N°1 VOL. 01 · MAY 2026 NOXLEE AUTOMATIONS

A PRACTICE WORKBOOK

The Claude *Code* Workbook.

Five activities. One installed AI Operating System. 30 days.

AUTHORED BY

Noxlee Automations

NOXLEEAUTOMATIONS.COM

How to use this workbook

This workbook is the *do* half of the course. The Field Guide is the *read* half. The two are designed to be opened side by side: read a module in the Field Guide, then run the corresponding activity here.

The four week-one-through-week-four activities each ship a real, deployable artifact — not a sketch, not a placeholder. The fifth activity assembles those four artifacts into a portfolio page on Vercel that doubles as a sales asset.

Activities are sequential. Activity 2 depends on Activity 1's output, Activity 3 depends on Activity 2's, and so on. Don't skip ahead.

The 30-day cadence

This course runs on weekly checkpoints, not a single sprint. The activities are designed to fit between Monday and Friday of each week with the weekend free:

```
Week 1   →   Activity 1 — Build your CLAUDE.md            (60 min)
Week 2   →   Activity 2 — Ship your first skill           (90 min)
Week 3   →   Activity 3 — Schedule your first autonomous run (60 min)
Week 4   →   Activity 4 — Scope one productized offer     (90 min)
Final    →   Activity 5 — Assemble portfolio, deploy URL  (60 min)
```

Total active time: ~6 hours over 30 days, or ~12 minutes per day if you'd rather sprinkle. The reason for the weekly cadence isn't workload — it's that each artifact needs ~5 days of *living with it* before you know whether it's actually working.

Set up your portfolio folder before you start

Create this folder structure on disk now. The folder name doesn't matter; the file names do.

```
claude-code-portfolio/
  README.md                 ← write at the end (template in Activity 5)
  week1-claude-md.md        ← Activity 1 output
  week2-first-skill/        ← Activity 2 output (whole skill folder)
  week3-scheduled-run.md    ← Activity 3 output
  week3-screenshot.png      ← Activity 3 evidence
  week4-offer.md            ← Activity 4 output
  final-mobile.png          ← Activity 5 output
  final-desktop.png         ← Activity 5 output
  LIVE-URL.txt              ← Activity 5 output
```

Every activity below tells you exactly what file goes where. Treat this folder as the work product. By the end of the workbook, it is a complete portfolio entry and a public landing page — not a draft, not a sketch, but a real artifact you can show a client or hiring manager.

Self-grading rubric (use after each activity)

SCORE	MEANING
5	The output is good enough to send to a real paying client today. No edits needed.
4	Strong. Minor polish would close it but the substance is there.
3	The mechanics worked but the substance is generic — placeholder copy, no real audience-fit, no opinion.
2	Half-finished. Some part didn't render or didn't match the brief. Try the activity again.
1	Didn't run. Setup error, tooling missing, or the prompt was wrong. Read the troubleshooting in the Field Guide and retry.

A score of 4+ on every activity means you have a portfolio piece. A score of 3 means you're still learning the shape of the install — that's fine, but don't pitch a productized offer to a real client until you've re-run the activity to a 4.

Stuck-mode

Every activity includes a **Stuck? Use this prompt** block at the bottom. If you've spent 15 minutes on an activity without progress, paste that exact prompt into a fresh Claude Code session in your project folder. Don't grind. The whole point of the install is that the install is supposed to install itself.

Activity 1 — Build your CLAUDE.md (Week 1, 60 min)

Goal: Produce a real `CLAUDE.md` for a real project — your business, your side project, or your most-used personal codebase. A `CLAUDE.md` Claude Code reads automatically every session and that defines how you and the model collaborate inside this project.

Why this is the first activity

`CLAUDE.md` is the front door of the install. Without it, every session starts from zero. With it, every session inherits your project's vocabulary, your standards, your preferred patterns, and your decision rules — without you having to re-explain them.

This is also the only activity that produces an artifact you'll touch every single workday from this point forward. Get it right and the next three activities ride on top.

Setup

You need: - Claude Code installed (`claude --version` should work in your terminal) - A real project folder. Not a hypothetical. A folder that already has code or content in it that you actually use. If you don't have one, use the folder where you store your daily work files — your "second brain", your invoicing scripts, your blog repo, anything. - 60 minutes of uninterrupted time

> ***Do not skip this for an empty `~/test` folder.*** *A `CLAUDE.md` written for an empty project is a mush placeholder. Pick a real project, even a small one. The activity gets sharper, not harder, with a real subject.*

Step-by-step

Part A — Interview yourself (15 min, no Claude Code yet)

Open a fresh markdown file and answer these six questions. Be specific. One sentence each. Don't be polite about it — the more brutally specific, the better the file.

```
# {{Project name}} — Interview answers (delete this file when done)

## 1. What is this project for?
One sentence. Bad: "My business." Good: "An invoicing automation
that turns Stripe webhooks into Xero invoices for my 12 retainer
clients."

## 2. Who reads or uses what's in this folder?
Be specific. "Just me" is a valid answer. "Me + 2 contractors who
touch the content/ folder" is better. "Public users via the deployed
Next.js app" is better still.

## 3. What's the highest-leverage rule a future Claude session
should know about this project?
The one rule whose absence would cause the most damage. Examples:
"Never push to main without running tests."
"Always use double quotes for strings — single quotes break the
linter."
"Email outreach goes through Gmail API, not raw SMTP — there's a
sending-quota dance documented in scripts/email/README.md."

## 4. What's the highest-leverage decision a future Claude session
should NOT make on its own?
The one judgment call the model should always defer to you on.
Examples:
"Never add a new dependency to package.json — ask first."
"Never publish a LinkedIn post without my approval."
"Never commit anything in credentials/."

## 5. What three commands does a Claude session run most often
in this folder?
e.g. `npm run dev`, `pytest`, `python build.py`, `gh pr create`.
If you don't know yet, list the three you wish it would.

## 6. What's the project's "north star"?
One sentence. The thing that, if Claude can keep in mind, will
make every decision easier. Examples:
"Reduce the time I spend on invoicing from 4hr/week to 30min/week."
"Ship one new client case study per month with zero placeholder
copy."
"Keep the auto-publish pipeline running so I don't have to log in
to LinkedIn during business hours."
```

Part B — Generate the first draft (20 min, in Claude Code)

In your project folder, start Claude Code:

```
cd /path/to/your/project
claude
```

Paste this prompt:

```
I'm installing an AIOS in this project. The first artifact I need is a
CLAUDE.md that you'll automatically load every session.

Please:
1. Read the existing files in this folder to understand the project's shape
   (file structure, existing docs, package.json or equivalent).
2. Read my interview answers below.
3. Generate a CLAUDE.md that includes:
   - A one-paragraph "Project context" block
   - A "Build standards" section with the rules from my interview
   - A "What to ask before doing" section listing the decisions you should
     defer to me
   - A "Common commands" section with the three I named (and any obvious
     ones from package.json / scripts/ etc.)
   - A "North star" closer
4. Keep it under 80 lines. CLAUDE.md is loaded EVERY session, so every line
   pays a context tax. Be ruthless.

Show me the draft, but DO NOT write the file yet. I want to review.

My interview answers:
{paste your answers from Part A here}
```

Part C — Iterate to land (15 min)

Read the draft. If it's a 5/5 first try, you got lucky. More likely you'll redline 30% of it. Things to look for:

- **Stale or generic lines.** Anything that could apply to any Node project, any Python project, any agency — cut it. The whole value of `CLAUDE.md` is project-specific knowledge.
- **Missing tribal knowledge.** That weird gotcha about how the deploy script needs `NODE_OPTIONS=--max-old-space-size=8192`? Add it. Future-you will thank present-you.
- **Wrong vocabulary.** If the model wrote "users" but your team always says "operators", change it. `CLAUDE.md` should sound like *your* project.

When the draft is solid, tell Claude:

```
Write that as CLAUDE.md in the project root. After writing, run
`cat CLAUDE.md | wc -l` to confirm it's under 80 lines.
```

Part D — The 7-day living test (passive)

Use Claude Code normally for the next week. Pay attention to: - Times the model asks a question that's already answered in `CLAUDE.md` → tighten the wording - Times the model breaks a rule → add the rule explicitly - Times the model surprises you with the right judgment call → that's `CLAUDE.md` doing its job, leave it

By Friday, you'll have edited it 2-4 times. That's the point. Save the final version as `week1-claude-md.md` in your portfolio folder.

Hard rules for this activity

- **80-line cap is real.** Anything bigger and the file starts costing more than it saves. If yours grows past 80 lines, split rules into a `.claude/rules/` directory loaded on demand instead.
- **Do not paste a generic CLAUDE.md template you found on the internet.** They look impressive and add zero project-specific value. The interview is the value.
- **Do not put credentials, tokens, or client names in `CLAUDE.md`.** It's a documentation file; assume it'll get committed to git.

Output

`CLAUDE.md` in your project root, plus a copy saved to your portfolio folder as `week1-claude-md.md`.

Self-grade (1-5)

Read the file out loud. Three tests:

1. Could a senior engineer who's never seen this project make a non-trivial change to it correctly using *only* what's in `CLAUDE.md`? If yes, score 4-5. If no, the file is missing tribal knowledge — add it.
2. Is there any line that could apply to *any* project in your industry? If yes, score 3 — rewrite that line specifically or cut it.
3. Did you notice the model behaving differently in your sessions this week? If you can name one specific moment ("Tuesday, when I was editing the deploy script, it stopped me before I pushed without running tests"), score 5.

What this unlocks for your business

Every future Claude Code session in this project starts pre-aligned. That's hours of re-explaining context per week — at a typical 10 hr/week of Claude Code use, an effective `CLAUDE.md` saves 60-90 min of re-explanation alone, before any of the install's downstream wins compound.

Stuck? Use this prompt

If by minute 45 you don't have a working draft, paste this into a fresh Claude Code session in the project folder:

```
I'm trying to build a CLAUDE.md for this project but I'm stuck.
Please:
1. Read everything in this folder.
2. Tell me the FIVE most important things you wish I'd told you about
   this project before you started reading.
3. Draft a CLAUDE.md based on those five things and the file structure
   you observed.

I'll redline yours instead of writing mine from scratch. Keep it
under 80 lines.
```

This inverts the activity — Claude tells you what it wishes it knew, and you confirm/correct.

Activity 2 — Ship your first skill (Week 2, 90 min)

Goal: Build a real, invokable Claude Code skill that does one thing your project needs done repeatedly. By the end, you can type a keyword in any session and the skill loads, reads its context, and does its job.

Why this is the second activity

Skills are how the AIOS scales. `CLAUDE.md` aligns Claude with the project. Skills hand Claude reusable, narrow capabilities — like browser tabs that open on demand instead of always-loaded.

This is the activity most people get wrong. They build one giant "do everything" skill and call it done. The whole point of the skill system is **modular**. The 200-line cap is a teachable rule. Read Module 2 of the Field Guide before starting if you haven't.

Setup

You need: - Activity 1 complete (`CLAUDE.md` in your project root) - A clear sense of one repetitive task in your project. Examples: - "Every time I write a LinkedIn post, I want it auto-fact-checked against my knowledge base." - "Every time I export a Stripe invoice, I want it transformed for Xero import." - "Every time I add a new client, I want their folder scaffolded with the standard 6 sub-folders and a `client.md` ." - "Every time I add a new YouTube video to my catalog, I want a draft LinkedIn post + tweet generated." - 90 minutes

> ***Pick a task you do at least once a week.*** *Skills you invoke once a quarter aren't worth building — you'll forget the keyword. Pick frequency over importance for your first one.*

Step-by-step

Part A — Scope the skill (10 min)

Open a fresh markdown file. Answer four questions:

```
# {{Skill name}} — Scope

## 1. What's the trigger phrase / keyword?
Pick one short, memorable phrase. Bad: "linkedin-post-helper".
Good: "draft-li-post". Will be matched by Claude's skill discovery.

## 2. What input does the skill need from me when I invoke it?
Examples: "the URL of the YouTube video", "the client name and tier",
"the topic of the LinkedIn post". List them as required vs optional.

## 3. What artifact does the skill produce?
A file? A web request? A console output? Be specific about the
format and the destination.

## 4. What's the ONE thing this skill should always do, no matter
how I phrase the invocation?
The non-negotiable. Examples: "Always check the lead-magnet brand
guide before generating LinkedIn copy." "Always confirm the Xero
sandbox vs production toggle before exporting."
```

Part B — Scaffold the skill folder (10 min)

In your project root, run:

```
mkdir -p .claude/skills/{your-skill-name}/references
touch .claude/skills/{your-skill-name}/SKILL.md
```

The `references/` subfolder is for files the skill loads on demand — examples, templates, longer prompts. Anything that doesn't need to be in `SKILL.md` itself.

Part C — Generate the SKILL.md (30 min)

In Claude Code:

```
I'm building a Claude Code skill in this project. The folder is at
.claude/skills/{your-skill-name}/. Here's my scope (paste your Part A
answers below).

Please draft SKILL.md. Constraints:
1. ≤ 200 lines. Hard cap. Anything longer goes in references/ and gets
   loaded on demand from the SKILL.md.
2. Start with frontmatter:
   ---
   name: {your-skill-name}
   description: {one-line what this skill does, ≤120 chars, will be
     matched by Claude's skill discovery against my prompts}
   ---
3. After frontmatter, sections:
   - Trigger phrases (the keyword + 2-3 variants)
   - Required inputs (with examples)
   - Workflow (numbered steps the skill takes when invoked)
   - Output format (the artifact this produces)
   - References (which files in references/ to load and when)
4. Reference my CLAUDE.md vocabulary and standards — this skill should
   sound like the project, not a generic template.

Show me the draft. Don't write yet.

Scope:
{paste your Part A answers}
```

Part D — Build the references (15 min)

Look at your draft. Anywhere it says "here's an example" or "the format is", pull that into `references/<filename>.md`. Examples: - `references/linkedin-style-guide.md` — your tone-of-voice rules - `references/xero-csv-format.md` — the exact column mapping - `references/client-folder-template.md` — the 6 sub-folders and what goes in each

The `SKILL.md` should reference these by relative path: `Load references/linkedin-style-guide.md before drafting.`

Part E — Test the invocation (15 min)

Quit Claude Code. Restart it. In a fresh session in the project folder, invoke the skill by its trigger phrase:

```
draft-li-post for https://www.youtube.com/watch?v=...
```

Claude should: 1. Detect the trigger phrase 2. Load the skill 3. Confirm it has the inputs it needs 4. Run the workflow 5. Produce the artifact in the format you specified

If it gets stuck at step 1 (didn't detect the trigger), your `description` frontmatter wasn't keyword-rich enough — rewrite it. If it gets stuck at step 2 (loaded but asked for the wrong input), your "Required inputs" section was unclear. If it produces the artifact in the wrong format, your "Output format" section needs to be more prescriptive.

Iterate until invocation → artifact in one prompt.

Part F — The 7-day usage test (passive)

Use the skill in your normal week. Track: - How many times you invoked it - How many invocations produced a usable artifact on the first try - Which times you had to re-prompt with extra context (those bits should be added to references/)

By Friday, copy the whole `.claude/skills/{your-skill-name}/` folder to your portfolio as `week2-first-skill/`.

Hard rules for this activity

- **One skill, one job.** Resist the urge to add a second job "while I'm in here." Build a second skill instead.
- **200 lines or less in `SKILL.md`.** This is the rule that makes skills modular. Anything longer goes to `references/` and loads on demand.
- **The trigger phrase must be unique in your project.** If you have two skills that could both fire on "draft post", Claude will pick one and you'll never know which.
- **Do not wrap a CLI tool in a skill if a one-line bash alias would do.** Skills are for *judgment* tasks (drafting, transforming, deciding). Pure mechanical scripts belong in `scripts/`.

Output

`.claude/skills/{your-skill-name}/` in your project root, plus a copy at `week2-first-skill/` in your portfolio folder.

Self-grade (1-5)

1. Did you invoke the skill at least 3 times in the 7-day test? If no, you picked a task that wasn't actually frequent — pick another for next time. Score 2.
2. Of those invocations, what % produced a usable artifact on the first try? <50% = 2. 50-80% = 3. 80-95% = 4. 95%+ = 5.
3. Could another operator on your team invoke this skill correctly without you explaining it? If yes, you wrote a real skill. If no, the description / trigger phrases need to be more obvious.

What this unlocks for your business

A library of skills compounds. The first one takes 90 min and saves ~30 min/week. The fifth one takes 60 min (you got faster) and saves ~30 min/week. By skill #10 you've got 5 hours/week of recurring work moved off your plate, against ~10 hours of one-time setup. That's the leverage curve everyone in wiki/04 (AIOS install playbook) is pointing at.

Stuck? Use this prompt

If your skill isn't firing on the trigger phrase, paste this in a fresh session:

```
I built a skill at .claude/skills/{your-skill-name}/SKILL.md but
when I type "{your trigger phrase}" you don't seem to load it.

Please:
1. Read SKILL.md in that folder.
2. Tell me the EXACT trigger phrases you would have detected.
3. Suggest a rewrite of the `description` frontmatter that would
   make detection more reliable.

I'll update the description and re-test.
```

Activity 3 — Schedule your first autonomous run (Week 3, 60 min)

Goal: Set up a real recurring Claude Code task — daily, weekly, or on a webhook — that produces a real artifact without you starting it manually. By the end, you wake up Monday and a digest, report, or scaffolded folder is already waiting for you.

Why this is the third activity

`CLAUDE.md` aligns the model. Skills give it reusable capabilities. Scheduling gives it **agency** — the ability to run when you're not watching. This is the moment the install crosses from "tool I use" to "system that works for me."

This is also the activity most underrated. People obsess over the first two and skip this one because it feels like infrastructure. It's not — it's where the leverage actually shows up.

Setup

You need: - Activity 2 complete (one working skill) - One repetitive task you'd love to wake up to. Examples: - "A daily digest of the LinkedIn DMs I haven't replied to, sorted by deal size" - "A weekly

summary of every PR merged in my client's repo with one-line context for each" - "A daily scrape of the 3 manhwa series I follow with new chapters flagged" - "A Friday-afternoon draft of next week's content schedule" - A way to schedule the task. Options: - **Cron** (Mac/Linux/WSL) — built-in, free, opaque - **Windows Task Scheduler** (Windows native) — built-in, free, GUI - **Claude Code's `/schedule` skill / managed agents** — easier, paid (per Anthropic's pricing) - **n8n cron node** triggering a Claude Code shell call — best if you already run n8n - 60 min

> ***Pick a task whose absence is mildly painful.*** *If you'd be fine never seeing the artifact, you'll never check whether the schedule fired. The activity needs stakes.*

Step-by-step

Part A — Define the task (10 min)

Open a fresh markdown file:

```
# Scheduled Task: {{name}}

## 1. What artifact does this produce?
Be specific about format and location. Examples:
- Markdown file in ~/Notes/daily/{{date}}.md
- Email to me with subject "Weekly LinkedIn digest"
- Slack message to #ops-daily

## 2. When should it run?
Cron expression or English. Examples: "Every Monday at 6am", "Daily
at 7am except weekends", "Every 4 hours", "Every Friday 4pm".

## 3. What inputs does it need?
Anything the run needs to read first — a CSV, a folder, an API.

## 4. What's the success signal?
How do you know it ran AND produced the right thing? "The file
exists and is non-empty by 6:05am" is a good answer. "The model
didn't error" is a bad one — silent wrong-output is worse than
loud failure.

## 5. What's the failure mode you most fear?
Examples: "Sends the wrong digest to the wrong client", "Posts
half-baked LinkedIn drafts publicly", "Burns API quota in a loop".
Bake the mitigation into the task NOW, not after the first
incident.
```

Part B — Build the runnable script (20 min)

In Claude Code in your project folder:

```
I'm building a scheduled Claude Code task in this project. The task
spec is below. Please:

1. Build a single shell script (or .py / .ps1 — whatever fits the
   project) that runs Claude Code headless on this task and writes
   the artifact to the path specified.
2. Use `claude --print` (non-interactive) so the run finishes and
   exits without a TTY.
3. Add a "guardrail" check: if the artifact looks empty / malformed,
   don't overwrite the previous one — write to a .failed file
   instead. Send me an alert (email or just a file marker).
4. Show me the script. Do not yet write it.

Task spec:
{paste your Part A answers}
```

Review the script. Test it once manually:

```
bash scripts/scheduled-task.sh   # or your equivalent
```

Confirm the artifact exists and is correct. **If the manual run isn't right, do not schedule it.** A broken cron is worse than no cron — it gives you false confidence that you're covered.

Part C — Schedule it (15 min)

Pick the platform that matches your OS. The Field Guide Module 3 walks through each in detail. Below is the minimum.

Cron (Mac / Linux / WSL):

```
crontab -e
```

Add a line. Example: every weekday at 6:30am:

```
30 6 * * 1-5 /bin/bash /full/path/to/scripts/scheduled-task.sh >> /full/path/to/logs/cron.log 2>&
```

The `>> ... 2>&1` part is non-negotiable — without logs, you have no way to debug a silent failure.

Windows Task Scheduler:

```
$action = New-ScheduledTaskAction -Execute "powershell.exe" -Argument "-File C:\path\to\scripts\s
$trigger = New-ScheduledTaskTrigger -Daily -At 6:30am
Register-ScheduledTask -Action $action -Trigger $trigger -TaskName "claude-daily-digest"
```

Claude Code managed agents (`/schedule`):

```
/schedule run scripts/scheduled-task.sh every weekday at 6:30am
```

(Adjust to your actual Claude Code version's schedule syntax — check `claude --help`.)

Part D — The 7-day verification (passive)

Each morning of the 7-day test, before opening the artifact, ask yourself: *did it fire?* Then check.

Track: - # of expected runs - # of successful runs - # of `.failed` markers - # of times the artifact was actually correct vs technically-produced-but-wrong

If success ratio is <90% by Friday, the task isn't reliable enough to keep — debug before moving on.

By Friday, take a screenshot of the most recent successful artifact and save as `week3-screenshot.png`. Save your task spec as `week3-scheduled-run.md`.

Hard rules for this activity

- **No scheduled task that posts publicly without a human review step.** Drafts to a folder, drafts to your inbox — fine. Auto-publishing to LinkedIn or Slack #general — never on the first build. Add the human-in-the-loop checkpoint, then automate it away later once you trust it for 30+ runs.
- **Always log to a file.** A scheduled task with no log is a black box. Cron's default behavior is to silently swallow output and email you on stderr — both wrong.
- **Run the script manually FIRST.** If the manual run isn't right, the schedule will reliably produce wrong output on a schedule. That's worse than nothing.
- **Set a kill-switch.** Know how to disable the task in 30 seconds. `crontab -r` deletes everything; you want a single line you can comment out.

Output

- `week3-scheduled-run.md` — your task spec
- `week3-screenshot.png` — the most recent successful artifact

Self-grade (1-5)

1. Did the task fire on schedule for 5+ consecutive expected runs? <3 = 2. 3-4 = 3. 5+ = 4. 7+ with zero `.failed` markers = 5.
2. Was the artifact useful? "I read it every day" = 5. "I read it once and skimmed the rest" = 3. "I forgot it existed by Wednesday" = 1 — kill the task and pick a different one.
3. Could you disable it in 30 seconds if it started misbehaving? If no, build the kill-switch before scoring.

What this unlocks for your business

This is where the install starts paying you in your sleep. A daily 10-minute task that runs without you = ~50 min/week of recovered time, but more importantly, it's an artifact that *exists when you wake up*. That changes how you start the day, which changes what you get done.

The bigger unlock: this is the foundation for the productized offers in Activity 4. "Internal reporting & status notifications" (one of the 3 high-margin offers from wiki/07) is exactly this pattern, scaled to a client.

Stuck? Use this prompt

If the cron / schedule fires but the artifact is wrong (or missing), paste this in a fresh session:

```
My scheduled Claude Code task at scripts/scheduled-task.sh is firing
but the output isn't what I expected. Please:
1. Read the script.
2. Read the most recent log file at logs/cron.log.
3. Tell me the THREE most likely reasons the artifact is wrong.
4. For the most likely one, suggest a specific fix.

Don't write the fix yet — explain it first.
```

Activity 4 — Scope one productized offer (Week 4, 90 min)

Goal: Take everything you've installed and turn it into a one-page proposal you could send to a real prospective client today. By the end, you have a real offer with real ROI math, scoped against a real scenario.

Why this is the fourth activity

This is the money module. Activities 1-3 are the install. Activity 4 is what you sell *because* of the install.

The three productized offers from wiki/07 (May-3 batch synthesis) are:

1. **Document Processing** (insurance, law, accounting, logistics) — pitched as $70K/yr labor savings
2. **Database Reactivation** (gyms, SaaS, coaching) — the "1,200% ROI in 60 days" story
3. **Internal Reporting & Status Notifications** (every business) — Slack/email automation, the stickiest sale

Each one is a real, productized service with proven ROI math. Pick the one that fits your install best, run it through the proposal template, and you have a real sales asset.

Setup

You need: - Activities 1-3 complete (so you can speak to a real install, not a hypothetical) - A real prospective client in mind. Even "the small accounting firm two blocks down" works. Don't write the proposal for "businesses" — write it for *one named business*. - 90 min

> ***Do not write a generic offer.*** *"I help businesses with AI" is not an offer. "I help [specific firm] cut their invoicing review time from 8 hr/week to 30 min/week" is an offer. Activity 4 is the test of whether you can be specific.*

Step-by-step

Part A — Pick the offer (10 min)

Read the three offers in wiki/07 §"Tier 1 — adopt this week". Pick the one that matches:

- **The install you actually built.** If your scheduled task in Activity 3 was a daily digest of internal data, "Internal Reporting" is your natural match. If your skill in Activity 2 transformed documents (invoices, contracts, intake forms), "Document Processing" is the match. If your project touches a CRM or contact database, "Database Reactivation" is the match.
- **The client you have in mind.** Some offers fit some industries better. Don't pitch Database Reactivation to a one-person law firm.

Write down: **Offer name + Target client name + Why this offer for this client.** One sentence each.

Part B — Run the ROI math (20 min)

The whole reason these offers work is the ROI math is concrete. Open a fresh markdown file:

```
# {{Offer name}} for {{Client name}} — ROI Math

## Current state (no AI)

- Hours/week the client spends on [this task]: {{your estimate}}
- Hourly cost (loaded): ${{rate}}
- Weekly cost: ${{hours × rate}}
- Annual cost: ${{weekly × 50}}
- Error rate / rework cost: ${{your estimate}}/yr (optional but powerful)
- Total annual pain: ${{sum}}

## Proposed state (with the install)

- Hours/week reduced to: {{your estimate}}
- Annual savings: ${{difference}}

## Investment

- One-time install: ${{your fee, e.g. $5,000-15,000}}
- Monthly retainer (optional, for ongoing tuning): ${{e.g. $500-1,500}}
- Total Year 1 cost: ${{sum}}

## ROI

- Year 1 net savings: ${{savings - cost}}
- Payback period: {{cost / monthly savings}} months
- 3-year ROI: {{(3yr savings - cost) / cost × 100}}%

## The closer line

"For ${{install fee}} one-time, you save ${{annual savings}} every year forever. The math says th
```

Fill in real numbers. Don't bracket-leave. If you don't know the client's hourly cost, look it up — the BLS publishes loaded rates by industry, and a 30-second Google gives you a defensible estimate.

Part C — Write the one-pager (40 min)

Open `week4-offer.md`. Use this exact structure (the doctor-not-pharmacist + lead-with-outcome structure from wiki/07 §"Sales / positioning principles"):

```
# {{Offer name}} for {{Client name}}

## The outcome

One sentence. Outcome the client gets, in their language.
Bad: "AI-powered document processing automation."
Good: "Cut invoice review time from 8 hours/week to 30 minutes,
       so your office manager stops working Saturdays."

## The diagnosis (doctor, not pharmacist)

Two-three sentences. What's the actual clog in their pipe?
Bad: "You don't have AI yet."
Good: "Right now, every supplier invoice gets manually keyed into
QuickBooks. That's 6 hr/week, ~$15K/yr in labor, and at least
one transposition error a month that costs another ~$2K each
to chase down."

## What we'll install

3-5 bullets. Specific, technical-enough-to-credible, but in their
vocabulary. Reference your actual install.
- A {{your skill from Activity 2}} that {{specific transform}}
- A {{your scheduled task from Activity 3}} that {{specific cadence}}
- A {{your CLAUDE.md vocabulary}} that {{specific alignment with their workflow}}

## ROI

Insert the numbers from Part B as a 2-line summary:
"Year 1 net savings: ${{X}}. Payback in {{Y}} months. 3-year ROI: {{Z}}%."

## What this is NOT

This is the trust line — name what the offer does NOT do, so they
don't expect it. Examples:
- "This is not an AI chatbot for your customers."
- "This does not replace your office manager — it removes the
  worst part of her job so she can do the parts she's actually good at."
- "This is not 'set it and forget it' — we'll spend the first 30 days
  tuning it together based on edge cases that surface."

## Investment + timeline

- One-time install: ${{X}} (paid 50% on signing, 50% on first
  successful run)
- Optional monthly retainer: ${{Y}}/mo (you can cancel anytime)
- Timeline: {{2-4}} weeks from kickoff to first scheduled run
- Pilot option: {{paid micro-task — e.g. "$1,500 to install one
  workflow as a proof, refunded against full install if you proceed"}}
```

```
## Why us

Two sentences max. Reference your actual install (Activities 1-3).
Bad: "We're experts in AI automation."
Good: "We've installed this exact pattern in our own operation:
{{public URL of your portfolio Vercel page from Activity 5}}.
The same pattern that runs our [your business] day-to-day is
what we'll install in yours."

## Next step

One specific, easy yes.
"Reply with a 30-min slot this week and we'll diagnose your
{{specific workflow}} together. No charge, no slides — just
a working session."
```

Hard rules for this activity

- **No "free trial weeks."** This contradicts your durable preference (per saved feedback). De-risk via a paid pilot with refund clause OR a scoped paid micro-task. Free 30-min discovery call is fine; free build is not.
- **No "and we can also..." paragraphs.** Each offer is one offer. Adding three more services to "increase the value" makes the offer mushier and reduces conversion.
- **No fake numbers.** ROI math has to be defensible if the client asks "where did you get 8 hours/week?" "I asked you in our call" or "industry benchmark from the BLS" are both fine answers. "I made it up" is not.
- **No reference to your install that you couldn't show them.** "We've installed this in our own operation" only works if Activity 5 is shipped and they can click the URL.

Output

`week4-offer.md` in your portfolio folder. One page. ROI math defensible. Specific to one named client.

Self-grade (1-5)

1. Could you send this to the named client today, by name, with no further edits? If yes = 5. If you'd be embarrassed = 2 — rewrite the diagnosis section, that's almost always the weak point.
2. Read the ROI math out loud. Could you defend every number against "where did you get that?" If yes = 4-5. If any number is bracketed, hand-wavy, or "industry standard" without a source = 3.
3. Show the one-pager to a peer who runs a business. Ask: "If a contractor sent you this, would you take the discovery call?" Yes = 5. Maybe = 3. No = rewrite the outcome line.

What this unlocks for your business

This is the asset that turns the install into revenue. One offer document, scoped specifically, sent to one named prospect, has a meaningfully higher reply rate than ten generic emails. And once it's written, it's a template — the second offer takes 30 min, not 90, because the structure is locked.

The compounding play: by the end of the 30-day install, you have one offer ready. By Day 60, with another 90 min of work each, you have all three. Then your discovery calls become "which of these three fits you" instead of "do you want me to think about what AI could do for you." The conversion math changes.

Stuck? Use this prompt

If the offer reads generic, paste this in a fresh session:

```
I'm trying to write a productized AI offer for {{specific client name}}
in the {{industry}} industry. I've drafted week4-offer.md but it
reads generic.

Please:
1. Read week4-offer.md.
2. Imagine you ARE {{client name}} reading it for the first time.
3. Tell me the THREE specific lines that would make {{client name}}
   click "delete." Be brutal.
4. For each, suggest a specific rewrite that names {{client name}}'s
   actual workflow.

Use my CLAUDE.md vocabulary if anything in there overlaps with
the client's industry.
```

Activity 5 — Assemble portfolio + deploy URL (Final, 60 min)

Goal: Take the four artifacts you've built across the previous 30 days and assemble them into a single coherent portfolio page deployed to a public Vercel URL. This is the asset you point to from sales conversations, LinkedIn posts, and your one-page offer (Activity 4).

Setup

Open your portfolio folder. You should have:

- `week1-claude-md.md` — your `CLAUDE.md`
- `week2-first-skill/` — your skill folder
- `week3-scheduled-run.md` + `week3-screenshot.png` — your scheduled task spec + evidence

- `week4-offer.md` — your one-pager

If anything is missing or marked at score 2 or below, go back to that activity. The portfolio is only as credible as its weakest artifact.

You also need: - A free Vercel account - Either Next.js or any static site framework you're comfortable with (this course assumes Next.js but anything that builds to static HTML works) - Optionally: domain control if you want to mount this at a sub-route of an existing site

Step-by-step

Part A — Generate the portfolio page (30 min)

In Claude Code in your project folder:

```
I want to deploy a single-page portfolio to Vercel that showcases
my Claude Code AIOS install. The audience is prospective clients
who'd hire me for the productized offer in week4-offer.md.

Please:

1. Read all the artifacts in my portfolio folder:
   - week1-claude-md.md
   - week2-first-skill/SKILL.md
   - week3-scheduled-run.md
   - week3-screenshot.png
   - week4-offer.md

2. Generate a single Next.js (or Vite + React, your call) page that
   has these sections in order:

   a. HERO — The outcome line from week4-offer.md, big. Subtitle: my
      one-line positioning. CTA button: "Book a 30-min discovery call"
      → mailto: noxle.chumperum@gmail.com (or whatever I tell you).

   b. THE INSTALL — Three columns, one per artifact (CLAUDE.md, Skill,
      Scheduled task). Each column: name, one-paragraph description,
      and a screenshot or code snippet preview.

   c. THE OFFER — A condensed version of week4-offer.md. Outcome,
      diagnosis, ROI bullet, "what this is NOT," one CTA.

   d. WHY THIS WORKS — One paragraph: the install you see above is
      the same install I'd put in their business. Link out to anything
      else (your wiki, your case studies, your existing portfolio).

   e. CONTACT — Email + LinkedIn + book-a-call. Same CTA as the hero.

3. Style: midnight navy + champagne gold (matches my Field Guide
   brand). Inter font. Use the exact CSS from
   knowledge-bases/claude-cowork/course/assets/style-guide.css as
   the starting palette.

4. Make it responsive. Mobile-first.

5. Ship the build instructions: `npm run build && vercel --prod`.

Do not write the files yet. Show me the structure + the hero + the
first column of "The Install" so I can confirm the direction.
```

Iterate once or twice on the draft. When it's right:

```
Build it. After building, run `npm run dev` and tell me the
localhost URL.
```

Open the localhost URL. Check on mobile (use Chrome DevTools mobile preview or your phone on the local network).

Part B — Deploy (20 min)

If you're mounting this at an existing domain (recommended: `noxleeautomations.com/claude-code` route), the structure depends on your existing site. The simplest path:

1. Add a new route `/claude-code` to your existing Next.js project
2. Drop the page generated in Part A into that route
3. `git push` triggers Vercel auto-deploy

If you don't have an existing site:

```
npx vercel
```

Vercel asks four questions; defaults are fine for the course: - Set up and deploy? → Y - Scope → your account - Link to existing project? → N - Project name → `claude-code-portfolio` or similar

Wait 60-90 sec. Vercel prints a production URL like `your-project.vercel.app`. Save it to `LIVE-URL.txt` in your portfolio folder.

Part C — Verify and screenshot (10 min)

1. Open the production URL on your phone. Take a screenshot. Save as `final-mobile.png`.
2. Open the production URL on your desktop browser. Take a full-page screenshot. Save as `final-desktop.png`.
3. Test the email CTA — does `mailto:` open with the right address?
4. Test the link to your one-pager / wiki — does it resolve?

Hard rules for this activity

- **Test the URL from a network you're not building on.** Localhost-only success doesn't count. Use your phone on cellular or an incognito window.
- **The page must reference real artifacts.** No placeholder lorem ipsum. If a section needs content you don't have, cut the section — don't pad it.
- **Lead-capture form is optional for v1.** A clean `mailto:` CTA converts well enough for the first 50 visitors. Add a Vercel form + Sheets sink in v1.1 once you have signal.

Output

- `LIVE-URL.txt` — production URL
- `final-mobile.png` — mobile screenshot
- `final-desktop.png` — desktop screenshot

Self-grade (1-5)

1. Loads in <1 sec on mobile cellular = 5. <3 sec = 4. >3 sec = 3 (probably needs image optimization).
2. Could a stranger landing cold understand the offer in 15 seconds? If yes = 5. If they have to scroll to find what you do = 3 — rewrite the hero.
3. Did the page reference all four artifacts (CLAUDE.md, skill, scheduled task, offer) AND link out to at least one external proof (your wiki, an existing case study)? If yes = 4-5. If only the offer is real = 3.

What this unlocks for your business

This is the asset that closes the loop. The offer (Activity 4) gets you the conversation; the portfolio (Activity 5) is the proof during the conversation; the install (Activities 1-3) is what you deliver after the conversation.

Without the public URL, every sales conversation has to re-prove your credibility from scratch. With the URL, the credibility is one click away — and the URL itself is what gets shared in DMs, in LinkedIn comments, and in cold-email replies.

This is also the asset that survives. Your skills will evolve, your scheduled tasks will be replaced, your offer will get sharper — but the URL stays at the same address, so every link out from every previous post still resolves to "the most current version of you."

Stuck? Use this prompt

If the deployed page looks broken but localhost was fine:

```
I deployed my portfolio to Vercel and it's live at {{URL}}, but
something looks wrong (describe what — page is blank / images
404 / styles missing).

Please:
1. Open {{URL}} via WebFetch.
2. Read the build logs in .vercel/output (if they exist locally).
3. Tell me the THREE most likely root causes.
4. For the most likely one, suggest a specific fix.
```

Final reflection

You installed an AI Operating System in 30 days. The install has four moving parts:

1. `CLAUDE.md` — alignment layer. Every Claude Code session in this project starts pre-aligned to your standards, vocabulary, and decision rules.
2. **A skill** — reusable capability. One repetitive judgment task that used to live only in your head now lives in a file Claude can invoke on demand.
3. **A scheduled task** — agency. The model produces an artifact when you're not watching, on a cadence you set.
4. **A productized offer** — sales asset. The thing you sell *because* of the install, with ROI math defensible against any prospect who asks.

The portfolio page (Activity 5) is the public surface that ties all four together.

The next 30 days, you do this again. Add a second skill. Schedule a second task. Sharpen the offer based on the first 1-2 conversations it generates. The install compounds.

The 30 days after that, you teach this to someone else — internally, or as a service. That's where the AIOS goes from a personal leverage tool to a business model.

If you want help installing this in your business, or installing it in your client's business as a productized service, that's something we do at Noxlee Automations. `noxle.chumperum@gmail.com`.

Now ship something else.

— Noxlee