From Discovery to Production

A hands-on workshop on building and shipping custom AI solutions for clients



Ibrahim Jaafar · v1.0.0

What we’ ll cover today



1. Discovery

Find the real problem and pick the right use case



2. Scope & sell

PoC vs MVP, proposals, 2-week sprints



3. Build

Standard stack, evals, AI-assisted coding



4. Ship

Deploy, secure, monitor



5. Grow

Economics and long-term partnerships

Who it’s for: freelancers, indie developers, and in-house teams turning AI prototypes into production systems.

Key terms

Six short terms you will see in this workshop. Come back to this slide any time.

ROI

Return on investment

What the client gains compared with what the project costs.

KPI

Key performance indicator

The one number that shows whether the project succeeded, e.g. 90% of fields correct.

LLM

Large language model

The AI model that reads and writes text, such as GPT or Claude.

CI/CD

Automatic testing and deployment

Every code change is checked and put live without manual steps.

DAG

Directed acyclic graph

A workflow drawn as steps that run in a fixed order and never loop back.

VPN

Virtual private network

A secure private connection, so only your team can reach the server.

Why so many AI projects never reach production

It's rarely the model.
It's the execution.

The average company scraps almost half of its AI proofs of concept (small test projects) before they reach production. Models are more capable every year, but doing it right at scale is still hard.

Source: S&P Global Market Intelligence, 2025



Wrong use case

Too big, unclear ROI, no way to measure success



Wrong scope

Overbuilding before the core metric works



Wrong expectations

Client expects 99% accuracy on day one

MODULE 1

Discovery & use-case selection

This is where you win or lose a project

Discovery: uncover the real problem

Before building anything, find the real problem behind the client's request.

Problem, not request

Find what the client actually needs, not what they think AI should do. Their own proposed use case is often too complex or too big.

EXAMPLE

Asked for "an AI chatbot", the real problem was staff answering the same questions by email all day.

Ask the ROI question

Custom software still costs real money, even when AI speeds up delivery. Be brutally honest: is there a real return?

EXAMPLE

Saving 10 people 2 hours a week each justifies a sprint. Saving one person an hour a month doesn't.

Quick wins over moonshots

Most organisations have low-hanging fruit they don't see. Shadow an employee for an afternoon — simple automations can have big impact.

EXAMPLE

Shadowing one employee showed daily copy-paste between two tools. A simple Zapier link fixed it, no AI needed.

Red flags to watch for

Warning signs during discovery that a project may fail. Address them before you commit.

“We want AI because everyone has it”

A fine way to open a conversation — but come back to: what are we automating, and what’s the ROI?

Example: *A retail CEO says: “Our rivals all have AI chatbots. Build us one by next month.” Nobody can say what it should do.*

No clear success criteria

If you can’t tell good from bad, you can’t track impact or course-correct.

Example: *A law firm wants AI to “speed up contract review”, but can’t say how long a review takes today or what a good one looks like.*

Expecting magic on day one

LLM systems won’t hit 100% at first — and for some tasks 80% isn’t good enough to ship.

Example: *A company deploys AI to read receipts, expecting 100% accuracy. At 90%, the errors create noise in its financial records.*

No data access

Without the right data, permissions and security access, you can’t build or deploy.

Example: *A clinic wants AI to summarise patient files, but legal hasn’t approved access, so you’d be building with no real data.*

Typical project types

Most custom AI projects fall into one of these five categories.



Document processing



Content generation



Customer support



Internal knowledge assistants



Data extraction

Tip: map every new lead to one of these. If it doesn't fit any, treat it as higher risk and scope it as a proof of concept (slide 15).

Use-case evaluation checklist

Ask these six questions about every use case before you commit to building it.

Simple & repeatable?

Can it really be automated? If not, scope it down: keep the valuable, doable part and drop or postpone the rest.

Clear success/fail rules?

Can outputs be checked against a standard?

Escalation: can errors go to a human?

Automate the 80% you can; route the rest.

Review: is a human in the loop?

A required review step adds control and lets you go live faster.

What if the AI is wrong?

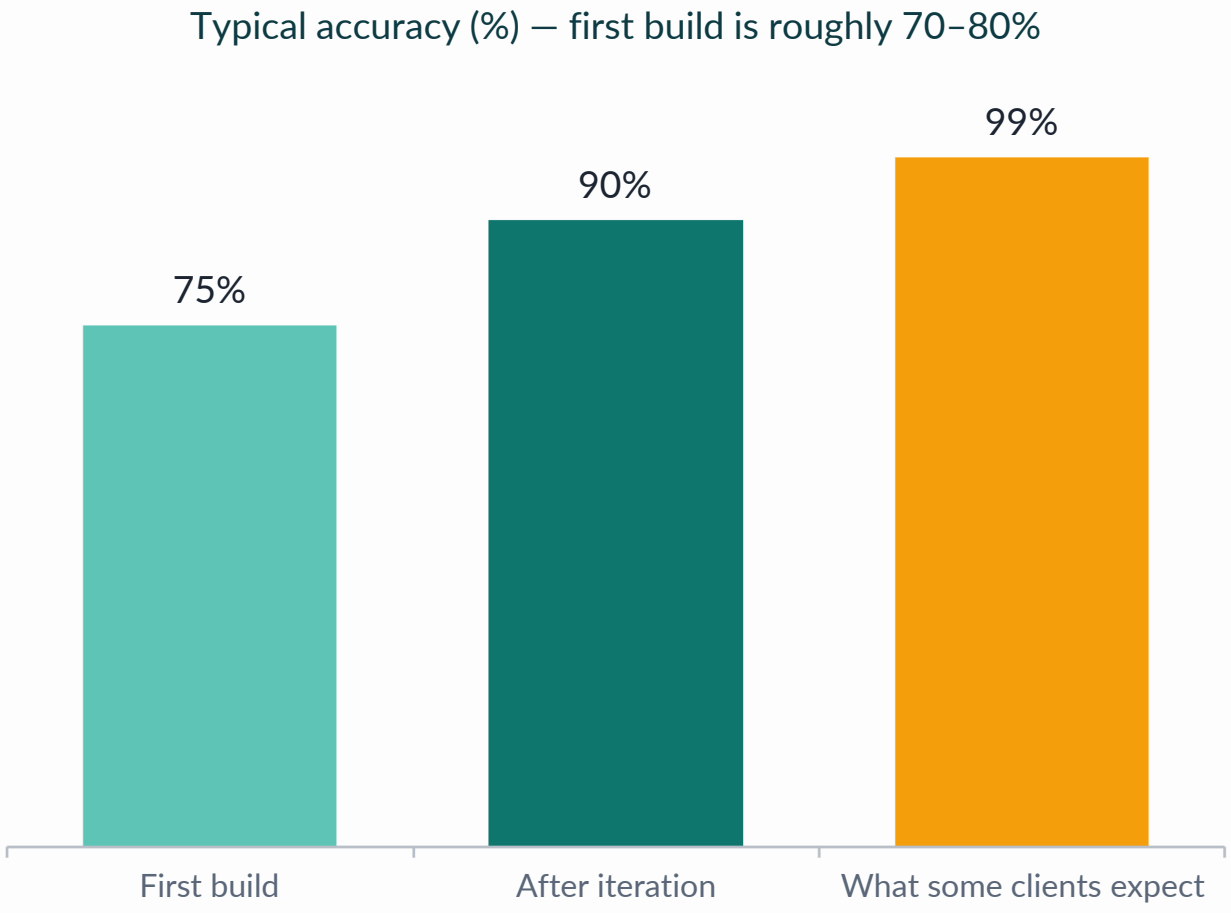
Will someone catch the mistake later, or does it reach customers directly? The bigger the harm, the more checks you need.

Data & access ready?

Permissions, security, and real data available.

The accuracy conversation

AI systems improve through iteration, so agree on realistic accuracy with the client from day one.



Frame it as iteration

"Here's v1, and here's our plan to improve it."

Example: *"v1 reads 80% of receipts correctly. In two weeks, we target 90%."*

Build feedback loops

Make it easy for the client to send feedback straight into development.

Example: *a "Report error" button that saves each wrong receipt for the next fix.*

Ask for their time

Domain experts must help judge what good looks like.

Example: *the finance lead reviews 20 results every Friday and marks the wrong ones.*

Exercise 1 — Evaluate a use case



HANDS-ON EXERCISE

In pairs, take a real or imagined client request and decide: build, scope down, or walk away?



20 minutes

1. Write the client's request in one sentence — then rewrite it as the underlying problem. (slide 6)
2. Estimate the ROI: time or money saved per month vs. likely build cost. (slide 6)
3. Score it against the 6-point checklist (repeatable, success rules, escalation, human in the loop, error impact, data). (slide 9)
4. Name one "quick win" that could be automated without AI at all. (slides 6, 8)
5. Decide: build, scope down, or walk away — and justify it in two lines. (slides 7, 9)

Exercise 1 — Example answer

A filled-in report for a sample client, a mid-size distribution company. Figures are illustrative.

CLIENT REQUEST	*"Build an AI to handle all our finance paperwork."*
1. Real problem (slide 6)	4 accountants spend about 2 hours a day typing receipt data by hand.
2. ROI estimate (slide 6)	Saves about 115 of 160 hours a month (≈ €3,450); the rest stays manual. Cost: e.g. €15k to build + about €200/month to run. Pays back in about 5 months.
3. 6-point checklist (slide 9)	Repeatable: yes · Clear rules: totals must match · Escalation: unclear receipts go to a person · Review: weekly · Error impact: high · Data: 2,000 samples
4. Quick win without AI (slides 6, 8)	A Zapier rule forwards all receipt emails into one shared folder, so nobody searches inboxes anymore.
5. Decision (slides 7, 9)	Scope down: start with restaurant and travel receipts (about 80% of volume), and add other types once accuracy passes 90%.

MODULE 2

Scoping, proposals & sprints

Decide what it will do — and what it won't

Scoping: avoid the classic traps

Agree on what the system will and won't do before any code is written.

Common traps

- Scope creep: "Can it also do X?" early on
- Underestimating how unpredictable LLMs are at scale
- Edge cases you didn't plan for
- Overbuilding v1 before the core KPI works

What to do instead

1. Start with one core workflow
2. Define the input and the expected output
3. Split must-haves from nice-to-haves
4. Identify technical risks early
5. Set clear acceptance criteria and the KPI you optimise for

Proof of concept vs. MVP

Choose the right first step: prove it can work, or ship the smallest useful version.

Proof of concept

When: you're sure you can build something, but not sure current AI will hit the client's KPIs.

- Local demo or script: input → system → output
- Both sides agree whether output is good enough
- If not: gap analysis on how to get there
- Deliverable: demo + report — not live value

Minimum viable product (MVP)

When: you've built something similar before, or the solution is simpler.

- Real, tangible, deployed solution
- The smallest thing that produces value
- Delivered, handed over, running somewhere
- Deliverable: working software

What goes into the proposal

A standard proposal keeps scope, success criteria and costs clear for both sides.

1. Problem statement — in the client's own words
2. Proposed solution and scope boundaries
3. Technical approach — architecture, features (depth depends on complexity)
4. What's included — and what's not
5. Success criteria and how they're measured
6. Timeline, pricing — and ongoing costs (hosting, infra, LLM APIs)



Don't forget ongoing costs

A cheap chatbot is easy to sell. But who hosts it, what does the infrastructure cost, and what will the LLM API bill be each month? Give a rough estimate upfront.

The two-week sprint

Sell and deliver the work in two-week blocks: plan, build, show the result, then agree on the next block.

Days 1–2

Set up

You: Fork the standard template and set up the project, database and access to the client's data.

Client sees: A kickoff call on Monday to agree on goals and how you'll communicate.

Days 3–8

Build

You: Build the core workflow, connect the AI model, test it on real examples and improve accuracy every day.

Client sees: Short written updates (async), with no daily meetings.

Day 9

Test & polish

You: Fix bugs, polish, and test on new examples the system hasn't seen.

Client sees: Nothing new yet. This day makes sure the demo goes smoothly.

Day 10

Demo

You: Meet the client and show them what you built.

Client sees: What works today, then agrees with you on the next sprint.

Price

A fixed price per sprint, set by the project's size, complexity and your market.

Why sprints

No big 10-week quote up front. Ask "what can we prove or ship in two weeks?"

After each sprint

The client tests, gives feedback and decides whether to book the next sprint.

Reality check: the gap between sprints

Clients often pause between sprints, so plan your pipeline around the gaps.

The plan



What usually happens



Why it happens

Clients get busy, test internally, go on holiday and re-prioritise. Bigger companies simply move slower than AI-speed delivery.

How to handle it

Keep a second project lined up. If both clients are active at once, AI coding tools let one developer do the work of several, and a freelancer can cover what's left.

Exercise 2 — Draft a one-page proposal



HANDS-ON EXERCISE

Using your Exercise 1 use case, draft a one-page proposal and a two-week sprint plan.

25 minutes

1. Write the problem statement in the client's words. (slide 16)
2. Decide PoC or MVP — and write one line the client will read about what they will (and won't) get. (slide 15)
3. List 3 must-haves, 3 nice-to-haves, and the single KPI that decides success. (slide 14)
4. Sketch the sprint: days 1–2, 3–8, 9–10. (slide 17)
5. Estimate the sprint price and the monthly ongoing costs (hosting + LLM API). (slides 16, 17)

Exercise 2 — Example answer

A filled-in report for the same sample client as Exercise 1. Figures are illustrative.

PROJECT	*Receipt reading for a mid-size distribution company (the Exercise 1 client).*
1. Problem statement (slide 16)	"Our accountants spend about 2 hours a day typing receipt data by hand, and mistakes slip into our books."
2. PoC or MVP (slide 15)	**MVP**, since similar systems have worked before. Client line: "You'll get a working receipt reader for restaurant and travel receipts. Other types are not included yet."
3. Scope and KPI (slide 14)	**Must:** read date, vendor and total; flag unclear receipts; export to the accounting system. **Nice:** VAT split, mobile upload. **KPI:** 90% of fields correct.
4. Sprint plan (slide 17)	Days 1–2: template + sample receipts · Days 3–8: extraction workflow, daily accuracy checks · Day 9: test on new receipts · Day 10: demo with the finance lead
5. Price and running costs (slides 16, 17)	Sprint: your usual fixed price for two weeks. Running costs: hosting + AI usage per receipt, estimated at about €200/month.

MODULE 3

Build: standard stack, evals & AI-assisted coding

Same stack, same patterns, every single project

The tech stack

One proven stack, reused on every project, instead of new tools for each client.

Front end (when needed)	Next.js + component library, Supabase auth
Backend API	Python · FastAPI · Pydantic AI · step-by-step workflows (DAGs), with loops only where needed
Background jobs	Celery workers with Redis as the task queue
Data	PostgreSQL (self-hosted Supabase) + vector search
LLM provider	Chosen to fit the client's data and compliance rules (e.g. Azure OpenAI)
Packaging	Docker Compose — containerised from day one

Key takeaway

Nothing here is exotic. The edge comes from using the same stack, the same patterns and the same deployment process on every project.

Standardisation is the secret sauce

Starting every project from the same template is what makes speed and quality possible.



Fork one template

Every project starts from the same repo — Docker, database, auth and search already wired up.



Same folders, same files

Jump into any project and instantly know where everything lives. Essential when running several at once.



Keeps AI agents on rails

A fixed foundation stops coding agents from inventing their own architecture — and makes bad PRs easy to spot.

AI tools change every month. Your process, core stack and patterns shouldn’ t.

Testing & evaluation in three layers

Three kinds of tests, so you know when something breaks or gets worse.

1. Unit tests

Does each small piece work?

Test one function on its own.

Example: *Does the dollars-to-euros function return the right number?*

2. Integration tests

Do the pieces work together?

Test the whole flow from start to end.

Example: *Receipt in → read → saved → sent to accounting, without breaking.*

3. LLM evals

Does the AI give the right answers?

Grade the AI like an exam, on examples where you know the answer.

Example: *45 of 50 receipts read correctly = 90%.*

How to run LLM evals — rerun them after every prompt or AI model change

1. Use a fixed form

Make the AI fill in fields (date, vendor, total…), not free text, so answers can be checked automatically.

2. Collect test examples

Start with made-up examples, then switch to real data from the client.

3. Write the right answer

For each example, note what correct looks like: “this receipt’s total is €48.20”.

4. Add every reported mistake

When the client finds an error, turn it into a new test so it never comes back.

AI-assisted development workflow

How to use AI coding agents to ship faster while staying in control.

5–10×

more output per developer than three years ago: what took two months now ships in two weeks

Stay a small team: AI tools let each person do more, so you don't need to hire to grow.

Examples on this slide use Claude Code. Other AI coding tools have similar features.

Keep it simple	Use the AI's default settings at first and add tools later. Run only 1–3 sessions at once so you can check their work.
Plan first, then execute	Use plan mode before letting the agent write code.
Mind the context window	Every session starts fresh. Keep CLAUDE.md files (also in subfolders) and a docs folder up to date.
Prime each session	Point it to the exact docs and files — otherwise it burns context re-indexing the codebase.
Turn repeats into skills	Done something 2–3 times? Package it as a reusable skill.

Exercise 3 — Plan the build



HANDS-ON EXERCISE

Using your Exercise 1 use case, plan how you would build it on a standard stack.

20 minutes

1. List the stack layers your project needs (front end, API, background jobs, data, LLM provider, packaging) — and which you can skip. (slide 22)
2. Sketch the folder structure of a starter template you would reuse on every project. (slide 23)
3. Pick one workflow step and define its structured output: which fields does a "correct" answer contain? (slide 24)
4. Write 3 eval cases for that step: a known input and the output you expect. (slide 24)
5. Write the opening message for a fresh AI coding session: which docs and files it should read first. (slide 25)

Exercise 3 — Example answer

A filled-in report for the same sample client as Exercise 1. Figures are illustrative.

PROJECT	*Receipt reading for a mid-size distribution company (the Exercise 1 client).*
1. Stack layers (slide 22)	**Need:** API (FastAPI), background jobs (Celery + Redis), database (Postgres), LLM provider, Docker. **Skip:** front end, because results go straight into the accounting system.
2. Template folders (slide 23)	app/api · app/workflows · app/prompts · app/models · tests/evals · docs · docker
3. Structured output (slide 24)	ReceiptResult: date, vendor, total, currency, confidence, needs_review (yes/no)
4. Three eval cases (slide 24)	Clear restaurant receipt → total = €48.20 · Faded taxi receipt → needs_review = yes · Hotel bill in dollars → currency = USD
5. Session opening message (slide 25)	"Read docs/overview.md and app/workflows/receipts.py first. Then add VAT extraction to ReceiptResult."

MODULE 4

Ship: deploy, secure, monitor

Know what your AI is doing in production

Deployment pipeline

A simple, cheap and repeatable path from your laptop to a live HTTPS app.



One rented server

Rent one plain server (e.g. from Hetzner) and run everything on it yourself, with no extra cloud services. Simple, reliable and about 10× cheaper than Azure or AWS.

Pull-based CI/CD

A GitHub Action triggers it, checks for leaked secrets and vulnerable libraries, then a script on the server pulls the new version.

Client's own cloud?

If the client wants it in their own Azure, AWS or Google Cloud, use the same steps: their servers work just like yours.

Security baseline for every deployment

Assume everything is exposed by default, then open only what is needed.



Default = exposed

This is the minimum, not the whole story.

Block all IPs by default

Then whitelist only the front ends and integrations that need access.

Team VPN with a static IP

Whitelist it so the production database is reachable only from your team.

Firewall on the server

Only ports 80 and 443 open; everything else stays behind it.

Private internal networks

Services talk to each other privately where possible.

Security beyond the server

Five more habits that protect your client's system and data, on top of the server basics.

Keep secrets out of the code	Store API keys and passwords in environment variables or a secrets manager, never in the code or on GitHub.
Control who gets in	Require a login for the app and its API, and give each person only the access they need.
Keep everything updated	Regularly update the server, Docker images and libraries to close known security holes.
Back up, and test the restore	Back up the database automatically every day, use immutable backups (they can't be changed or deleted) when possible, and practise restoring them.
Protect data sent to the AI	Send personal or sensitive data to the AI model only with the client's agreement, and watch for hidden instructions in documents (prompt injection).

Monitoring & observability

See what your AI does in production, so you can find and fix problems fast.

LLM tracing (e.g. Langfuse)

- Traces every LLM call and workflow step automatically
- Drill down to the exact step, prompt and context
- Use it to optimise prompts and debug client reports

Error tracking (e.g. Sentry)

- Catches app errors around your API endpoints
- Alerts to a Slack channel so you can react fast
- Copy the error as markdown → hand it to your coding agent

Why it matters: async background jobs fail silently — the server stays up, the event just never gets processed. Without monitoring you're flying blind.

Exercise 4 — Make it production-ready



HANDS-ON EXERCISE

Take your Exercise 3 build and plan how it goes live safely.



15 minutes

1. Draw the deployment path: local → GitHub → CI/CD → server → a URL with HTTPS. (slide 29)
2. List which ports will be open, which IPs or services you will whitelist, and what stays on a private internal network. (slide 30)
3. Decide where secrets (API keys, passwords) are stored and how often the database is backed up. (slide 31)
4. Name three things you will trace for the LLM steps and two errors you want an alert for. (slide 32)
5. Write down where alerts go and who reacts to them. (slide 32)

Exercise 4 — Example answer

A filled-in report for the same sample client as Exercise 1. Figures are illustrative.

PROJECT	*Receipt reading for a mid-size distribution company (the Exercise 1 client).*
1. Deployment path (slide 29)	Laptop → GitHub → GitHub Action → server pulls and runs Docker → receipts.client.com with HTTPS (Caddy)
2. Ports and private network (slide 30)	Open: ports 80 and 443 only. Whitelisted: the accounting system's IP and the team VPN's fixed IP. The database and Redis stay internal.
3. Secrets and backups (slide 31)	API keys and passwords live in environment variables on the server, never on GitHub. The database is backed up every night, and a restore is tested once a month.
4. What to trace and alert (slide 32)	**Trace:** each receipt's input, prompt, output, confidence and cost. **Alert:** a failed export to accounting, or a crashed workflow.
5. Who reacts to alerts (slide 32)	Alerts go to a #receipts-alerts Slack channel. The developer on duty responds the same day.

MODULE 5

Economics & the long game

Where the real money in software is made

The changing economics of AI-assisted development

AI makes delivery faster. Use that to deliver more value, not to charge less.

Don’t

- Cut your price just because AI made the work faster. The client still gets the same value.
- Charge only by the hour. If AI does 5 hours of work in 10 minutes, you earn less for the same result. Thus you'd punish yourself for efficiency. That’s why we price per sprint (slide 17).

Do

- Keep your price where clients already expect it. The value of the result hasn’t changed.
- Use the time you save to take on more projects at once, instead of earning less per project.
- Over-deliver: say yes to small extras like “can it also do X?”, which now take minutes.



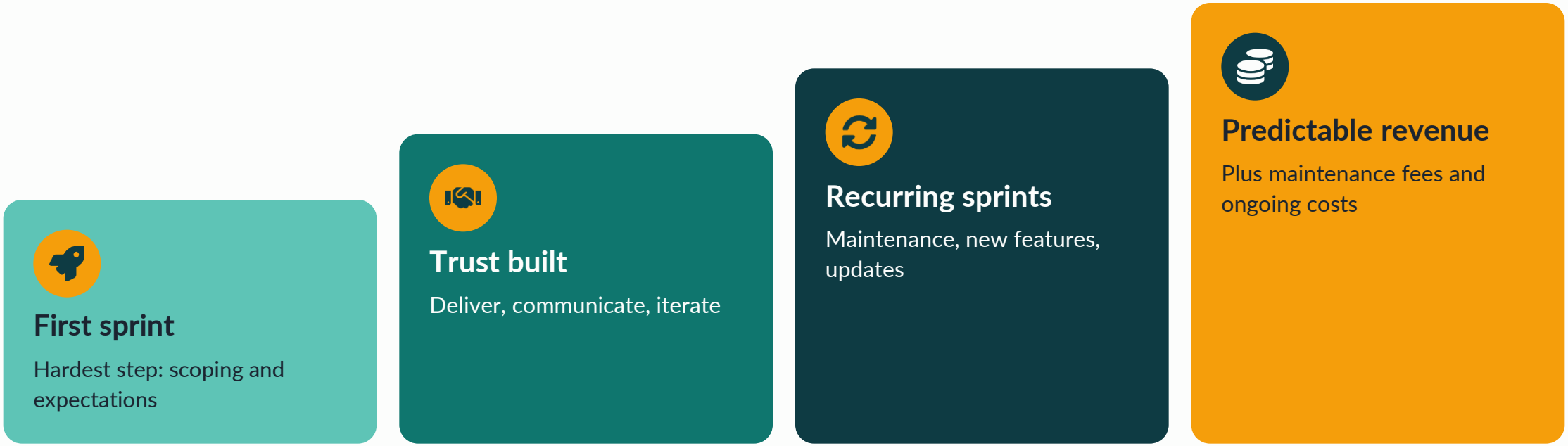
A window, not forever

Right now clients still pay normal prices for faster work. Competitors will catch up in a year or two, so use this time to win long-term clients.

The long game: become their AI partner

The real value comes from becoming a client's long-term AI partner, not one-off projects.

You don't need many clients — one big, long-term client can be a full business.



Exercise 5 — Price it and plan the partnership



HANDS-ON EXERCISE

Turn your project into a fairly priced first sprint and a long-term relationship.



15 minutes

1. Price your two-week sprint where clients already expect it — no race to the bottom. (slide 36)
2. List two small extras you could over-deliver thanks to AI speed. (slide 36)
3. Map how this first project could turn into recurring work: maintenance, new features, updates. (slide 37)
4. Estimate the monthly maintenance fee and the ongoing costs you would bill. (slide 37)
5. List three reasons this client would keep working with you after the first project. (slide 37)

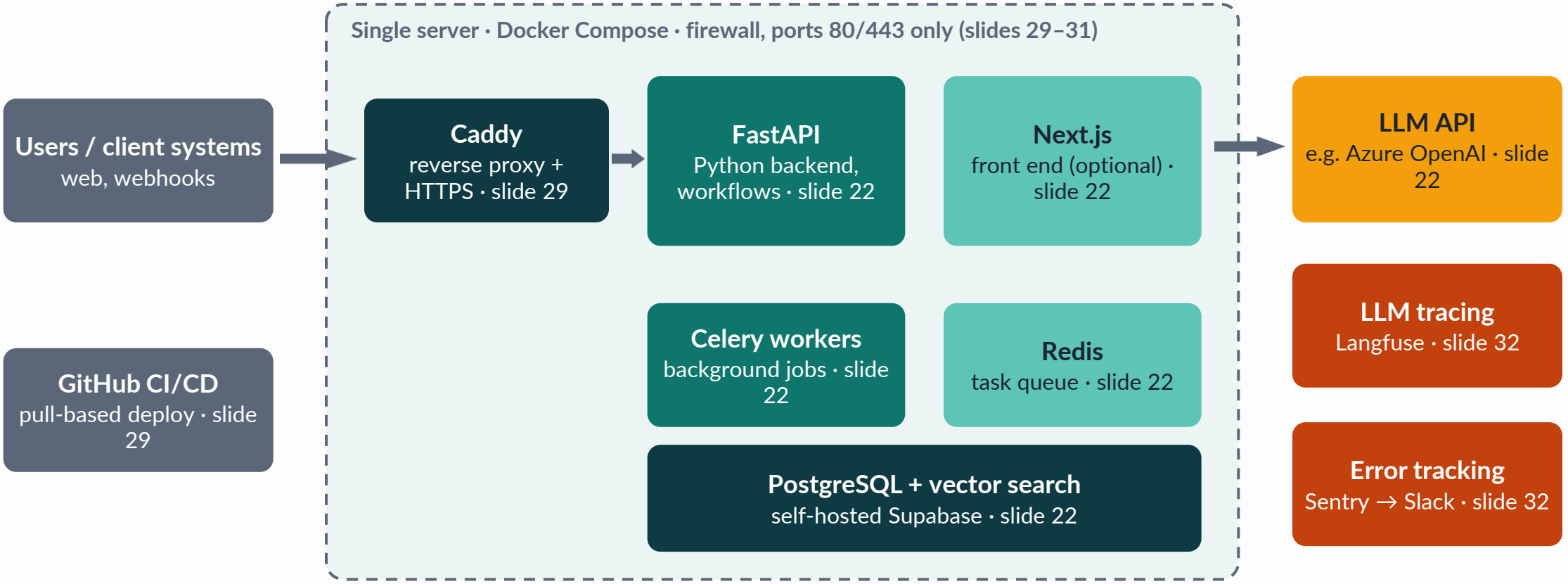
Exercise 5 — Example answer

A filled-in report for the same sample client as Exercise 1. Figures are illustrative.

PROJECT	*Receipt reading for a mid-size distribution company (the Exercise 1 client).*
1. Sprint price (slide 36)	Keep your usual market price for a two-week sprint, even though AI makes the build faster.
2. Extras to over-deliver (slide 36)	A VAT split per receipt, and a weekly email listing the receipts that need review.
3. Path to recurring work (slide 37)	Next: supplier invoices, then contracts. Plus monthly accuracy reviews and updates when AI models change.
4. Maintenance and costs (slide 37)	A fixed monthly maintenance fee, with hosting and AI usage billed on top at cost.
5. Reasons to stay (slide 37)	Fast fixes when something breaks, a system built around their own processes, and new AI ideas brought to them each quarter.

Recap: the whole system at a glance

Everything from Modules 3 and 4 in one picture, on a single server. Each box shows the slide where it was covered.



Key takeaways

Discover the real problem	Push for ROI and quick wins; walk away from red flags.
Scope tight, frame as iteration	PoC vs MVP, one core workflow, one KPI, two-week sprints.
Standardise everything	One template, one stack, one deploy process — AI stays on rails.
Test, secure, monitor	Evals with real data, locked-down servers, traces + error alerts.
Play the long game	Don’t undercut on price; become the long-term partner.

Exercise 6 — Your 30-day action plan



HANDS-ON EXERCISE

Turn today into a concrete plan for your next (or current) AI project.

15 minutes + share-out

1. Pick one real client or internal use case and run it through the discovery checklist. (slides 6–9)
2. Choose your standard stack and create a starter template repo you'll fork every time. (slides 22, 23)
3. Write a reusable proposal template with the six sections — including ongoing costs. (slide 16)
4. Set up the minimum: evals, a firewall + IP whitelist, LLM tracing and error alerts. (slides 24, 30, 32)
5. Share your plan with a partner and agree on one thing you'll ship in the next two weeks. (slide 17)

Exercise 6 — Example answer

A filled-in report for the same sample client as Exercise 1. Figures are illustrative.

PROJECT	*Receipt reading for a mid-size distribution company (the Exercise 1 client).*
1. Use case and checklist (slides 6–9)	Receipt reading passes 5 of the 6 checks. Error impact is high, so a weekly human review is added.
2. Stack and template (slides 22, 23)	Python + FastAPI + Celery + Postgres + Docker, saved as a starter repo to fork for every new project.
3. Proposal template (slide 16)	A one-page document with the six sections, ready to fill in after each discovery call.
4. Minimum setup (slides 24, 30, 32)	10 eval cases, a firewall with a VPN whitelist, LLM tracing and Slack error alerts.
5. Two-week commitment (slide 17)	Demo the receipt-reading MVP to the finance lead by [date].

Thank you — questions?

AI changes every month. Your process shouldn’t.

