --prod --yes
```

Vercel CLI should already be logged into your account. If `vercel whoami` errors, run `vercel login` before continuing. Capture the live URL from the CLI output.

10. Disable deployment protection

> IN PLAIN ENGLISH
>
> Fresh sites on Vercel's Hobby plan come with a login wall in front of them by default, meaning only someone signed into your own Vercel account could open the link. A candidate clicking it would just hit a dead end. This step switches that wall off automatically, so the link simply works for anyone you send it to. It's a deliberate choice, not a security slip: the pack is built to be shared, so the only thing made public is the pack you're choosing to send out.

New Hobby projects ship with **deployment protection** ON, which blocks anyone who isn't logged into your Vercel account from viewing the website. Default behaviour for this skill: turn it off automatically. These packs are demo artefacts; the salary band is already public-by-design once the link is shared.

Read the Vercel auth token from the CLI's local config and PATCH the project to disable `ssoProtection`:

```
TOKEN=$(python3 -c "import json,sys; print(json.load(open('$HOME/Library/Application
Support/com.vercel.cli/auth.json'))['token'])")
PROJECT=$(python3 -c "import json,sys;
print(json.load(open('.vercel/project.json'))['projectId'])")
TEAM=$(python3 -c "import json,sys; print(json.load(open('.vercel/project.json'))['orgId'])")
curl -s -X PATCH "https://api.vercel.com/v9/projects/$PROJECT?teamId=$TEAM" \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"ssoProtection":null}'
```

The response body will include `"ssoProtection": null` on success. If the auth file is in a different location on a future macOS or Linux machine, also try `~/.local/share/com.vercel.cli/auth.json` or `~/.config/com.vercel.cli/auth.json`.

11. Final output check (last gate before reporting)

> IN PLAIN ENGLISH
>
> One last safety pass before it tells you the pack is ready. The tool opens the live link the way an outsider would, with no login, and checks three things: the page actually loads, the new role's details are really on it, and none of the old template placeholder text has leaked through. Doing it as an outsider is the point: a pack can't look "done" just because you happen to be signed in. If any of the three checks fail, it fixes and republishes rather than reporting a false success.

Hit the live URL with plain `curl` (no auth attached, so this simulates a candidate opening the link) and confirm the new content shipped *and* none of the old template strings leaked through.

```
URL="<full live URL from step 9>"
# 1. Public reachability (must be 200 = OK, not 401 = locked behind login)
curl -s -o /dev/null -w "%{http_code}\n" "$URL"
# 2. New content present (must show all three lines)
curl -s "$URL" | grep -oE "(<Role Title>|£<low> - £<high>|<reportsTo name>)" | sort -u
# 3. Old template content absent (must be empty)
curl -s "$URL" | grep -oE "(<Template Role Title>|<Template Salary>|<Template Manager>)"
```

What "pass" looks like:

- HTTP 200
- Check 2 lists the role title, salary band and reports-to all present
- Check 3 returns nothing — if it prints any string, the wrong content is live and you need to debug before reporting back

12. Report back

> IN PLAIN ENGLISH
>
> The handover. You get the role it pulled out, the private GitHub copy, and the live, shareable link. It also flags a short "worth a second look" list, because reading an intake involves a few judgement calls (the two-line pitch, who's on stage three, anything it inferred rather than found stated), and it would rather point those out for your eye than pretend they're gospel.

Tell the user:

- The role title that was extracted
- The new GitHub repo URL
- The live Vercel URL (now publicly shareable — protection was disabled in step 10)
- A nudge to verify any role-specific data the user might want to tweak (especially the snapshot, the stage 3 `with` field, and any judgement-call inferences from the transcript)

When extraction is ambiguous

If the transcript doesn't clearly contain a required field, ask the user ONCE before proceeding. Don't guess on:

- The compensation band (specific numbers)
- The number of interview stages (default to 4 unless transcript says otherwise)
- Who's on each interview stage

For everything else (responsibilities, prep, assessing criteria), do your best to infer from the transcript. The user ran the call — treat the note as the source of truth on what they want in the pack.

What lives where

A quick map of every file/service this skill touches so it knows exactly where to find it:

- **Skill files** (this folder, the instructions you're reading right now): `~/.claude/skills/candidate-pack/`
- **Template repo** (the shared starter, don't modify): `[YOUR_GITHUB_USERNAME]/[YOUR_TEMPLATE_REPO]` on GitHub
- **Generated packs** (one per role): cloned to `/tmp/candidate-packs/<slug>` locally, then pushed to GitHub and deployed to Vercel
- **Notion intake notes** (the source meeting notes): in your Notion workspace, search by role keyword