MAH Founders Crash Course

Git, Agents, Supabase, OpenRouter, Virtual Servers — A Handbook for Stephen, Olu, and Teddy

Melbourne AI Hub

2026-04-28

\newpage

How to use this handbook

This is the printable companion for the **Apr-28 Melbourne AI Hub crash course**. It walks two new founders (Stephen and Olu) from "I have a laptop" to "I just shipped a real change to the live website."

The chapters are designed to be read in order during the workshop, but each is also self-contained for later reference. Total reading time: about **2 hours**. Hands-on time during the workshop: **45 minutes**.

Reading map

Chapter	What it covers	When to read
00	The five accounts you need (GitHub, Anthropic, Supabase, OpenRouter, Cursor)	**Before** the workshop
01	Welcome to MAH — what we are, who is involved	First 5 min of workshop
02	Getting write access (GitHub + PR flow)	Workshop minute 5–15
03	Local setup (clone, env, run the site)	Workshop minute 15–30
04	Claude Code setup (terminal-driven AI agent)	Workshop minute 30–35
05	Cursor setup (alternative editor-driven AI)	Skip if using Claude Code
06	First task — end-to-end	Workshop minute 35–45
07	Do-not-touch — surfaces that bite	Read before first PR
08	Glossary — terms you'll see	Reference, keep open in tab
09	OpenRouter — how we talk to LLMs	After workshop, week 1
10	Virtual servers — SSH, Lightsail, where MAH runs	After workshop, week 1
11	dev.melbourneaihub.com.au — shared dev plan	Read once, ask questions

The workshop schedule

```
0:00  Welcome + quick demo of the live site
0:05  Confirm signup links (chapter 00) are done
0:10  GitHub invites + clones
0:20  Local env setup (mah doctor green)
0:30  Run mah serve — see the site on 127.0.0.1
0:35  Make a tiny copy edit
0:40  Branch + commit + push + open PR
0:45  Watch dev deploy — open the change live on dev URL
0:50  Q&A + setting up Claude Code or Cursor
1:00  Wrap — what to read after the workshop
```

If we slip on time, the safety net is: every founder leaves with a working local environment. The deploy step is a stretch goal for day one.

What is and isn't in this handbook

In: the five service accounts, getting code on GitHub, running MAH locally, using an AI agent, the first end-to-end task, OpenRouter basics, SSH/Lightsail basics, and the dev environment plan.

Out: Stripe internals, AWS SES, the newsletter pipeline, Google Workspace mirror, the Telegram bot, agent memory architecture. Those live in the wider `docs/notebooklm/` pack and should be picked up after the workshop, in whichever order serves your work.

Spirit: This is meant to demystify, not impress. If a sentence reads as gatekeeping, fix it. The whole point of MAH is that non-engineers can ship.

\newpage

ewpage

Signup Links — Do These Before the Crash Course

Each link below opens a free signup. Aim to have all of them done **before** the 45-minute setup block in the workshop. If you only get through three, prioritise the top three.

#	Service	What it does for MAH	Cost to start	Link
1	**GitHub**	Where the MAH code lives. Where you push changes.	Free	https://github.com/signup ⌘
2	**Anthropic / Claude**	Your AI pair programmer (Claude Code in the terminal, Claude on the web).	Free tier	https://console.anthropic.com/login ⌘
3	**Supabase**	Our database + auth + Edge Functions. You log in to read schema & secrets.	Free	https://supabase.com/dashboard/sign-up ⌘
4	**OpenRouter**	One API key, every LLM. We use it for the AI features in members area.	Free credit	https://openrouter.ai/sign-in ⌘
5	**Cursor (optional)**	AI-native editor if you don't want to drive from the terminal.	Free	https://cursor.com/sign-up ⌘
6	**Stripe (testing)**	Read-only access to the Stripe dashboard so you can see test payments.	Free	Ask Teddy to invite your email to the MAH org
7	**AWS (read-only)**	Lightsail / SES dashboards. Optional — only if you want to see prod logs.	Free	Ask Teddy to invite you as IAM user

What to do on each, in order

1. GitHub — 5 minutes

1. Sign up at the link above with the email you want as your dev identity.
2. Choose a **handle** you don't mind appearing in `git log` forever (e.g., `olu-mah` or `stephen-mah`).
3. After signup, send your handle to Teddy in Telegram. He'll add you as a `Maintain` collaborator on `teddyscleaningserviceaus-design/melbourne-ai-hub`.
4. You'll get an email invitation — accept it.

That's all. The actual SSH-key + clone steps happen in chapter 02 of this book.

2. Anthropic — 5 minutes

1. Sign up at https://console.anthropic.com/login ⌘.
2. Verify your email.
3. The free tier gives you enough usage to install and run Claude Code (`npm install -g @anthropic-ai/claude-code`) and test it out. We'll use it during the workshop.
4. **Don't add a credit card yet.** We'll talk about which plan makes sense for you at the end of the workshop.

3. Supabase — 5 minutes

1. Sign up at https://supabase.com/dashboard/sign-up ⌘ with the same email you used for GitHub if possible (makes future SSO easier).
2. After signup, send Teddy your Supabase email. He'll add you as a member to the `pvugmifdnanffuessqhf` project (Melbourne AI Hub).
3. You'll be able to see: tables, the SQL editor, Edge Function logs, the Auth → Users list.
4. **Read access only at first.** Write/delete needs co-founder approval (book-07/07-do-not-touch.md explains why).

4. OpenRouter — 5 minutes

1. Sign up at https://openrouter.ai/sign-in ⌘. GitHub SSO is fine.
2. Go to **Keys** → **Create Key** → name it `mah-personal-<yourname>`.
3. Copy the key (`sk-or-v1-...`). Save it in 1Password / your password manager.
4. **Don't paste it into the MAH `.env` yet.** You use this key for your own personal AI experiments. The MAH production key is separate and lives in Supabase secrets — see chapter 09.

OpenRouter gives you ~$1 of free credit on signup, which is enough to test a few hundred small calls. You can add $5 by card later if you want to play with frontier models.

5. Cursor — 5 minutes (skip if using Claude Code)

1. Sign up at https://cursor.com/sign-up ⌘.
2. Download the editor for your OS.
3. The free tier covers Cursor Tab + a few hundred AI completions per month. We'll switch you to the paid tier ($20/mo) if you find you use it daily — not before.

If you're going to drive from the terminal with Claude Code, **skip Cursor entirely**. Don't run both — you'll confuse yourself.

6. Stripe (read-only) — Teddy invites you

1. Send Teddy the email you want on the Stripe team.

2. He invites you with **read-only** access to the MAH Stripe org.
3. You'll be able to see: payments, customers, products, webhook events, logs.
4. You **cannot** issue refunds, change prices, or rotate keys without explicit elevation. That's intentional. This week's $1 test prices will be visible there.

7. AWS (optional) — Teddy invites you

Skip on day 1 unless you specifically want to inspect SES email logs or Lightsail metrics. We'll come back to this once you've shipped your first PR.

Quick checklist before the workshop

- [ ] GitHub account created, handle sent to Teddy
- [ ] Anthropic console account active
- [ ] Supabase account active, Teddy added you to the MAH project
- [ ] OpenRouter key generated, saved in password manager
- [ ] Editor of choice installed (Cursor or Claude Code)

If two or more of those are blocked on Teddy, that's fine — the workshop opens with him doing the inviting in real time.

Why this many accounts?

MAH is built on small composable services rather than a single platform. Each account is a separate trust boundary:

- GitHub stores the source code.
- Supabase stores the data and runs the backend.
- OpenRouter routes the AI calls.
- Stripe handles money.
- AWS handles email and the public web server.
- Anthropic / Cursor are your editor tools.

That separation is by design. If one of those vendors disappears tomorrow, the others keep working. No single sign-on, no lock-in. The cost is more accounts to manage. The benefit is that we can swap any one of them for a competitor in a weekend if we need to.

Welcome aboard. Once these accounts are set up, jump to `01-welcome.md` and read in order.

ewpage

Welcome to Melbourne AI Hub

Hi. If you're reading this, somebody trusts you with the keys. This page exists so a friendly human who has never touched a line of MAH code can finish the afternoon having safely shipped a real change. Bring your AI agent (Claude Code, Cursor, ChatGPT, whatever you like) — this whole pack is written for the two of you working together.

What MAH is, in plain English

Melbourne AI Hub (MAH) is a small, ambitious community for Melbourne founders who want to build with AI. We run paid workshops, host a private member portal, send a weekly AI-curated newsletter, and use AI agents to help our members ship projects. The product is a website (`https://melbourneaihub.com.au`), a member portal behind a magic-link login, and a set of background services that handle payments, emails, content scraping, and Google Workspace mirroring.

We are deliberately small: a handful of founders, a few dozen members, growing carefully. We chose a stack — vanilla HTML + Supabase + AWS SES — that a non-engineer founder can read, change, and deploy with help from an AI agent. You don't need a CS degree to ship here. You need to read carefully, run the commands you're given, and tell your agent the truth about what you don't understand.

Who is involved

- **Founders.** A small group with write access. You are one of them. You can ship code, send emails, approve members.
- **Members.** Paying community members who log into `members.html`. They see tabs for projects, events, learning tracks, the directory, DMs, and tools.
- **Workshop attendees.** Anyone who pays to attend a workshop via `workshop.html`. They get an emailed confirmation; some convert to members.
- **Agents.** Claude Code, Cursor, and friends. They read the docs you're reading right now, help you make changes, and write decisions back to memory so the next agent picks up where the last one left off.

Why we use NotebookLM + agents

A non-engineer founder cannot hold the entire MAH codebase in their head. We don't expect you to. Instead:

- This documentation pack (`docs/notebooklm/`) is the canonical narrative — it lives next to the code so it stays honest.

- Every file is also ingested by NotebookLM in our Google Workspace, so you can ask grounded questions in plain English (for example, "How does the workshop signup flow work?") and get cited answers.
- An AI coding agent reads `/CLAUDE.md` (or `/AGENTS.md`) on startup, which points it at this pack, which tells it what is safe to change and what isn't.

You bring product judgement. The agent brings code fluency. The pack keeps both of you honest.

Source-of-truth pointers

Read these in roughly this order as you go:

1. `docs/notebooklm/00-index.md` — the index for the whole pack, including a precedence list for when documents disagree.
2. `MAH-MASTER-ARCHITECTURE.md` (repo root) — older but useful narrative architecture document. Use it as backup context.
3. `STATUS-MAH.md` (repo root) — last-known production state snapshot.
4. `CONCEPT.md` (repo root) — the founding concept and product vision.
5. `VISION-2045.md` (if present in repo or vault) — the long-arc vision.

When two documents disagree, the precedence order in `00-index.md` wins. When code disagrees with documentation, **the code wins** and the documentation needs updating.

What this book promises

By the end of book-07, you will be able to:

- Get added as a GitHub collaborator and clone the repo.
- Run MAH locally on your laptop.
- Set up Claude Code (or Cursor) so your agent has the right context.
- Walk through a real first task end-to-end (frontend + database + Edge Function + deploy).
- Know which files to never touch without a second pair of eyes.
- Understand the jargon that appears in commits, comments, and Slack/Telegram chatter.

You don't need to read all of book-07 in one sitting. The "First Task Checklist" (`06-first-task-checklist.md`) is the moment things click.

Reading order from here

1. `02-getting-write-access.md` — get added on GitHub, set up `gh`, understand the PR flow.
2. `03-local-setup.md` — clone the repo, install dependencies, run `mah serve`.
3. `04-claude-code-setup.md` OR `05-cursor-setup.md` — pick the agent you want to drive with.

4. `06-first-task-checklist.md` — your first real change, end-to-end.
5. `07-do-not-touch.md` — the high-blast-radius surfaces. Skim this before your first PR.
6. `08-glossary.md` — keep this open in a tab while you read everything else.

After book-07, you'll usually go to:

- `book-01-architecture/01-system-overview.md` to see the whole topology.
- The book matching whatever you're working on (frontend, backend, integrations, content, ops).
- `book-08-memory-and-decisions/06-how-agents-should-write-memory.md` to understand the memory system you're contributing to.

A note on tone

This pack tries to be honest about what is and isn't built. If a file says `status: PARTIAL` or `status: PLANNED`, believe it. We'd rather tell you something is half-finished than pretend it works and waste your afternoon. If you find a doc lying, fix it — that's a perfectly good first PR.

Welcome aboard.

ewpage

Getting Write Access

This page walks you from "I have a GitHub account" to "I have an open PR on `melbourne-ai-hub`". Total time: ~20 minutes if your SSH key is already set up; ~40 minutes if it isn't.

1. Get added as a collaborator

Ask the lead founder (currently Kelebet / Teddy) to add your GitHub username to:

- **Repo:** `teddyscleaningserviceaus-design/melbourne-ai-hub`
- **Role:** `Maintain` (can push to non-protected branches, approve PRs, but cannot delete the repo)

You'll receive a GitHub email invitation. Accept it. Until you accept, `git clone` of a private branch will fail with `remote: Repository not found`.

2. Authenticate locally

You need GitHub to recognise you when you `git push`. Two options — pick one. SSH is recommended.

Option A — SSH key (recommended)

Check whether you already have one:

```
ls ~/.ssh/id_ed25519.pub ~/.ssh/id_rsa.pub 2>/dev/null
```

If neither file exists, generate one:

```
ssh-keygen -t ed25519 -C "your-email@example.com"
# Press Enter through the prompts. A passphrase is optional but encouraged.
```

Copy the public key to your clipboard:

```
cat ~/.ssh/id_ed25519.pub | pbcopy   # macOS
# or: cat ~/.ssh/id_ed25519.pub | xclip -selection clipboard   # Linux
```

Paste it into GitHub at https://github.com/settings/keys ⌘ → "New SSH key".

Test the connection:

```
ssh -T git@github.com
# Expected: "Hi <username>! You've successfully authenticated..."
```

Option B — GitHub CLI

```
brew install gh        # macOS (or use your distro's package manager)
gh auth login
# Choose: GitHub.com → HTTPS or SSH → "Login with a web browser"
```

`gh` is also useful for opening PRs from the terminal — install it even if you went with SSH.

3. Clone the repo

```
mkdir -p ~/clawd/projects && cd ~/clawd/projects
git clone git@github.com:teddyscleaningserviceaus-design/melbourne-ai-hub.git
cd melbourne-ai-hub
```

If you cloned via HTTPS instead, the URL is `https://github.com/teddyscleaningserviceaus-design/melbourne-ai-hub.git`.

4. Branch protection — what's enforced

The `main` branch is protected. You cannot `git push origin main` directly. The expected flow is:

1. Create a feature branch: `git checkout -b feat/<short-name>`
2. Commit your changes there.
3. Push the branch: `git push -u origin feat/<short-name>`
4. Open a PR against `main`.
5. Get a review from another founder.
6. Merge.

`main` is what gets deployed. If a bad commit lands on `main`, it ships to production the next time someone runs `release.sh`.

5. PR review flow

Every PR needs **one founder approval** before merging. The reviewer is checking:

- Does the change match the stated intent in the PR description?
- Does it touch any "do-not-touch" surfaces (see `07-do-not-touch.md`)? If so, was a security review done?
- Are migrations idempotent and RLS-enabled? (see `.claude/rules/migrations.md`)

- Does the commit message make sense in `git log` six months from now?

If your PR sits for more than 24 hours without review, ping in Telegram. Founders are often deep in operator work and miss notifications.

6. What needs a second pair of eyes

Treat these surfaces as "always pair with another founder before merging":

Surface	Why
`supabase/functions/stripe-webhook/`	Signature verification is sacred — getting it wrong loses payments.
`supabase/functions/_shared/auth.ts`	Magic-link generation. Breaking it locks members out.
`supabase/functions/_shared/ses.ts`	AWS SigV4 signing. Subtle. Breakage = silently failed emails.
`backend/migrations/*.sql`	Schema changes are forever. RLS mistakes leak member data.
`release.sh`	The deploy path itself — break it, break every future deploy.
`.claude/rules/`	Changes the contract every agent operates under.
Anything touching Stripe `sk_live_` or AWS production keys	Money + email reputation.

7. What you can ship solo

These are normal-sized changes that don't need a co-founder review (though one is always nice):

- Copy and content edits in `*.html` files
- Visual / CSS tweaks in `styles.css`
- Adding a new content source to the newsletter pipeline
- Adding a new doc file under `docs/notebooklm/`
- Fixing typos in existing docs
- Bumping `last-verified` dates after re-checking a doc

If you're not sure whether your change is solo-shippable, ask. Defaulting to "request a review" is never wrong.

8. Commit message style

We follow the conventional-commits convention:

```
<type>: <short imperative description>
```

Allowed types:

- `feat:` — new user-visible feature
- `fix:` — bug fix
- `docs:` — documentation only
- `chore:` — tooling, deps, config
- `refactor:` — code change that doesn't change behaviour
- `style:` — formatting only
- `test:` — adding/updating tests
- `perf:` — performance improvement
- `ci:` — CI/CD config

Examples from real MAH commits:

```
feat: add transparency-log section to transparency.html
fix: stripe-webhook double-handles refund events
docs: update book-03 with onboard-member rate-limit note
chore: bump supabase deno version to 1.41
```

Keep the description under 72 characters. If you need more, add a body after a blank line.

9. No CI today (acknowledged gap)

There is currently **no continuous-integration pipeline** on this repo. No automatic tests run when you open a PR. No linter blocks bad commits. This is flagged as a gap in `book-09-roadmap-and-gaps/`.

Until CI exists, your safety net is:

1. Manual local testing — `mah serve`, `supabase functions serve <name>`, `mah doctor`.
2. The reviewer checking the diff carefully.
3. The pre-deploy checklist in `book-06-operations/01-deploy-runbook.md`.
4. Production smoke tests after `release.sh`.

Be the rigorous one. The repo is small enough that careful manual verification works — but only if you actually do it.

10. Quick reference

```
# Daily flow
git checkout main
git pull
git checkout -b feat/my-change
# ... edit files ...
git add <specific-files>
git commit -m "feat: my change"
git push -u origin feat/my-change
gh pr create --base main --web    # opens browser to fill out PR
```

After merge, delete the feature branch (`git branch -d feat/my-change`) and pull `main` again before starting the next change.

ewpage

Local Setup

Goal: clone the repo, install dependencies, and load `http://127.0.0.1:8787` in your browser. Allow about 30 minutes the first time.

Pre-requisites

You need the following installed before starting. Check each with the listed command — if it errors, install it.

Tool	Version	Check	Install hint (macOS)
`git`	2.30+	`git --version`	`brew install git` (or already on macOS)
Python	3.11+	`python3 --version`	`brew install python@3.11`
Node.js	20+ (LTS)	`node --version`	`brew install node`
Supabase CLI	1.180+	`supabase --version`	`brew install supabase/tap/supabase`
Deno	1.40+	`deno --version`	`brew install deno`
`gh`	any	`gh --version`	`brew install gh`
`jq`	any	`jq --version`	`brew install jq`
Ollama (opt.)	latest	`ollama --version`	https://ollama.com/download ⌘

Linux: use your distro's package manager (`apt install`, `dnf install`, etc.). The Supabase CLI install instructions are at https://supabase.com/docs/guides/cli/getting-started ⌘.

Ollama is optional. The newsletter pipeline can use either Zhipu's GLM API (preferred, lives in `.env`) or a local Ollama model. If you skip Ollama, set the right env vars and you're fine.

1. Clone the repo

```
mkdir -p ~/clawd/projects && cd ~/clawd/projects
git clone git@github.com:teddyscleaningserviceaus-design/melbourne-ai-hub.git
cd melbourne-ai-hub
```

If `git clone` fails with `Repository not found`, you haven't accepted the GitHub invite yet — go back to `02-getting-write-access.md` step 1.

2. Set up your `.env`

```
cp .env.example .env
```

`.env` is gitignored. `.env.example` is the canonical list of keys. Open `.env` in your editor and fill in the values.

You'll need:

- **Supabase keys** — anon key, service role key, project ref `pvugmifdnanffuessqhf`. Ask another founder for these.
- **Stripe keys** — use the **TEST** keys for local dev (`sk_test_…`, `pk_test_…`). The LIVE keys are only for production.
- **AWS SES keys** — for sending email. Ask for the SES IAM user credentials.
- **GLM API key** — for the newsletter pipeline. From the Zhipu dashboard.
- **Google Workspace service account** — JSON file path. Usually `~/.config/mah/gws-sa.json`.
- `MAH_TEST_MODE_RECIPIENT` — your own email. **Set this.** It redirects all SES sends to you in non-production runs, so you don't accidentally email a real member.

For the full key list with descriptions, read `book-06-operations/04-environment-variables.md`. For "where do I get this secret?", read `book-06-operations/05-secrets-management.md`.

> **Important.** There is **no separate dev Supabase project** today. Local development hits the **same** database as production. This is a known gap (see book-09). In the meantime:
>
> - Use Stripe TEST keys, not LIVE keys.
> - Set `MAH_TEST_MODE_RECIPIENT` to your own email so SES doesn't go to real members.
> - Be very careful with `INSERT` / `UPDATE` / `DELETE` queries against `mah_*` tables.
> - Prefer reading; write only to clearly named test rows you can identify and clean up later.

3. Install Python dependencies

We use a virtualenv so MAH's deps don't pollute your system Python.

```
python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip
pip install -e .
```

`-e .` installs the `mah` Python package from `pyproject.toml` in editable mode — changes to `mah/` source take effect immediately without reinstalling.

You'll know it worked when:

```
mah --help
```

prints the CLI usage with subcommands like `serve`, `doctor`, `newsletter`, `email`, `gws`, `research`.

4. Verify your env with `mah doctor`

```
mah doctor
```

`doctor` checks:

- All required env vars are set (`SUPABASE_URL`, `SUPABASE_SERVICE_ROLE_KEY`, `AWS_ACCESS_KEY_ID`, `STRIPE_SECRET_KEY`, etc.)
- Supabase is reachable
- AWS SES credentials are valid
- The Google Workspace service account JSON exists and is readable
- The local Ollama daemon is running (warning only, not failure)

If something's missing, `doctor` will tell you which env var. Go fill it in and re-run.

5. Run the local site

```
mah serve
```

This starts a static file server on `http://127.0.0.1:8787` that serves the repo root. Open it in your browser. You should see the MAH landing page.

Things to try:

- `http://127.0.0.1:8787/` — landing page
- `http://127.0.0.1:8787/workshop.html` — workshop signup
- `http://127.0.0.1:8787/transparency.html` — transparency page
- `http://127.0.0.1:8787/members.html` — member portal (requires login)

Edits to HTML, JS, and CSS are picked up on browser refresh — no rebuild step. That's by design.

6. (Optional) Run an Edge Function locally

If you're going to touch a Supabase Edge Function, you need to run it locally too.

```
supabase login                              # one-time, opens browser
supabase link --project-ref pvugmifdnanffuessqhf
supabase functions serve register-workshop  # or any function name
```

This serves the named function on `http://localhost:54321/functions/v1/<name>`. Curl it from another terminal:

```
curl -X POST http://localhost:54321/functions/v1/register-workshop \
  -H "Content-Type: application/json" \
  -d '{"email":"you@example.com","session":"morning"}'
```

The function reads secrets from your local `.env` plus a special `supabase/.env` file if present. See `book-06-operations/02-edge-function-deploy.md` for the full local-EF workflow.

7. Stop everything cleanly

- Static server: `Ctrl-C` in the `mah serve` terminal.
- Edge Function: `Ctrl-C` in that terminal.
- Virtualenv: `deactivate` (returns to your normal shell Python).

Troubleshooting

`mah: command not found` Your virtualenv isn't activated. Run `source .venv/bin/activate`.

`mah doctor` says `OLLAMA_NOT_RUNNING` Either start Ollama (`ollama serve`) or ignore the warning if you don't plan to use local LLMs. The newsletter pipeline can use the GLM API instead.

`Address already in use` on port 8787 Another `mah serve` is still running, or another process grabbed the port. Find it: `lsof -i :8787`. Kill it: `kill <pid>`.

`supabase login` opens browser but never returns Sometimes the callback hangs. `Ctrl-C` and try again. If the second attempt also fails, run `supabase login --token <your-personal-token>` instead.

`pip install -e .` fails with build errors Probably a missing system header. On macOS, run `xcode-select --install`. On Linux, install `build-essential` and `python3-dev`.

Supabase queries return `permission denied for table mah_*` RLS is doing its job — anon key cannot read protected tables. Use the service-role key in scripts (carefully) or query via an Edge Function. Never expose the service-role key to the browser.

Stripe webhook signature errors locally Use the Stripe CLI to forward events: `stripe listen --forward-to http://localhost:54321/functions/v1/stripe-webhook`. Use the signing secret it prints, not the production webhook secret.

`gws-proxy` **returns "service account not found"** Check `GWS_SERVICE_ACCOUNT_PATH` in `.env`. The JSON file must exist and be readable. Never commit it.

What you have now

- A local site running at `127.0.0.1:8787`.
- A working `mah` CLI in your virtualenv.
- A `.env` filled out so future agents and CLI commands work.
- Optional ability to run individual Edge Functions locally.

Next: pick your editor + agent setup. Claude Code is in `04-claude-code-setup.md`, Cursor is in `05-cursor-setup.md`. Then go straight to `06-first-task-checklist.md`.

ewpage

Claude Code Setup

Claude Code is Anthropic's official CLI agent — it lives in your terminal, reads files, edits files, runs commands, and asks for confirmation when needed. This is the agent setup we recommend for non-engineer founders because it auto-loads `CLAUDE.md` from the repo root, which already points at this whole documentation pack.

If you prefer Cursor or another AI editor, see `05-cursor-setup.md` instead.

1. Install

```
npm install -g @anthropic-ai/claude-code
```

Verify the install:

```
claude --version
```

If `claude: command not found`, your global npm bin directory isn't on `PATH`. Run `npm config get prefix` to find it (usually `/usr/local` on macOS) and ensure its `bin/` is on your `PATH`.

You'll be prompted to authenticate the first time you run `claude`. Follow the browser prompt to log in to your Anthropic account.

2. Run from the repo root

From any terminal, change into the repo and run `claude`:

```
cd ~/clawd/projects/melbourne-ai-hub
claude
```

Claude Code automatically:

- Reads `/CLAUDE.md` (the repo's agent contract)
- Reads `/AGENTS.md` (alternate file for tool-agnostic agents)
- Loads any `.claude/rules/*.md` files relevant to the area you touch
- Knows the documentation pack lives in `docs/notebooklm/`

That's it. You don't need to copy-paste anything to give the agent context. Just describe what you want.

3. What `CLAUDE.md` does for you

Open `/CLAUDE.md` once, just to see. It establishes:

- The reading order before any edit (`docs/notebooklm/00-index.md` first)
- The hard rules (`release.sh` is the only deploy path, RLS on every table, etc.)
- The critical paths in the repo (where to look for which feature)
- The per-domain rule files in `.claude/rules/` (frontend, edge-functions, migrations, deploy)
- Where to write decisions and learnings after a change

Every Claude Code session starts with this loaded. If you ever feel the agent has gone off-track, ask it: "What does CLAUDE.md say about this?" — it should pull the relevant rule.

4. Recommended `.claude/settings.local.json`

Claude Code asks before running commands or editing files outside its allow-list. You can pre-approve common safe actions to reduce friction. Create or edit `.claude/settings.local.json` (gitignored — local to your machine):

```
{
  "permissions": {
    "allow": [
      "Bash(git status)",
      "Bash(git diff*)",
      "Bash(git log*)",
      "Bash(git add*)",
      "Bash(git commit*)",
      "Bash(git push origin feat/*)",
      "Bash(npm install*)",
      "Bash(npm run*)",
      "Bash(mah doctor)",
      "Bash(mah serve)",
      "Bash(mah newsletter*)",
      "Bash(mah email*)",
      "Bash(supabase functions serve*)",
      "Bash(curl http://127.0.0.1:*)",
      "Bash(curl http://localhost:*)",
      "Read(./**)",
      "Edit(./**)",
      "Write(./**)"
    ],
    "deny": [
      "Edit(./.env)",
      "Write(./.env)",
      "Bash(./release.sh*)",
      "Bash(supabase functions deploy*)",
      "Bash(psql*production*)",
      "Bash(stripe*--live*)",
      "Bash(rm -rf*)"
    ]
  }
}
```

Why this shape:

- **Allow** routine reads, edits, and local servers — these are safe.
- **Deny** the deploy script, EF deploy, and direct production database access — those should always involve a confirmation step.
- **Deny** edits to `.env` — secrets editing is always interactive.

Tweak to taste. The defaults err on the side of asking too often; loosening as you trust the agent more is fine. **Never** add `Bash(*)` or any wildcard that includes destructive operations.

5. Pointing the agent at a specific book

Claude Code respects file paths in your prompt. Two effective patterns:

Direct file mention. If you know which doc covers the task, say so:

> "Read `docs/notebooklm/book-02-frontend/07-adding-a-page.md` and walk me through adding a partners page."

Topic-based. Let the agent navigate the index:

> "I want to add a new member-portal tab. Read `docs/notebooklm/00-index.md`, find the right book, and tell me what you'll do before you start."

Both work. The second is friendlier for first-timers because the agent shows its reasoning.

6. Example session — adding a partners page

```
> I want to add a /partners.html page listing our partner orgs.
  Read the docs first, then propose a plan.

[Claude reads docs/notebooklm/00-index.md, then book-02-frontend/07-adding-a-page.md]

I've read book-02-frontend/07-adding-a-page.md. The procedure is:

1. Create partners.html at repo root, copying structure from team.html.
2. Add a section listing partners with logo, name, blurb.
3. Add a link from index.html and js/site-header.js.
4. Add to sitemap.xml.
5. Verify locally with `mah serve` → http://127.0.0.1:8787/partners.html.
6. Deploy via `release.sh`.

Shall I start by copying team.html to partners.html and showing you the diff
before any edits?

> yes, but don't deploy. Just get it ready locally.

[Claude makes the edits, runs `mah serve`, gives you the URL to check]
```

The agent should always pause before deploying. If it tries to run `release.sh` without explicit confirmation, stop it.

7. Slash commands worth knowing

Inside a Claude Code session, type `/` to see the menu. The ones you'll use most:

Command	What it does
`/plan`	Switch to plan mode — agent proposes steps, doesn't execute until you approve.
`/review`	Ask for a code review of recent changes.
`/explore`	Map the codebase or a specific subsystem before changing it.
`/explain-error`	Paste an error and get an explanation + likely fix.
`/clear`	Clear the conversation context (start fresh).
`/compact`	Summarise the long chat into a shorter context (when you've been at it for hours).
`/exit`	End the session.

Use `/plan` for anything non-trivial. It's the easiest way to catch a misunderstanding before code changes ship.

8. Memory writing — the agent's job

After every non-trivial change, the agent should append to:

- `.claude/agent-memory/founders/decisions.md` — what changed and why
- `.claude/agent-memory/founders/learnings.md` — anything surprising
- `.claude/agent-memory/founders/integration-status.md` — re-tick the affected integration row

`/CLAUDE.md` already instructs the agent to do this. If it forgets, prompt it: "Don't forget to write the decision to `decisions.md`."

The format is documented in `book-08-memory-and-decisions/06-how-agents-should-write-memory.md`. The short version: timestamp, what changed, why, what the founder should know next time.

9. Hooking into the global memory system

Kelebet's global Claude config (`~/.claude/CLAUDE.md`) sets up Obsidian session logging and a separate `~/clawd/memory/` daily log. If you're working on Kelebet's machine, those hooks fire automatically. If you're on your own machine and want similar habits, the simplest version:

1. Create a `~/clawd/memory/` directory.
2. At the end of each session, ask the agent: "Write a short session log to `~/clawd/memory/$(date +%Y-%m-%d).md` with what we did, decisions, and next steps."
3. Review what it wrote before quitting.

This habit pays for itself the first time you come back two weeks later and need to remember why a thing is the way it is.

10. When the agent is wrong

It will be, sometimes. Symptoms:

- Confidently invents a file path that doesn't exist
- Skips reading the docs and jumps straight to code
- Tries to use `npm:` packages in an Edge Function (forbidden — see `.claude/rules/edge-functions.md`)
- Tries to deploy without local verification

Counter-moves:

- "Read `<exact path>` first and quote the relevant section back to me."
- "Stop. Switch to `/plan` mode."
- "What does `.claude/rules/edge-functions.md` say about npm packages?"
- "Show me the local test result before we deploy."

You are the brake. The agent is the engine. That division of labour is the whole point.

11. Next

You're ready to do real work. Open `06-first-task-checklist.md` and walk through your first end-to-end task with your agent.

ewpage

Cursor (and other AI editors)

Cursor is a VS Code fork with first-class AI integration. Many founders prefer it because it has a familiar editor UI plus a side-panel agent. This page also covers other AI editors that support a "rules" file (Windsurf, Continue, Aider in editor mode, etc.) — the principle is the same: point the editor at our existing rules so it operates within our contract.

If you'd rather drive from the terminal with Anthropic's CLI, see `04-claude-code-setup.md`.

1. Install Cursor

Download from https://cursor.sh ⌘ and install. Sign in with the account that has Cursor Pro access (or use the free tier — most onboarding tasks fit fine).

Open the MAH repo:

```
cursor ~/clawd/projects/melbourne-ai-hub
```

Or use the GUI: `File → Open Folder...` → select the repo.

2. The `.cursorrules` file (or `.cursor/rules/`)

Cursor reads `.cursorrules` (single file at repo root) **or** any `.md` files in `.cursor/rules/` (newer format) and loads them as system context for every chat in this project. We use this to point Cursor at our existing agent contract instead of duplicating it.

Recommended single-file `.cursorrules` (drop this at repo root if it doesn't exist; do not commit if it conflicts with the team's choice — check first):

```
# Cursor Rules — Melbourne AI Hub

You are working in the canonical MAH repository.

## Read these before any edit

1. `/CLAUDE.md` — the agent contract (hard rules, critical paths, per-domain rules)
2. `/AGENTS.md` — tool-agnostic version of the same contract
3. `docs/notebooklm/00-index.md` — documentation pack index
4. The book in `docs/notebooklm/` matching the area you're touching

## Per-domain rules

When the change touches:

- Frontend (`*.html`, `js/*`, `styles.css`) → `.claude/rules/frontend.md`
- Edge Function (`supabase/functions/*`) → `.claude/rules/edge-functions.md`
- Migration (`backend/migrations/*.sql`) → `.claude/rules/migrations.md`
- Deploy (`scripts/release.sh`, `scripts/deploy-edge-functions.sh`) → `.claude/rules/
  deploy.md`

## Hard rules

1. `release.sh` is the only deploy path. No raw rsync/scp.
2. RLS on every table. Use `(SELECT auth.uid())`, not `auth.uid()`.
3. Edge Functions must use `_shared/` utilities — never reimplement CORS, SES SigV4,
   magic links.
4. Stripe webhook signature verification is sacred — never weaken.
5. No emoji as icons. Custom SVG or Lucide system-chrome only.
6. Secrets never in repo. `.env` is gitignored.
7. 600-line file limit. Target 200-400.
8. Confirm before external actions (Stripe live, SES broadcasts, Drive uploads, deploys).

## Memory

After any non-trivial change, append to:

- `.claude/agent-memory/founders/decisions.md`
- `.claude/agent-memory/founders/learnings.md`
- `.claude/agent-memory/founders/integration-status.md`

## Verification

Don't claim "done" without local verification:

- Static page: `mah serve` → load on 127.0.0.1:8787 → visual check
- Edge Function: `supabase functions serve <name>` → curl → verify response
- Migration: apply on a branch, query tables, verify RLS

If you cannot verify, say so explicitly.
```

If the team has standardised on the newer `.cursor/rules/` directory, mirror the same content into `.cursor/rules/mah.md`. Both formats work — pick whichever the rest of the team uses.

3. Cursor's context window — how to hint it

Cursor's chat panel has a context window that includes the open file, recently viewed files, and anything you `@`-mention. This matters because the doc pack is large and Cursor won't always pull in the right book on its own.

Effective patterns:

- `@docs/notebooklm/book-02-frontend/07-adding-a-page.md` — explicitly attach a doc to your prompt.
- `@CLAUDE.md` — pull in the contract for a tricky question.
- `@.claude/rules/edge-functions.md` — for any EF work.
- `@supabase/functions/_shared/ses.ts` — when the change touches SES.

If you're not sure which book to attach, ask: "Read `docs/notebooklm/00-index.md` and tell me which book covers `<task>` before you start."

4. Per-domain rules apply equally

Cursor doesn't auto-discover the per-domain rules in `.claude/rules/` the way Claude Code does. The `.cursorrules` file above tells Cursor to look there — but you may need to nudge it manually with an `@`-mention when starting a frontend / EF / migration task.

A good habit at the start of a session:

> "I'm about to add an Edge Function for `<thing>`. Read `@.claude/rules/edge-functions.md` and `@docs/notebooklm/book-03-backend/01-edge-functions-index.md` before proposing any code."

This costs you 30 seconds and saves you from agent-invented patterns that violate our contract.

5. Memory — write decisions back

Cursor doesn't have an opinion about where to write session memory, so we make it explicit. Tell the agent at the end of each task:

> "Append a decision entry to `.claude/agent-memory/founders/decisions.md` describing what we changed, why, and any known follow-ups. Use the format already in that file."

The format is documented in `book-08-memory-and-decisions/06-how-agents-should-write-memory.md`. If the file is empty, the format is roughly:

```
## 2026-04-28 — Added partners.html page

**Why:** Onboarding partner orgs needed a public listing.

**What changed:**

- Added `partners.html` (copied from `team.html`)
- Linked from `index.html` and `js/site-header.js`
- Added to `sitemap.xml`

**Verification:**

- `mah serve` → visual check on `127.0.0.1:8787/partners.html`
- All 6 partner cards render
- Mobile breakpoint OK at 375px

**Follow-ups:**

- Logos are placeholder SVGs — replace with vetted partner-supplied artwork.

**Agent:** Cursor (claude-3.5-sonnet)
**Founder:** <your-name>
```

This pattern works in any AI editor. The folder is `.claude/agent-memory/founders/` regardless of which tool you're using.

6. Other AI editors — same pattern

The contract scales. For any AI editor that supports a rules file:

Editor	Rules file path	Notes
Cursor	`.cursorrules` or `.cursor/rules/*.md`	Auto-loaded
Windsurf	`.windsurfrules`	Same shape as Cursor
Continue	`.continue/config.json` → `systemMessage`	Embed the rule text in the system message
Aider	`.aider.conf.yml` → `read` + `auto-commit`	Add `CLAUDE.md` and rules files to `read` so they're in context
Codex CLI	Reads `AGENTS.md` automatically	Same content as `CLAUDE.md`

In all cases, the rule body should redirect to **`/CLAUDE.md`**, **`/AGENTS.md`**, **`docs/notebooklm/00-index.md`**, and **`.claude/rules/`** rather than duplicating the rules. One source of truth.

7. Inline edits vs chat

Cursor's `Cmd-K` (inline edit) is great for quick small changes — renaming, adding a CSS rule, fixing a typo. The chat panel (`Cmd-L`) is better for anything that touches multiple files or requires reading docs first.

For MAH work, default to chat. The doc pack rewards being read.

8. When the agent is wrong

Same patterns as Claude Code:

- "Read `<exact path>` and quote the relevant section back to me."
- "What does `.claude/rules/<area>.md` say about this?"
- "Show me the local test result before we deploy."
- "Don't run `release.sh` without my explicit confirmation."

Cursor will sometimes try to use `npm:` packages in an Edge Function (forbidden), generate inline `<script>` blocks (CSP-blocked), or write `auth.uid()` instead of `(SELECT auth.uid())` in SQL. Catch these in review. The rules in `.claude/rules/` document each one.

9. Next

Open `06-first-task-checklist.md` and walk through your first end-to-end task with Cursor.

ewpage

Your First Task — End-to-End

This walkthrough exercises every layer of the MAH stack: HTML, database (with RLS), Edge Function, deploy script, and memory write. Time estimate: about **2 hours** with an agent helping. By the end you will have shipped a real, useful change to production.

Goal

Add a "Last updated" section to `transparency.html` that pulls the most recent transparency-log entry from a new Supabase table called `mah_transparency_log`.

The page already exists. We're adding:

1. A new database table to hold log entries
2. A new Edge Function to read the latest entry (anon-readable; no auth needed)
3. A new HTML section + a small JS fetch on `transparency.html`

Why this task

It's small enough to finish in an afternoon. It touches the database, an Edge Function, and the frontend, so you'll see how the layers fit together. And it's genuinely useful — visible transparency reinforces our positioning.

Prerequisites

- Local setup is working (`mah serve` loads `127.0.0.1:8787` — see `03-local-setup.md`)
- Your agent is configured (`04-claude-code-setup.md` or `05-cursor-setup.md`)
- You have a feature branch checked out: `git checkout -b feat/transparency-log`

Step 0 — Read the relevant docs

Before touching code, read these (or have your agent read them and summarise):

- `docs/notebooklm/book-02-frontend/07-adding-a-page.md` — frontend patterns
- `docs/notebooklm/book-03-backend/05-migrations-history.md` — migration conventions
- `docs/notebooklm/book-03-backend/04-rls-policies.md` — RLS patterns
- `docs/notebooklm/book-03-backend/01-edge-functions-index.md` — EF index
- `.claude/rules/migrations.md` — the rule file for migrations
- `.claude/rules/edge-functions.md` — the rule file for EFs

Time spent reading is never wasted on a first task. Twenty minutes here saves an hour of confused refactoring later.

Step 1 — Write the migration

Create `backend/migrations/037-add-transparency-log.sql`. Increment from whatever the current highest number is — check `ls backend/migrations/ | sort | tail -5` first. (At time of writing, the migrations have grown past `037`, so use the next free number you see in the directory.)

Template:

```sql
-- 037-add-transparency-log.sql
-- Adds a public-read transparency log for the /transparency.html page.
-- Rollback: DROP TABLE public.mah_transparency_log;

CREATE TABLE IF NOT EXISTS public.mah_transparency_log (
  id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  created_at timestamptz NOT NULL DEFAULT now(),
  updated_at timestamptz NOT NULL DEFAULT now(),
  entry_date date NOT NULL,
  title text NOT NULL,
  body text NOT NULL,
  author text,
  visible boolean NOT NULL DEFAULT true
);

ALTER TABLE public.mah_transparency_log ENABLE ROW LEVEL SECURITY;

-- Anyone (anonymous) can read visible entries.
CREATE POLICY "transparency_log_select_visible"
  ON public.mah_transparency_log
  FOR SELECT
  TO anon, authenticated
  USING (visible = true);

-- Founders can insert/update/delete.
CREATE POLICY "transparency_log_founders_all"
  ON public.mah_transparency_log
  FOR ALL
  TO authenticated
  USING (
    EXISTS (
      SELECT 1 FROM public.mah_members m
      WHERE m.id = (SELECT auth.uid()) AND m.tier = 'founder'
    )
  )
  WITH CHECK (
    EXISTS (
      SELECT 1 FROM public.mah_members m
      WHERE m.id = (SELECT auth.uid()) AND m.tier = 'founder'
    )
  );

-- Service-role bypass for backfills / scripts.
CREATE POLICY "transparency_log_service_all"
  ON public.mah_transparency_log
  FOR ALL
  TO service_role
  USING (true)
  WITH CHECK (true);

CREATE INDEX IF NOT EXISTS idx_transparency_log_entry_date
  ON public.mah_transparency_log (entry_date DESC);

CREATE TRIGGER set_updated_at BEFORE UPDATE ON public.mah_transparency_log
  FOR EACH ROW EXECUTE FUNCTION public.update_updated_at_column();
```

Things your agent might get wrong here — catch them:

- Using `auth.uid()` instead of `(SELECT auth.uid())` (planner-cache rule)
- Forgetting `ENABLE ROW LEVEL SECURITY`
- Forgetting the service-role policy (you'll need it later for scripted writes)
- Adding `NOT NULL` columns without defaults (breaks existing rows when migrations run)

Step 2 — Write the Edge Function

Create `supabase/functions/get-transparency-log/index.ts`:

```ts
// supabase/functions/get-transparency-log/index.ts
// GET — returns the most recent visible transparency-log entry. Anon-readable.

import { corsHeaders, handlePreflight } from "../_shared/cors.ts";
import { getServiceClient } from "../_shared/supabase.ts";

Deno.serve(async (req) => {
  const preflight = handlePreflight(req);
  if (preflight) return preflight;

  if (req.method !== "GET") {
    return new Response(
      JSON.stringify({ error: "method_not_allowed", message: "Use GET" }),
      {
        status: 405,
        headers: { ...corsHeaders, "content-type": "application/json" },
      },
    );
  }

  try {
    const supa = getServiceClient();
    const { data, error } = await supa
      .from("mah_transparency_log")
      .select("entry_date, title, body, author")
      .eq("visible", true)
      .order("entry_date", { ascending: false })
      .limit(1)
      .maybeSingle();

    if (error) throw error;

    return new Response(JSON.stringify({ entry: data ?? null }), {
      status: 200,
      headers: { ...corsHeaders, "content-type": "application/json" },
    });
  } catch (err) {
    console.error("get-transparency-log failed:", err);
    return new Response(
      JSON.stringify({ error: "internal", message: "Failed to read log" }),
      {
        status: 500,
        headers: { ...corsHeaders, "content-type": "application/json" },
      },
    );
  }
});
```

Notes:

- Uses `_shared/cors.ts` and `_shared/supabase.ts` — never reimplement these.
- Service-role client is fine here because the table's anon SELECT policy already restricts to `visible = true`. We're using service-role to avoid round-tripping a JWT for a public endpoint.
- Error logging is deliberately vague — never echo the raw `err` to the client.

Step 3 — Edit `transparency.html`

Open `transparency.html` in your editor. Find a sensible place near the top (after the hero, before the existing transparency content) and add:

```
<section id="transparency-log" class="mah-section">
  <div class="mah-container">
    <h2>Last updated</h2>
    <div id="transparency-log-content" class="mah-card">
      <p class="mah-muted">Loading latest entry…</p>
    </div>
  </div>
</section>

<script type="module">
  const SUPABASE_URL = "https://pvugmifdnanffuessqhf.supabase.co";

  async function loadLatestEntry() {
    const el = document.getElementById("transparency-log-content");
    try {
      const res = await fetch(
        `${SUPABASE_URL}/functions/v1/get-transparency-log`,
      );
      if (!res.ok) throw new Error(`HTTP ${res.status}`);
      const { entry } = await res.json();
      if (!entry) {
        el.innerHTML = '<p class="mah-muted">No entries yet.</p>';
        return;
      }
      const date = new Date(entry.entry_date).toLocaleDateString("en-AU", {
        year: "numeric",
        month: "long",
        day: "numeric",
      });
      el.innerHTML = `
        <p class="mah-eyebrow">${date}${entry.author ? " · " + entry.author : ""}</p>
        <h3>${entry.title}</h3>
        <p>${entry.body}</p>
      `;
    } catch (err) {
      console.error(err);
      el.innerHTML = '<p class="mah-muted">Could not load latest entry.</p>';
    }
  }

  loadLatestEntry();
</script>
```

A couple of things to notice:

- We call the EF directly from the browser. CORS is already handled by `_shared/cors.ts` in the function.

- We use `textContent` style assignments here via template strings — be careful with anything user-influenced. For an admin-only-write table this is fine, but for any user-input field, escape it before injection.
- Styling uses existing `mah-section`, `mah-container`, `mah-card`, `mah-eyebrow`, `mah-muted` classes from `styles.css` — we're not adding new CSS.

Step 4 — Test locally

In one terminal:

```
mah serve
```

In another:

```
supabase functions serve get-transparency-log
```

Now you have two problems to solve:

1. **There's no data in the table yet.** Apply the migration first, then insert a test row.
2. **The page calls the production Supabase URL, not your local function.** For local testing, temporarily change `SUPABASE_URL` in `transparency.html` to `http://localhost:54321`, OR test the function with `curl` directly and skip the browser test for now.

Apply the migration locally (against the live database — see the warning in `03-local-setup.md`):

```
psql "$DATABASE_URL" -f backend/migrations/037-add-transparency-log.sql
```

Insert a test row using the Supabase SQL editor or psql:

```
INSERT INTO public.mah_transparency_log (entry_date, title, body, author)
VALUES (
  CURRENT_DATE,
  'Transparency log live',
  'We added a transparency log section to make our weekly updates visible at a glance.',
  'Founders'
);
```

Curl the local EF:

```
curl http://localhost:54321/functions/v1/get-transparency-log
```

Expected response shape:

```
{
  "entry": {
    "entry_date": "2026-04-28",
    "title": "Transparency log live",
    "body": "We added a transparency log section…",
    "author": "Founders"
  }
}
```

If you get an error, the most common causes are:

- Supabase function not running → check the `supabase functions serve` terminal
- Service-role key missing in your `.env` → `mah doctor` will tell you
- RLS blocking the read → test the SELECT directly in the SQL editor

Once curl works, change `SUPABASE_URL` back to the production URL, and visit `http://127.0.0.1:8787/transparency.html`. The new section should render. Reset the URL change before committing — never commit a localhost URL.

Step 5 — Deploy

Three pieces deploy in order:

1. Migration is already applied to production (we applied it directly in Step 4 because there's no separate dev DB). Confirm with:

```
psql "$DATABASE_URL" -c "\d public.mah_transparency_log"
```

2. Deploy the Edge Function:

```
supabase functions deploy get-transparency-log
```

Smoke-test it:

```
curl https://pvugmifdnanffuessqhf.supabase.co/functions/v1/get-transparency-log
```

Tail the logs for a minute to make sure no errors:

```
supabase functions logs get-transparency-log --tail
```

3. Deploy the static site:

```
./release.sh --commit "feat: add transparency-log section to transparency.html"
```

`release.sh` will:

- Commit your staged changes
- rsync to staging on the server
- swap into `/var/www/melbourneaihub/`
- curl the production URL to confirm 200

Wait for it to exit cleanly. If it errors mid-way, read the message; do not retry blindly.

Step 6 — Verify in production

Visit https://melbourneaihub.com.au/transparency.html ⌘ in your browser. You should see your test entry rendered at the top.

Curl the production EF one more time to confirm:

```
curl https://pvugmifdnanffuessqhf.supabase.co/functions/v1/get-transparency-log
```

If both look good, you're done.

Step 7 — Write the decision

Append to `.claude/agent-memory/founders/decisions.md`:

```
## 2026-04-28 — Added transparency-log section

**Why:** Visible weekly updates reinforce our transparency positioning.

**What changed:**

- Added migration `backend/migrations/037-add-transparency-log.sql` — new
  `mah_transparency_log` table with RLS (anon read, founder write).
- Added Edge Function `supabase/functions/get-transparency-log/` — anon-readable,
  returns most recent visible entry.
- Edited `transparency.html` — new section + module script that fetches and
  renders the latest entry.

**Verification:**

- Migration applied and `\d` shows the expected shape.
- `curl https://pvugmifdnanffuessqhf.supabase.co/functions/v1/get-transparency-log`
  returns the seeded entry.
- `melbourneaihub.com.au/transparency.html` renders the section.

**Follow-ups:**

- No admin UI yet — entries are inserted via SQL. Worth building a simple
  founder-only form on `members.html`.
- `book-03-backend/01-edge-functions-index.md` should be updated to list
  `get-transparency-log` (separate small PR).

**Founder:** <your-name>
**Agent:** <claude-code | cursor>
```

Update `.claude/agent-memory/founders/integration-status.md` if any integration row changed (probably none for this task).

Step 8 — Open the PR

```
git push -u origin feat/transparency-log
gh pr create --base main --web
```

In the PR description, paste the decision entry above. The reviewer will:

- Confirm the migration is RLS-compliant
- Confirm the EF uses `_shared/`
- Confirm the HTML doesn't break CSP

After merge, pull `main` locally and delete the feature branch.

Step 9 — Update the docs you used

If anything in this checklist was wrong (commands changed, paths drifted, a step was missing), fix it. That's the easiest second-PR you can ship, and it makes the next founder's first task cleaner. Bump `last-verified` to today.

What you just exercised

Layer	What you did
Database	Wrote a migration with RLS, applied it, seeded data
Backend	Wrote an Edge Function using `_shared/` utilities, deployed it
Frontend	Edited a public HTML page with a module-style fetch, no build step
Deploy	Used `release.sh` for the static site, `supabase functions deploy` for the EF
Verification	Tested locally with curl, smoke-tested production
Memory	Logged the decision, updated integration status
Process	Branched, opened a PR, requested review, merged

That's the whole loop. From here, every change you ship will look broadly like this — different details, same shape.

ewpage

Do-Not-Touch — Surfaces That Bite

This page lists the parts of MAH where a small mistake causes large, sometimes invisible, damage. **Read this before your first PR.** When you do need to change something on this list, pair with another founder, plan carefully, and verify thoroughly.

This isn't about gatekeeping — it's about protecting members and the business. The same agent that handles a frontend tweak in five minutes can silently delete a payment integration if it edits the wrong file with too much confidence. Knowing what's fragile is half the job.

How to read this list

Each item has:

- **What:** the file or area
- **Why it's dangerous:** what breaks if you get it wrong
- **What "touching" means:** how to know when you're in the danger zone
- **If you must:** the safer path forward

When in doubt, open a draft PR with your proposed change and ask another founder to review **before** you implement.

1. Stripe webhook signature verification

What: `supabase/functions/stripe-webhook/index.ts`, specifically the section that verifies `stripe-signature` against `STRIPE_WEBHOOK_SECRET`.

Why it's dangerous: This is the only thing standing between us and a forged webhook payload. If the verification is removed, weakened, or moved before the signature check, anyone on the internet can mark workshop registrations as paid without paying. We lose money silently — there's no error, the payments just never happen.

What "touching" means: Any edit to `stripe-webhook/index.ts`, especially anything before the line `await stripe.webhooks.constructEventAsync(...)` (or whichever helper is currently used).

If you must: Read `book-03-backend/02-ef-stripe-webhook.md` and `book-04-integrations/01-stripe.md` end-to-end. Plan the change with another founder. Test against a Stripe TEST webhook secret first. Use the Stripe CLI's `stripe listen` to replay events. Never deploy this function without a co-founder review.

2. Magic-link generation

What: `supabase/functions/_shared/auth.ts` — the helper that mints magic-link URLs for member login.

Why it's dangerous: Members log in with magic links. If the helper generates malformed URLs, expired tokens, or links that point to the wrong host, every member is locked out. The function uses Supabase's auth admin API, which is forgiving in some ways (it'll happily mint links to nonsense addresses) and unforgiving in others (token format must be exact).

What "touching" means: Any edit to `auth.ts` itself. Also any edit to a function that calls `generateMagicLink()` (`onboard-member`, `send-member-invite`, anywhere a member is invited or re-invited).

If you must: Test against your own email first via `MAH_TEST_MODE_RECIPIENT`. Trace the link end-to-end: generate → SES → click → land on `members.html` → session created. If any step fails silently, stop. Pair-review before deploying.

3. AWS SES SigV4 signing

What: `supabase/functions/_shared/ses.ts` — uses Web Crypto to compute AWS Signature Version 4 for the SES `SendEmail` API.

Why it's dangerous: SigV4 is a multi-step signing protocol. Get any byte of the canonical request wrong (wrong header order, wrong path encoding, missing `\n`) and AWS rejects with a useless error. Worse, if you have an old version cached and a new version partially deployed, some emails go through and some don't — silently failed sends that members never know about.

What "touching" means: Any edit to `ses.ts`. Also any change to the way headers are built before the SES call (e.g., adding a custom `X-…` header that needs signing).

If you must: Read the AWS SigV4 spec. Test with a single send to your own email first. Compare the `Authorization` header byte-for-byte against a known-good request. Deploy only after the test send is delivered.

4. Already-applied migrations

What: Every `*.sql` file in `backend/migrations/` that has already run against production.

Why it's dangerous: Migrations are append-only history. Editing a migration that's already run means production state and migration files diverge. Some teams run migrations idempotently on every deploy — if you change `037-foo.sql` and someone re-runs it, you can drop or alter columns that other code depends on.

What "touching" means: Editing the contents of any existing `*.sql` file in `backend/migrations/`. Even fixing a typo in a comment is a smell — comment changes can wait for the next migration.

If you must: Don't. Always add a new migration with a higher number. If you genuinely need to undo something, write a new migration that does the undo (e.g., `038-revert-bad-thing.sql`) and explain in a top comment what `037` got wrong.

5. `release.sh` itself

What: `scripts/release.sh` — the only sanctioned deploy path.

Why it's dangerous: Every founder deploy goes through this script. If it has a bug — wrong rsync exclude pattern, wrong server path, wrong curl check — every future deploy ships that bug. We've previously had a 195-junk-commit incident on a sister project from raw rsync use. `release.sh` exists specifically to prevent that.

What "touching" means: Any edit to `release.sh`, `scripts/deploy-edge-functions.sh`, or related deploy plumbing.

If you must: Read `book-06-operations/01-deploy-runbook.md` first. Pair with another founder. Run the modified script in `--commit "test"` mode against a throwaway commit before relying on it. Always test on a non-production change before merging.

6. Google Workspace service account JSON

What: The `gws-sa.json` file (typically at `~/.config/mah/gws-sa.json` or wherever `GWS_SERVICE_ACCOUNT_PATH` points).

Why it's dangerous: This file impersonates the MAH GWS workspace. Anyone who has it can read every doc, send mail as `hello@melbourneaihub.com.au`, create Calendar events, and modify Drive permissions. Committing it to git, pasting it into a chat, or uploading it to a shared bucket is an immediate incident.

What "touching" means: The file itself. Also: agent prompts where you might be tempted to paste a snippet to "show what's in it". Don't.

If you must: When rotating the key, generate a new key in the GCP console first, distribute privately, then revoke the old one. Never commit. Never log. Never paste into chat. If the file is exposed, rotate immediately and audit Drive activity for anomalies.

7. Stripe LIVE keys

What: Anything starting with `sk_live_` or referencing the production webhook secret `whsec_…`.

Why it's dangerous: Live keys move real money. A misuse — e.g., test data hitting `sk_live_`, or a debug log printing a live secret — is an incident on multiple axes (financial, security, PCI scope).

What "touching" means: Any time you copy a Stripe key, read environment variables that might contain one, or write code that calls Stripe.

If you must:

- Pay attention to the prefix. `sk_test_` for local. `sk_live_` only in production env vars on the F&W server and in Supabase project settings.
- `.env.example` has placeholders, never real keys.
- If you suspect a live key has been exposed, rotate in the Stripe dashboard immediately and audit recent charges/refunds.

8. The archived `mah-os/` directory

What: The `mah-os/` directory if you see one — historical merge-and-archive from the previous repo split.

Why it's dangerous: It's read-only historical reference. Editing it confuses future archeology and can cause duplicate-source-of-truth bugs if scripts accidentally pick it up.

What "touching" means: Any edit, rename, or delete of files under `mah-os/`.

If you must: Don't. Reference it for context if you're tracking down history; copy any still-relevant logic into the active tree (with attribution in a commit message) rather than reviving the directory.

(Note: the active CLI is the Python package at `/mah/` in this repo. `mah-os/` was archived 2026-04-28.)

9. The 23 historical docs in `docs/` (outside `notebooklm/`)

What: Files like `MAH-MASTER-ARCHITECTURE.md`, `MAH-PROJECT-TRACKER.md`, `STATUS-MAH.md`, `MAH-STATE-ANALYSIS.md`, `CONCEPT.md`, etc., as well as anything in `docs/*.md` that isn't under `docs/notebooklm/`.

Why it's dangerous: These are useful historical context, but they're not all current. Editing them creates two competing narratives (the historical doc and the `notebooklm/` book). NotebookLM and agents both read both, and start contradicting each other.

What "touching" means: Any edit to a file under `docs/` that isn't inside `notebooklm/`.

If you must: Update the corresponding file in `docs/notebooklm/` instead. Bump its `last-verified`. If the historical doc is genuinely misleading and editing notebook content isn't enough, leave a note at the top of the historical doc pointing readers to the current narrative — but don't rewrite the body.

10. Member PII

What: Names, emails, phone numbers, addresses, payment metadata of MAH members and workshop attendees. Anything in `mah_members`, `mah_registrations`, `mah_eoi_submissions`, `mah_assessment_submissions`.

Why it's dangerous: This is private user data. Pasting it into agent prompts (which may log to a third-party LLM provider), committing it to a test fixture, or logging it in an Edge Function creates privacy exposure that's hard to undo.

What "touching" means:

- Selecting member rows in a script and printing them
- Pasting an email or message snippet into a chat with an agent
- Inline test data with real-looking emails (use `test-<id>@example.com`)
- Logging a request body in production EFs

If you must: Anonymise before sharing. For agent prompts, replace real emails with `member@example.com` and real names with `Member A`. For test data, use `data/backups/` seeds with synthetic names. Never paste raw PII into a third-party prompt without a clear business reason and another founder's awareness.

11. CSP / nginx security headers

What: `nginx-security-headers.conf` and the deployed nginx config on the F&W server.

Why it's dangerous: CSP (Content Security Policy) blocks anything not on the allow-list. If a frontend change references a new external service (a new font CDN, a new tracking script, a new Telegram bot endpoint), the browser silently blocks it and members see broken UI. Conversely, weakening CSP — adding `'unsafe-inline'` or `'unsafe-eval'` — opens an XSS door.

What "touching" means: Editing `nginx-security-headers.conf`, `nginx-*.conf`, or anything in `scripts/apply-nginx-hardening.sh`.

If you must: Read `book-02-frontend/06-csp-and-security-headers.md`. Add the minimum specific source needed (e.g., `connect-src https://api.example.com`, not `connect-src *`). Test in your browser's devtools console — CSP violations log there clearly.

12. Co-tenant systems on the F&W server (`15.134.9.54`)

What: Anything outside `/var/www/melbourneaihub/`, particularly `clawdbot` / `openclaw` gateway services owned by TCS.

Why it's dangerous: The F&W server hosts MAH **and** Julio's TCS gateway agents (`clawdbot-gateway` , etc.). Touching their systemd units, killing their processes, or eating shared resources will cause a TCS incident that we then have to explain.

What "touching" means: Any `sudo systemctl` command for non-MAH services, any edit outside `/var/www/melbourneaihub/` , any heavy CPU/RAM-hungry process you forgot to scope.

If you must: Stay inside the MAH directory. If you genuinely need a server-level change (port mapping, new service, etc.), coordinate with Kelebet and Julio first.

Quick rule of thumb

If a change makes you nervous, that's information. Listen to it. Open a draft PR, write your plan in the description, ping another founder. The cost of a 10-minute review conversation is always less than the cost of a 3-hour rollback.

Related reading

- `book-06-operations/01-deploy-runbook.md` — how `release.sh` works in detail
- `book-03-backend/02-ef-stripe-webhook.md` — the webhook function specifically
- `book-09-roadmap-and-gaps/01-status-matrix.md` — known gaps that are slated to make some of these surfaces less fragile

ewpage

Glossary

Keep this open in a tab while you read the rest of the pack. Terms are grouped roughly by concern: organisations, products, infrastructure, services, technical concepts, tooling. When a term appears in commit messages, comments, or chat without explanation, this is where you look.

Organisations and people

MAH — Melbourne AI Hub The product you're working on. A small membership community plus paid workshops plus an AI-curated newsletter, aimed at Melbourne founders building with AI. Public site: `melbourneaihub.com.au`.

TCS — Teddy's Cleaning Services The parent business that funds and incubates MAH. Owned by Kelebet (Teddy). MAH and TCS share some infrastructure (the F&W EC2 server is a co-tenant) but are separate products with separate codebases.

F&W — F&W Services / Julio A client of TCS whose Lightsail/EC2 server (`15.134.9.54`) hosts the MAH static site behind nginx. F&W also runs Julio's own gateway services on the same box. We are co-tenants — see `07-do-not-touch.md` item 12.

Founders The small group with write access to the MAH repo and operational responsibility. You are one of them or you're being onboarded as one.

Members Paying community members who log into `members.html` via magic link.

Workshop attendees People who paid to attend a specific workshop via `workshop.html`. Some convert to members.

EOI submitters People who filled in the Expression of Interest form on `members.html` or via the EOI flow on the public site, signalling they want to join. EOIs are reviewed by founders.

Acronyms you'll see in commits and configs

EF — Edge Function A Supabase serverless function written in Deno + TypeScript. Lives in `supabase/functions/<name>/index.ts`. We have 17 of them. See `book-03-backend/`.

RLS — Row Level Security A Postgres feature that filters rows based on the current user/role. Every MAH table has RLS enabled. Policies are defined per-table and per-operation (SELECT/INSERT/UPDATE/DELETE). See `.claude/rules/migrations.md`.

SES — Simple Email Service AWS's transactional email service. We send all member and workshop email through SES from `hello@melbourneaihub.com.au`, region `ap-southeast-2`. See `book-04-integrations/02-aws-ses.md`.

SigV4 — Signature Version 4 AWS's request-signing scheme. SES (and most other AWS services) require every request signed with SigV4 over the request body, headers, path, and timestamp. Our `_shared/ses.ts` implements it in Web Crypto. Subtle and easy to break — see `07-do-not-touch.md` item 3.

JWT — JSON Web Token A signed token format used for authentication. Supabase issues JWTs to authenticated members. Edge Functions verify them. The anon key and service-role key are also JWTs.

RPC — Remote Procedure Call In Supabase parlance, a database function callable via `supabase.rpc('function_name', { args })`. We use RPCs sparingly; most operations go through Edge Functions instead.

CSP — Content Security Policy HTTP header set by nginx that tells the browser which scripts, styles, and external connections are allowed. Tightening CSP improves security; loosening it (`unsafe-inline`, wildcards) opens XSS doors. Config: `nginx-security-headers.conf`.

PII — Personally Identifiable Information Names, emails, phone numbers, addresses, anything that identifies a real person. Member PII lives in `mah_members` and related tables. Treat with care — see `07-do-not-touch.md` item 10.

EOI — Expression of Interest Pre-membership signal. A prospect fills the EOI form, founders review, then either invite to membership or follow up.

VIP A workshop tier — typically a higher price point with extra inclusions (front-row seating, post-workshop dinner, etc.). Stored as a column on `mah_registrations`.

MX records The DNS records that tell the world where to deliver mail for `melbourneaihub.com.au`. Ours point at Google Workspace (Gmail). Don't change without a clear plan — broken MX = no inbound mail.

Tech and tooling

Supabase Hosted Postgres + auth + storage + Edge Functions. Our database is project ref `pvugmifdnanffuessqhf` in region `ap-southeast-2`. See `book-03-backend/`.

Deno The runtime Edge Functions run on. Web-standards-first (uses `fetch`, `Response`, etc., not Node APIs). No `npm` packages by default — TypeScript files import from URL or `_shared/`.

nginx The web server on the F&W EC2 box. Serves the static MAH site from `/var/www/melbourneaihub/`. Config: `nginx-security-headers.conf`, `mah-nginx-hardening.conf`.

Vanilla HTML / JS / CSS What MAH's frontend is built with. No React, no Vue, no build step, no bundler. The browser loads files directly. See `.claude/rules/frontend.md`.

Magic link A passwordless login method. The system mails you a one-time URL that authenticates you when clicked. We use Supabase's built-in magic link auth, customised via `_shared/auth.ts` and SES-delivered emails.

Stripe Payment processor. Handles workshop tickets and (eventually) member subscriptions. We use Stripe Checkout (hosted page) and a webhook (`stripe-webhook` EF) for fulfilment. Test keys (`sk_test_…`) for local; live keys (`sk_live_…`) for production. See `book-04-integrations/01-stripe.md`.

Stripe webhook An HTTP endpoint Stripe calls when payment events happen (successful charge, refund, dispute, etc.). The signature must be verified using `STRIPE_WEBHOOK_SECRET` — non-negotiable.

GWS — Google Workspace Our domain email (Gmail), shared Drive, Calendar, Sheets. We use a service account to programmatically read/write Drive (PDF mirroring) and per-user OAuth for Gmail/Calendar via `gws-proxy`. See `book-04-integrations/03-google-workspace.md`.

Service account (Google) A non-human identity that can act on Google APIs without an interactive login. Authenticates with a JSON key file. Ours has access to specific Drive folders only.

Master Sheet The Google Sheet referenced by `MAH_MASTER_SHEET_ID` — source of truth for actions, members, workshop seats outside the database. Synced by `backend/sheets-sync/`.

Telegram Messaging app. We have a Telegram bot for member alerts and (planned) passive surveying. The bot runs on a server (currently as part of `backend/passive-logger/`, status PARTIAL).

Claude / Anthropic The LLM provider for `project-agent-chat` (per-project AI for members). Also the LLM behind Claude Code, the recommended agent for founder onboarding.

GLM — Zhipu's General Language Model The LLM used by the newsletter pipeline (`draft-weekly-newsletter`). Cheaper than Claude for the synthesis step. API keys live in `.env` as `GLM_API_KEY`.

OpenAI Used in some places as a fallback or for embeddings. Not the primary LLM in the stack, but expect to see it referenced.

Ollama A tool that runs LLMs locally on your machine. Optional. The newsletter pipeline can use a local Ollama model instead of GLM, if `OLLAMA_BASE_URL` is set.

Firecrawl A web-scraping API used by the newsletter pipeline to ingest articles from external sources. See `book-05-content-pipeline/`.

LightRAG A knowledge-graph indexer used to build the corpus for the newsletter. Indexes scraped articles into a graph that the LLM can query for synthesis. See `book-05-content-pipeline/`.

mmdc — Mermaid CLI Renders Mermaid diagram source files (`.mmd`) into PNGs for GitHub previews. We keep `.mmd` sources in `docs/notebooklm/diagrams/src/` and rendered PNGs in `docs/notebooklm/diagrams/png/`.

NotebookLM Google's document-grounded LLM. We feed it the contents of `docs/notebooklm/` so founders can ask plain-English questions and get cited answers. Access via the MAH GWS account.

launchd macOS's job scheduler (the Apple equivalent of cron). Used on Kelebet's machine for daily restic backups (`com.teddyscleaning.backup`). Plist files live in `scripts/io.melbourne-ai-hub.*.plist`.

PM2 A Node process manager. **Used on the TCS Lightsail server, NOT on MAH.** If you see `pm2` in MAH context, it's probably referring to TCS's adjacent infrastructure. MAH uses Edge Functions (managed by Supabase) and static files (served by nginx). No persistent Node processes.

clawdbot / openclaw TCS's gateway agent system, deployed as systemd units on the same F&W server that hosts MAH. They are co-tenants — do not disturb them. They show up in `systemctl status`, in `/var/log/`, sometimes in `top`. Treat as someone else's process.

Obsidian A markdown-based note-taking app. Kelebet runs sessions and decisions through an Obsidian vault. Some agent memory (session logs) is mirrored there. The vault lives outside this repo, so you don't need to interact with it directly to do MAH work.

Restic A backup tool. Kelebet's daily backups use restic to snapshot to a local NAS (TeddyBot) and an off-site target. Not part of MAH itself; mentioned because backup runs occasionally show up in shell history or cron logs.

Database / schema canon

These are the canonical column names. Do not rename them — old migrations and live data depend on the exact spelling.

Canonical column	Lives on	Wrong names you may hear (do not use)
`session`	`mah_registrations`	`session_preference`
`stripe_payment_intent`	`mah_registrations`	`stripe_payment_id`
`amount_paid` (numeric, dollars)	`mah_registrations`	`amount_cents`
`tier` (`founder` / `member` / `workshop`)	`mah_members`	`role`, `level`
`created_at`, `updated_at`, `deleted_at`	every table	`inserted_at`, `modified_at`, `archived_at`

If an agent proposes the "wrong" name, push back. The naming is load-bearing because old code reads these columns directly.

Environment shorthand

When founders refer to "production" they mean:

- The live `melbourneaihub.com.au` static site behind nginx on F&W
- The live Supabase project `pvugmifdnanffuessqhf`
- The live Stripe account (`sk_live_…`)
- The live SES sender (`hello@melbourneaihub.com.au`)

When they refer to "local" they mean:

- `mah serve` on `127.0.0.1:8787`
- The same Supabase project (no separate dev DB — gap noted in book-09)
- Stripe TEST keys
- SES with `MAH_TEST_MODE_RECIPIENT` set so emails redirect to you

There is currently no "staging" environment. This is in the gap list.

Process shorthand

Squash before push Used on some sister projects (LifeOS) to keep the git history clean. MAH does not enforce this — normal feature-branch + PR + merge-commit flow is fine.

Release Running `release.sh` to deploy the static site. See `book-06-operations/01-deploy-runbook.md`.

EF deploy Running `supabase functions deploy <name>` (or `scripts/deploy-edge-functions.sh` for all). Separate from a "release".

Verify Curl + visual + log-tail after a change. The pack uses "verify" specifically to mean "I have empirical evidence this works in production", not "I think it works".

Memory write Appending an entry to `.claude/agent-memory/founders/decisions.md` (or its sibling files). Required after any non-trivial change.

Plan first Before code, write the plan. Required by the Boris/Anthropic-inspired engineering contract in Kelebet's global config. Easy to skip when an agent is feeling helpful — push back if it does.

Last word

If a term is missing from this glossary and you had to ask another founder what it meant, that's a signal — add it here. Updating this file is one of the easiest first PRs.

ewpage

OpenRouter

OpenRouter is the layer we use to talk to language models. Instead of integrating with Anthropic, OpenAI, Google, Meta, Mistral, Zhipu and DeepSeek separately — each with their own SDK, billing, rate-limits — we send everything to one URL and OpenRouter routes it to the right model.

This page is for founders who have never used a hosted LLM API before.

Why we don't talk to model providers directly

Direct integration	OpenRouter
One API key per provider (5+ to manage)	One key
One SDK per provider (incompatible types)	One HTTP shape — works from `curl`, Deno, Python, browser
Switching models = code change	Switching models = changing one string
Outage in provider X = feature dead	OpenRouter automatically falls back to a substitute
Rate limits managed per provider	Pooled across the route
Bill from each provider	One monthly card

The cost is a small markup on token prices (a few percent). For a small org like MAH, this is the right trade.

How it works in MAH

We hit OpenRouter from two places:

1. **Edge Functions** (`supabase/functions/*`) — server-side, holds the production key in Supabase secrets, used for newsletter drafting (`draft-weekly-newsletter`), per-project agent chat (`project-agent-chat`), and (planned) the founder AI compose feature.
2. **Members portal** (`members.html` + `js/members-*.js`) — client-side flows go through an Edge Function, never directly. The browser **never** sees the OpenRouter key.

Rule: **the OpenRouter key never appears in `members.html`, `*.html`, or any `js/*.js` file.** If you're tempted to put it there to "make a quick demo work," stop and ask — there is always a server-side path.

Models we use today

Model	What for	Why
`google/gemma-2-9b-it:free`	Founder AI compose (planned), free-tier member tools	Free, fast, decent quality
`anthropic/claude-3.5-sonnet`	Per-project agent chat	Best at tool use + long context
`google/gemini-2.0-flash:free`	Newsletter drafting (alternative to GLM)	Long context, free, good at summarisation
`zhipu/glm-4-flash`	Newsletter drafting (current default)	Cheap and reliable for batch summarisation

When we want to try a new model, we change the model string in one Edge Function and redeploy. No SDK changes, no contract renegotiation.

Your personal OpenRouter key vs MAH's production key

There are **two** keys you should be aware of:

1. **Your personal key.** Generated when you signed up (chapter 00). Lives in your password manager. Use this for your own experiments — building a personal Claude Code workflow, testing prompts before adding them to MAH, learning the API. **Never commit it. Never put it in the MAH `.env`.**
2. **MAH production key.** A separate key called `MAH_OPENROUTER_KEY` that lives **only** in Supabase project secrets (`Project Settings → Edge Functions → Secrets`). Edge Functions read it via `Deno.env.get("MAH_OPENROUTER_KEY")`. Founders can rotate it, but day-to-day you do not need to touch it.

If you ever see the MAH key in chat, in a commit, or in a screenshot — that's a security incident. Tell Teddy immediately and we rotate.

Calling OpenRouter from the terminal — your first call

Before touching MAH code, prove your personal key works.

```
curl https://openrouter.ai/api/v1/chat/completions \
  -H "Authorization: Bearer $OPENROUTER_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "google/gemma-2-9b-it:free",
    "messages": [
      {"role": "user", "content": "Say hi in 5 words."}
    ]
  }'
```

Set `OPENROUTER_API_KEY` in your shell first:

```
export OPENROUTER_API_KEY="sk-or-v1-..."  # paste from password manager
```

Expected output (truncated):

```
{
  "choices": [
    { "message": { "role": "assistant", "content": "Hi there, friend!" } }
  ],
  "model": "google/gemma-2-9b-it",
  "usage": { "prompt_tokens": 14, "completion_tokens": 4, "total_tokens": 18 }
}
```

If you get `401 Unauthorized`, the key is wrong or expired. Regenerate it on https://openrouter.ai/keys ⌘.

If you get the response — congrats, you just paid `$0.000003` to Google via OpenRouter to say hi. That's the whole loop.

Streaming output (the good UX)

For an interactive feel, add `"stream": true` and read the response as Server-Sent Events:

```
curl https://openrouter.ai/api/v1/chat/completions \
  -H "Authorization: Bearer $OPENROUTER_API_KEY" \
  -H "Content-Type: application/json" \
  -N \
  -d '{
    "model": "google/gemma-2-9b-it:free",
    "stream": true,
    "messages": [{"role": "user", "content": "Count to 5 slowly."}]
  }'
```

Tokens stream in as `data: {...}` lines. The members area uses this pattern via `fetch().body.getReader()` so the founder AI compose feels like watching it think.

Calling from an Edge Function (production pattern)

```
const key = Deno.env.get("MAH_OPENROUTER_KEY");
if (!key) {
  return new Response(JSON.stringify({ error: "MISSING_KEY" }), {
    status: 500,
  });
}

const upstream = await fetch("https://openrouter.ai/api/v1/chat/completions", {
  method: "POST",
  headers: {
    Authorization: `Bearer ${key}`,
    "Content-Type": "application/json",
    "HTTP-Referer": "https://melbourneaihub.com.au", // OpenRouter wants this
    "X-Title": "Melbourne AI Hub",
  },
  body: JSON.stringify({
    model: "google/gemma-2-9b-it:free",
    stream: true,
    messages: [{ role: "user", content: prompt }],
  }),
});

// pipe the SSE stream straight back to the client
return new Response(upstream.body, {
  status: 200,
  headers: { "Content-Type": "text/event-stream", ...corsHeaders },
});
```

The full reference function is `supabase/functions/founder-ai-compose/` (Phase 5; planned).

Cost shape

OpenRouter prices are listed per million tokens. Most flagship models in late 2025 / early 2026 sit between $0.50 and $15 per million input tokens, $1 to $75 per million output tokens. The free models (`:free` suffix) are zero-cost but rate-limited and not for production volume.

For MAH today, our monthly OpenRouter spend is low double-digit dollars even with weekly newsletters and per-project agent chat. Don't worry about cost yet — worry about getting the call to work.

Where to read more

- OpenRouter docs: https://openrouter.ai/docs ⌘
- Model marketplace: https://openrouter.ai/models ⌘
- MAH-specific integration notes: `book-04-integrations/03-openrouter.md` (status: WORKING)
- Newsletter pipeline reference: `book-05-content/03-newsletter-pipeline.md`

Common errors and fixes

Error	Cause	Fix
`401 Unauthorized`	Bad / expired key	Regenerate at https://openrouter.ai/keys ⌘, update env
`402 Payment Required`	Free credit exhausted on a paid model	Add $5 to your account, or switch to a `:free` model
`429 Too Many Requests`	Rate limit on a free-tier model	Sleep + retry with exponential backoff, or move to a paid tier
`503 Service Unavailable`	Upstream provider outage	OpenRouter usually fails over; if not, switch model string
Empty stream / hangs forever	Network proxy mangling SSE	Use a non-stream call as a fallback (the members area does this — chapter 09 of book-04)
`model not found`	Typo in model string	Check https://openrouter.ai/models ⌘ for the exact ID

Why we don't use Anthropic's API directly

Sometimes we do — Claude Code, your terminal agent, talks straight to Anthropic. That's a different surface (your editor) and it's fine.

For **MAH product features** we route through OpenRouter because we want the option to A/B test models, fall back during outages, and swap providers without code changes. Diversity of route is a small but real form of resilience.

What you can do next

- Run the `curl` example above with your personal key.
- Read `book-04-integrations/03-openrouter.md` for the production-side details.
- Look at `supabase/functions/draft-weekly-newsletter/index.ts` to see a real call in context.
- Skim `supabase/functions/project-agent-chat/index.ts` for the pattern with conversation history.

You won't write a new OpenRouter integration on day one. By the end of week one you should be comfortable changing a model string in an existing function and redeploying.

ewpage

Virtual Servers

A "virtual server" is a computer in a data centre that you rent by the month. AWS Lightsail is the brand we use. The server is on 24/7, has a public IP, runs Linux, and serves the MAH website. This page exists because most non-engineers have never logged into a server, and that's a small but real barrier to becoming useful.

By the end of this chapter you will be able to:

- SSH into the F&W server (where MAH lives).
- Read the Nginx access log to see real visitors.
- Tail the application log of an Edge Function smoke-test.
- Restart Nginx if you ever need to.

You will **not** edit files in production directly. That's covered in `07-do-not-touch.md` and `book-06-operations/01-deploy-runbook.md`.

What a virtual server actually is

When you sign up for AWS Lightsail you're really renting a slice of a much larger physical server in an Amazon data centre. The slice has:

- A copy of Linux (we use Ubuntu 24.04 LTS).
- A small disk (40 GB on our F&W instance).
- A public IP address (`15.134.9.54` for F&W).
- A Bash shell you reach via SSH.
- Whatever you install on it (Nginx, Node, Postgres if you wanted to host your own DB).

It is functionally identical to a physical computer with a network cable, except you can boot a new one in 90 seconds and throw it away when done. That's the whole appeal.

The MAH server fleet

Server	Provider	IP	What it runs
F&W	AWS Lightsail	`15.134.9.54`	Nginx serving `melbourneaihub.com.au` (static HTML + JS + CSS)
TCS	AWS Lightsail	`54.206.129.212`	Apache serving `teddyscleaning.com.au`, jumpbox for psql tunnels
Mo	AWS Lightsail	`3.105.16.151`	Mo Roofing client (separate business, not MAH)
Koraco	AWS Lightsail	`3.104.248.32`	Kora AI client (separate business, not MAH)

You only need to know about **F&W** for MAH work.

What lives where on F&W

```
/var/www/melbourneaihub/          ← the public website (HTML, JS, CSS, images)
/etc/nginx/sites-enabled/         ← the Nginx config that routes traffic
/var/log/nginx/access.log         ← every request to melbourneaihub.com.au
/var/log/nginx/error.log          ← errors from Nginx
~ubuntu/mah-staging-upload/       ← where release.sh uploads new versions
```

The HTML files in `/var/www/melbourneaihub/` are byte-for-byte the same as the files in your local `~/clawd/projects/melbourne-ai-hub/` after `release.sh` runs. That's it — no build step, no transformation. What you see locally is what ships.

Getting an SSH key

If you already have `~/.ssh/id_ed25519` from chapter 02, you can skip ahead. Otherwise generate one:

```
ssh-keygen -t ed25519 -C "your-email@example.com"
# Press Enter through the prompts.
```

Send the **public** half (`~/.ssh/id_ed25519.pub`, *not* the private one) to Teddy. He adds it to the server's `~/.ssh/authorized_keys`.

> The **private** key (`~/.ssh/id_ed25519`, no `.pub`) **never leaves your laptop**. Don't paste it anywhere. Don't email it to anyone. If you do, generate a new one and ask for the old one to be removed from the server.

Your first SSH

Once Teddy has added your key:

```
ssh ubuntu@15.134.9.54
```

If it works, your prompt changes from `~/clawd/...` to `ubuntu@ip-10-1-2-3:~$`. You are now in the data centre.

If it fails:

- `Permission denied (publickey)` → Teddy hasn't added your key yet, or you sent the wrong half.
- `Connection timed out` → your IP isn't allowed (the server has a firewall). Send Teddy your current public IP from https://ifconfig.me ⌘.
- `Host key verification failed` → you have an old fingerprint cached. Run `ssh-keygen -R 15.134.9.54` and try again.

To leave the server: `exit` or `Ctrl-D`.

Three useful read-only commands

Once on the server, these four commands are safe to run any time. They show you what's happening without changing anything.

See the last 20 visitors

```
sudo tail -n 20 /var/log/nginx/access.log
```

Each line is one HTTP request: timestamp, IP, method, path, status code, user-agent. If a member just signed up, you'll see their `POST /functions/v1/onboard-member` here (well — you'll see the request to the static page; the EF call goes to Supabase, not us).

Watch the log live

```
sudo tail -f /var/log/nginx/access.log
```

`Ctrl-C` to stop. Useful when running a deploy or testing the signup flow — you can see your own request hit the server in real time.

Check disk and CPU

```
df -h    # disk usage
free -h  # memory
top -bn1 | head -20  # what's running
```

The F&W instance is small (1 GB RAM, 40 GB disk). If `df` shows the disk above 80%, something is wrong — most likely log files. We rotate them weekly via cron.

See the running processes related to MAH

```
sudo systemctl status nginx
ps aux | grep nginx | head -5
```

You should see Nginx workers as the `www-data` user. Static-site serving is so cheap there's barely any process activity.

Restarting Nginx (only when you have to)

If you change Nginx config in a PR and `release.sh` doesn't reload — or if a smoke test shows the site is down with a 502 — restart Nginx:

```
sudo nginx -t                    # test config first
sudo systemctl reload nginx      # graceful reload (preferred)
sudo systemctl restart nginx     # hard restart (if reload fails)
```

`reload` is the safe one — it keeps existing connections alive while swapping config. `restart` drops everything for a second. Use `reload` unless you have a reason not to.

What you cannot do

- **`sudo rm -rf /var/www/melbourneaihub/`** — you'd nuke the live site. The `release.sh` flow goes via a staging directory for exactly this reason.
- **Edit files in `/var/www/melbourneaihub/` directly** — your edits would be overwritten by the next deploy and you'd be confused. Always edit locally + deploy.
- **`sudo apt install` random things** — you can break the server's ability to boot. If you want a tool installed, ask Teddy.

SSH tips that pay off

1. Add a host alias

In `~/.ssh/config` on your laptop:

```
Host mah-fw
  HostName 15.134.9.54
  User ubuntu
  IdentityFile ~/.ssh/id_ed25519
```

Now `ssh mah-fw` works from anywhere.

2. Use `tmux` for long sessions

If you're doing anything that takes more than a minute (running a backup, watching logs across a deploy), start `tmux` first:

```
tmux new -s mah
# ... do stuff ...
# Detach: Ctrl-b, d  — your shell keeps running on the server even after you SSH out
# Reattach: tmux attach -t mah
```

This saves you from "Why did the deploy stop halfway?" — usually because your laptop went to sleep and your SSH session died.

3. Copy files with `scp`

Pull a file off the server:

```
scp ubuntu@15.134.9.54:/var/log/nginx/access.log ~/Desktop/
```

Push a file up (you almost never want this — `release.sh` is the only deploy path — but useful for debugging):

```
scp local-file.txt ubuntu@15.134.9.54:~/
```

How `release.sh` actually moves files

When you run `./release.sh` from your local repo:

1. It commits your staged changes (or you've already committed).
2. It rsyncs the repo to a staging dir on **your laptop**, applying excludes (no `.git`, no `docs/`, no `backend/`, no `.env`).
3. It scps the staging dir to F&W under `~ubuntu/mah-staging-upload/`.
4. It SSHes into F&W and runs an rsync from the upload dir into `/var/www/melbourneaihub/` with `sudo` and `chown www-data:www-data`.
5. It curls `https://melbourneaihub.com.au` from the server and expects HTTP 200.
6. It also curls `/workshop.html` and expects HTTP 200.
7. It prints "Deployment complete" if both checks pass.

That's the whole flow. There's no magic — every step is plain SSH + rsync. Read `scripts/release.sh` once just to confirm.

Supabase Edge Functions — different server, same idea

Edge Functions don't run on F&W. They run on Supabase's infrastructure (which is itself running on AWS in a different account, in `ap-southeast-2`). You don't SSH into them. You deploy with:

```
supabase functions deploy <name> --project-ref pvugmifdnanffuessqhf
```

And you read logs with:

```
supabase functions logs <name> --tail
```

Mental model: F&W is **our** server. Supabase Edge Functions are **a managed service** we rent. We can SSH into ours and we can't SSH into theirs.

What you should leave with

- Comfortable running `ssh mah-fw`.
- Able to read `/var/log/nginx/access.log` to see traffic.
- Knowing where `/var/www/melbourneaihub/` is and that you don't edit it in place.
- Aware that `release.sh` is the only acceptable way to ship.
- Calm about the fact that "the production server" is just a small Linux box you can log into.

That last bit matters. Servers are demystifying once you've SSHed into one and read a log file. The blast radius is bigger than your laptop, but the commands are the same. Be deliberate, not afraid.

ewpage

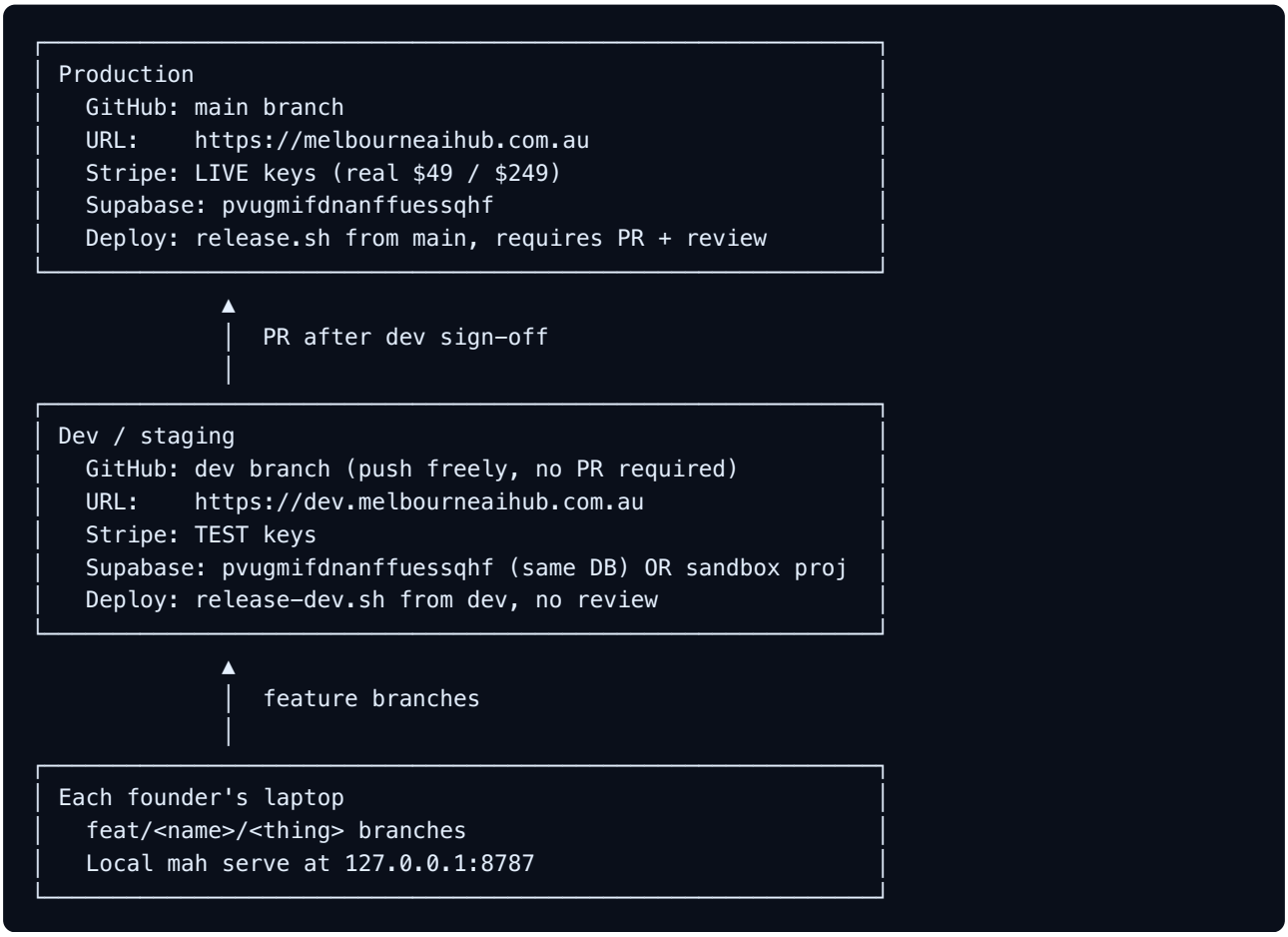
dev.melbourneaihub.com.au — The Shared Dev Environment

> Status: PLANNED. This chapter describes the agreed approach. The dev subdomain ships in the days after the Apr-28 crash course.

The reason this chapter exists: with three founders (Teddy, Olu, Stephen) shipping changes, we cannot have everyone editing main and pushing to production. We need a place to land work-in-progress that real humans can click on, before it goes live.

The plan is to stand up a separate environment at `dev.melbourneaihub.com.au` that mirrors production but runs from the `dev` git branch and uses test Stripe keys + a sandbox Supabase project. Olu, Stephen, and Teddy all push to `dev` freely. Production (`main` → `melbourneaihub.com.au`) stays gated by PR review.

The mental model



Why one repo, not three

We considered giving each founder their own fork or even their own repo. Rejected for these reasons:

- Three repos means three places to keep `_shared` utilities in sync. They drift in 48 hours.
- Three repos means PRs across forks, which is heavyweight in GitHub for `Maintain`-level access.
- Three repos means three Supabase Edge Function deploy lists, three release scripts, three places where bugs hide.

One repo, three branches is the right answer:

- `main` — production. Protected. PR + review required.
- `dev` — shared dev. All three founders can push directly. Auto-deploys to `dev.melbourneaihub.com.au`.
- `feat/<name>/<thing>` — personal feature branches. Live as long as needed, merged into `dev` when the founder is happy with their slice.

How a change flows, in practice

Olu adds a "press kit" page:

1. `git checkout dev && git pull` — get latest dev.
2. `git checkout -b feat/olu/press-kit` — branch off.
3. Edit, commit locally on the feature branch.
4. `git push -u origin feat/olu/press-kit` — push to GitHub.
5. Open a quick PR `feat/olu/press-kit → dev` (one-click on GitHub). Self-merge — no review needed for dev.
6. CI auto-deploys `dev` to `dev.melbourneaihub.com.au` (planned via GitHub Action).
7. Olu, Stephen, and Teddy all click around the dev URL. If something breaks for someone else, they tell Olu in Telegram.
8. Once happy, Teddy (or Olu, with co-sign) opens a PR `dev → main`. **Now** review is required. After Teddy approves and merges, `main` deploys to production.

This gives Olu autonomy to ship to dev, while keeping production gated by a second pair of eyes. Stephen's flow is identical with `feat/stephen/...` branches. Teddy's flow is identical except he has the right to merge `dev → main` without external review.

The dev branch protection settings

When this is set up:

- `dev` branch — no protection. Direct pushes allowed from any founder. Force-push allowed (so you can rebase and clean up history before merging to main).
- `main` branch — `Require pull request before merging`, `Require 1 approval`, `Restrict who can push` set to admins only.

The point of `dev` having no protection is so we don't slow ourselves down with ceremony for what is, by definition, throwaway-able state.

Stripe — separate keys, separate prices

Production uses LIVE Stripe keys (`sk_live_…`). Dev uses TEST keys (`sk_test_…`). The dev environment **never** touches live Stripe. This means:

- Dev workshop signup uses test card `4242 4242 4242 4242` and test prices. No real money.
- Webhook events on dev go to a dev webhook endpoint (registered separately in Stripe).
- The same code path runs — the only difference is which keys are loaded from Supabase secrets.

Until the dev Supabase sandbox exists (see below), dev still uses the production Supabase project, which means dev's Stripe webhook handler updates production rows. We mark those rows as test data via a `is_test_data boolean` column we add later. **Until then, don't let dev signups touch real-looking emails.** Use `dev+something@melbourneaihub.com.au` so they're easy to find and clean up.

Supabase — phase 1 (shared), phase 2 (sandbox)

Phase 1 (this week): dev and prod share the same Supabase project (`pvugmifdnanffuessqhf`). This is fast to set up and matches how MAH was running anyway. Risk: a dev migration could accidentally apply to prod data. Mitigation: nobody runs migrations against prod from a dev branch. Migrations land on `main` only.

Phase 2 (in 1-2 weeks): stand up a separate Supabase project for dev. Use Supabase Database Branches (preview) or a sibling project named `mah-sandbox`. Sync schema from prod weekly via `supabase db dump`. Dev points at the sandbox URL via env vars. This is the right end state.

We do phase 1 first because it gets us to "shared dev URL" in an afternoon. Phase 2 is the proper isolation but takes a week of fiddling.

Server-side — what changes on F&W

On the F&W Lightsail box (`15.134.9.54`):

1. Add a second Nginx `server` block listening on `dev.melbourneaihub.com.au`.
2. Point it at `/var/www/dev-melbourneaihub/`.
3. Get a Let's Encrypt cert for the dev subdomain via certbot.
4. Add the DNS A record `dev.melbourneaihub.com.au → 15.134.9.54` in Route 53.
5. Write a `release-dev.sh` that mirrors `release.sh` but:
    - Reads from the `dev` branch.
    - Targets `/var/www/dev-melbourneaihub/`.
    - Curls `https://dev.melbourneaihub.com.au` for verification.

Or, alternatively (preferred): set up a GitHub Action that triggers on every push to `dev` and runs the deploy automatically. This removes "did anyone deploy?" as a question.

The GitHub Action option is on the medium-term plan because it requires giving the workflow runner an SSH key and rotating it periodically. Acceptable but requires care. For week one, manual `release-dev.sh` from any founder's laptop is fine.

Edge Functions in dev

Edge Functions are trickier because Supabase deploys them per project, not per branch. Two options:

Option A — same project, prefixed function names. Deploy `register-workshop-dev`, `stripe-webhook-dev`, etc. The dev frontend calls the `-dev` versions. Cost: a lot of duplication.

Option B — Supabase database branching (paid feature). Each branch gets its own copy of Edge Functions. Cleaner, costs $10/branch/month.

Option C — separate Supabase project for dev. Same as phase 2 of the database plan. Cleanest but requires standing up a fresh project.

For week one, dev frontend calls **production** Edge Functions but with TEST Stripe keys configured to land in the test mode (Stripe keys determine LIVE vs TEST automatically per request). This works because Stripe's test/live separation is enforced by the key, not by the EF code. The EF accepts both — we just have to be careful about which keys are in scope.

We'll move to option B or C in phase 2 once we have headroom.

Onboarding Olu and Stephen — the 45-minute setup

Goal: get both founders from "I have a GitHub account" to "I just shipped a copy edit to dev.melbourneaihub.com.au" in under 45 minutes during the workshop. The runbook:

Min	Step
0–5	Confirm signup links from chapter 00 are done. Catch up stragglers.
5–10	Teddy adds Olu + Stephen as Maintain on the GitHub repo + Supabase project.
10–15	Both run `git clone git@github.com:teddyscleaningserviceaus-design/melbourne-ai-hub.git`.
15–20	Both `cp .env.example .env` and Teddy walks through which keys to fill (or shares a starter `.env` via 1Password).
20–25	Both `python3 -m venv .venv && source .venv/bin/activate && pip install -e .` and run `mah doctor`. Should be green.
25–30	Both run `mah serve` and load `http://127.0.0.1:8787` — confirm site renders.
30–35	Each picks a small copy edit (a typo, a date, a phrase). Edits in their editor.
35–40	Each runs `git checkout -b feat/<name>/first-edit && git add -p && git commit -m "..." && git push`.
40–45	Open a PR to `dev`, self-merge. Watch the dev deploy run. Click their change live on `dev.melbourneaihub.com.au`.

If we hit blockers, we descope: maybe they don't get to the deploy step in the workshop, but they leave with a working local site and an open PR. That's still a huge win on day one.

What we explicitly defer

- **CI / automated tests on PR.** No tests today; a known gap. We'll add Playwright smoke tests in week 2-3.
- **Per-founder Supabase API key isolation.** Today we share the service-role key. Long-term, each founder gets a personal key with audit trail.
- **Approval-required deploys to prod.** For now, founders trust each other on `main`. If MAH grows beyond three people, this gets formalised.
- **Schema-diff between dev and prod databases.** Phase 2 requirement.

Risks and what to watch for

- **Cross-env contamination.** Dev frontend hitting prod Stripe in LIVE mode would charge real cards. Mitigation: dev `.env.dev` ships with TEST keys only, and the build embeds them. We add a banner to dev that says "DEV ENVIRONMENT — TEST PAYMENTS ONLY" in red.
- **Branch divergence.** If `dev` and `main` diverge for weeks, merging gets hard. Mitigation: rebase `dev` onto `main` after every prod release.
- **Hidden state in prod DB.** If dev points at prod Supabase (phase 1), test rows will mix with real members. Mitigation: seed test members with email pattern `dev+xxx@melbourneaihub.com.au`, delete weekly.

Why this matters

The whole point of MAH having three founders shipping code is that we can move three times as fast. That only works if any one of us can ship to a real URL without breaking the others' work or damaging the live site. `dev.melbourneaihub.com.au` is the cheap solution that buys us that.

The rule we're committing to: **production is sacred, dev is a sandbox, your laptop is your sketchpad.** Protect each according to its purpose.