certificates.dev

FOR DEVELOPERS WORKING WITH AI

Prove Your AI Skills

A practical guide to proving you can build with AI.

INCLUDES A PRACTICAL
8-DAY PLAN

INSIDE THIS BOOK

AI DEVELOPER CERTIFICATION

ai.certificates.dev

Contents

Welcome

Showing someone you're good with AI is harder than it sounds. This book gives you a practical way to do it, using work you've already done.

The idea is pretty simple. Through the book, we'll take any AI assisted work you've shipped and explain the decisions behind it in multiple ways, gradually ramping it up and leaving behind a trail of evidence to help prove you're able to steer AI.

From there, we'll bring your best examples into your profile, CV and interview answers, so everything you claim points to something a real person can check. The 8-Day Plan at the end turns all of this into eight short tasks you can follow and check off, each with a clear result.

Oh, and if you're still building your AI skills, that's absolutely fine. Use this while you're learning.

I hope this helps you get noticed for the right reasons.

Alex GS
TECHNICAL EDUCATION LEAD

certificates.dev

How to use this book

Pick an AI-assisted change you've already finished. It could be a feature, a bug fix or a bit of maintenance. Anything you can explain works, and you can bring in more examples as you go.

BRING FINISHED WORK

Pick something you can explain

You don't need a full app. A small change with one decision worth explaining is plenty.

- You understand it well
- Allowed to share publicly

You don't need to build a full portfolio from scratch. To make the process easier, you can explain the decisions you're already making, adding a little more context each time you share one with a wider audience.

HERE'S THE BASIS OF GRADUAL SHARING WE'LL USE THROUGHOUT THE BOOK, ALL STEMMING FROM A DECISION OR PROBLEM YOU'VE MADE WHILE BUILDING WITH AI.

01	02	03	04
Keep it close to the work	**Show one person**	**Publish the lesson**	**Make it reusable**
e.g. write the decision into the PR description.	e.g. give a teammate a quick demo.	e.g. publish for anyone to read/watch.	e.g. a skill or checklist someone else can run.

Prove

AI can write code, but you still decide what to keep, what to change and how to check it. That’s the part that’s hardest to prove.

Grab the change you picked earlier and let’s get going.

Code doesn’t prove much anymore

If AI helped with your last change, chances are you made a load of decisions along the way. You planned (even in your head) before generating any code, threw away code, and picked one result over another for a good reason.

You don’t need to build anything new right now. The easiest way to prove yourself is to start by explaining your decisions where the work already lives.

Once an example has helped your team, you can reuse it. The same decision can become a blog post, a small tool or an interview answer, each one explaining the same work at a different level of detail.

And even better, you can repeat this process and build up a suite of decisions that prove you can work with AI.

So, let’s start with a decision you can explain.

Give people something they can check

Saying “I’m good with AI” doesn’t give anyone much to go on. A real example does. The problem, the decision you made, how you checked it, and what happened next.

PROBLEM → DECISION → CHECK → RESULT

01 · PROBLEM

Start with the problem

Explain what needed to change, who it affected and the goal.

02 · DECISION

Record the decision

Say what you kept, changed or rejected from the agent’s code, and why.

03 · CHECK

Show how you checked it

Point to the test, review or measurement that you used to verify.

04 · RESULT

Link the evidence

Link the PR, the tests or the demo so someone can check it.

Start where the work already lives

It's tempting to jump straight into building a polished public project to prove yourself.

Even easier though? Your latest pull request already has the problem, the code and (hopefully) the tests. A PR description can reveal a lot.

Nobody needs to see every prompt. They need the handful of decisions a future maintainer would otherwise have to work out for themselves.

So let's start with two small places: the PR description and the review.

Explain your decisions in every PR

The agent's draft vs yours

On the left is a typical agent-written description. On the right is a template you can copy: the problem, your decision and how you checked it.

WEAK EXAMPLE · AGENT DRAFT

```
# What changed

Adds the requested change.

- Generated implementation
- Added tests

Test plan:
- All tests pass
```

USEFUL EXAMPLE · YOUR EXPLANATION

```
# Why this exists

[Name the problem and the
constraint that shaped the work.]

# Main decision

[Explain what you chose and why.]
We considered [other option], but
rejected it because [reason].

# Checked

- [important behaviour]
- [failure or edge case]
- [how someone can reproduce the
result]
```

Tip: save yours as `.GITHUB/PULL_REQUEST_TEMPLATE.MD` and GitHub will pre-fill every new PR with it.

Save your best review catches

When you catch a real problem and post a comment, that's proof of your judgement. Keep a short note of it, like the example below. We'll shape these notes into proof later in the book.

- PS: this works even if you're reviewing your own work!

01 · THE REVIEW COMMENT

Y **you** commented on checkout.ts

This checkout code charges the card, creates the order and sends the email in one go. If it fails halfway, a retry could charge the customer twice. Could we split the steps up and record what's already run, then add a test that proves a retry is safe?

02 · WHAT CHANGED

We asked the agent to split the steps up and add a test that proves the customer is only charged once. Then we added the retry rule to our agent instructions, so it doesn't write code like this again.

03 · THE NOTE YOU KEEP · COPY THIS SHAPE

```
Problem: the agent's checkout code charges the card, creates the order and
sends the email in one go.
Risk: a retry after a partial failure could charge the customer twice.
Decision: had the agent split the steps and record each one as it completes.
Evidence: the retry test fails on the old code and passes on the new.
Reuse: added the retry rule to our agent instructions and review checklist.
```

Keep a few of these and you'll never be stuck for an example.

Share it with one person first

You don’t need a public post to start.

Start with one teammate and walk them through what you’ve spotted as you would in code review: the problem, the first result, the correction and how you verify.

If they ask about something you thought you’d covered, that’s useful. Improve the explanation while the example is still fresh, and your public version will be stronger for it.

Going public then becomes an editing job rather than a writing job. You’re cutting the private context and keeping the lesson, not inventing a new story.

Let’s make the smallest version first: a three-minute demo.

A useful demo doesn't need to be perfect

A three-minute recording is enough, and one take is fine. Just make sure it includes the first agent result, the correction that followed and all the important details.

Here's a rough timeline:

0:00

Set up the problem

Say what you were building and why.

0:30

Show the first result

Show what your agent gave you first.

1:15

Show one correction

Show what was wrong and how you fixed it.

2:30

Finish with the check

Run a test and show it passing.

RECORDING

Any screen recorder works

QuickTime, Loom or Screen Studio. Use sample data, and watch it back before sharing.

SHARING

Make it linkable

Upload it to Loom or unlisted YouTube. You'll be linking to it from your proof later.

Turn one decision into a useful post

You don’t need a big opinion about AI every week. A small engineering decision is far more useful.

01 · HOOK	02 · STORY	03 · FIX AND PROOF	04 · TAKEAWAY
One concrete line that makes a developer stop scrolling.	What the AI did and what you spotted. Keep the lines short.	What you changed, and the test that proves it.	The one thing others can copy, then a link to the write-up.

You
@you

AI nearly charged our customers twice today.

The code looked good, but I spotted that a retry after a partial failure would run the charge again.

I had the agent split the steps and add a test proving one charge only. The rule is in our agent instructions now, so it won’t happen again.

Full write-up! **[link]**

Go a step more public each time

You can move any example up the ladder: a PR, your team, a public write-up, then a talk. Each step can reuse what you already have.

SHARE IT MORE WIDELY AT EACH STEP →

01 · IN THE WORK

Review comment

Explain the decision in the PR.

02 · WITH THE TEAM

Demo or recording

Walk your team through it and collect the questions.

03 · IN PUBLIC

Case study or guide

Publish a version anyone can read.

04 · WIDER AUDIENCE

Talk or workshop

Teach it live and use the questions to improve it.

Not every example needs to reach the top. Move one up a step when the next audience would genuinely find it useful.

THE 8-DAY PLAN FOLLOWS THIS ROUTE

Day 1 improves the PR, Day 2 shares it with your team, Days 3–4 publish the lesson, Days 5–7 get it into someone else's hands, and Day 8 links everything in one place.

Put it where people can find it

Social posts are great for reaching people, but they disappear quickly. Keep the full example somewhere you control, then let each post point back to it.

GITHUB

Runnable work

For code someone can clone and run for themselves.

PERSONAL SITE

Full case study

The full post, on a page you control.

LINKEDIN

Professional context

Why the work mattered, with a link.

X, ETC.

Small technical lesson

One quick lesson, linking back to the full example.

YOU DON’T NEED ALL FOUR

Start with the platform you already use. And if you don’t have a personal site, a public repo with a good README makes a perfectly good home for the full example until you do.

A public post is only the start

So far you've explained your work in words. The next step is turning what you've learnt into something another developer can actually run.

Keep an eye out for the decision or fix you keep repeating. Chances are it could become a checklist, a template, a skill or a small tool, and the best way to find out whether it works is to let someone else try it.

So, let's look at a few things you could build.

Make something another developer can use

Keep the first version small enough for someone to quickly try. Where they get stuck tells you exactly what to fix next.

A skill

e.g. a skill that reviews code against your team's standards.

01

A template

e.g. the PR description your team fills in.

02

A checklist

e.g. the checks you run before trusting an agent's done.

03

A small integration

e.g. an MCP tool that lets an agent safely query your staging database.

04

Pick something specific to be known for

"Developer who uses AI" could describe almost anyone. Go narrower: a problem you keep solving, or an area you're genuinely interested in. Then publish a few useful examples around it.

BEFORE

Too broad

"AI full-stack developer."

AFTER

Specific shape

"[Speciality] developer helping [audience] solve [repeatable problem]."

01

Pick your lane

A problem you keep solving, or something you're good at and enjoy. Both work.

02

Say who it helps

A team, a framework community or a type of product.

03

Publish three examples

Get a few real examples out first, then update your profiles around them.

A lane could be "safe AI-assisted data imports", "refactoring legacy code with AI" or "making AI-generated UI accessible". Your lane can change over time.

Point to the proof

People will look you up on LinkedIn, GitHub or your own site. Each one has the same job: one line about what you do, then links to two or three real examples they can check.

EXAMPLE PROFILE

[Speciality] developer helping [audience] solve [specific problem]

Sharing case studies, checklists and small tools from [your lane].

FEATURED 01

Case study

e.g. refactoring a 2,000-line legacy class with AI safely.

FEATURED 02

Reusable checklist

e.g. the checks I run before merging an AI refactor.

FEATURED 03

Short demo

e.g. an AI refactor going wrong, and how I fixed it.

COMPLETED EXAMPLE

I'm a backend developer who refactors legacy code with AI. I write about preparing old code for agents, the tests I add before it touches anything, and the checks I implement before shipping.

01

LinkedIn

Use the headline, About section and Featured links to tell the same story.

02

GitHub

Pin the repos that matter and explain your part in each README.

03

Personal site

Keep the full case studies here, linking back to the code or the public PR.

Turn your proof into better answers

Chances are you'll be asked in an interview how you actually use AI. Let's write a CV line you can back up, then break the full answer into five parts.

WEAK CV LINE

```
Used AI to build backend features
and improve developer
productivity.
```

CV LINE WITH EVIDENCE

```
Reviewed an AI-assisted [change],
added checks for [failure or edge
case], and turned what I learnt
into an open-source skill now used
by [team].
```

01

The problem

Start with what you were building and why it mattered.

02

AI's first attempt

Describe what AI gave you.

03

What you changed

Explain what you kept, fixed or rejected, and why.

04

How you checked it

Say how you proved it worked.

05

What you kept

Finish with what you reused or improved for next time.

Check what the proof actually shows

Likes are nice, but they don't tell you whether an example actually proves anything. Run each of yours through these five questions.

01 **Can someone check it?**
There's a link, a diff, a test or a recording to look at.

02 **Is it specific?**
It names the problem and the decision you made.

03 **Is your part clear?**
It's obvious what you did and what the AI did.

04 **Did you prove it worked?**
People can see the tests passing or the feature working.

05 **Can someone else use it?**
They can learn from it or run it themselves.

KEEP A SIMPLE DECISION LOG

```
Date   | Decision                                | Proof
12 Mar | Caught [something unsafe] in [feature] | PR #142
28 Mar | Refactored the import job with AI      | [blog post link]
09 Apr | Checklist v0.2                         | [repo link]
```

The 8-Day Plan

Everything from the book as a practical plan. By the end, you'll have a trail of proof to point to.

Think of these as eight focused sessions rather than eight days in a row. Feel free to break them up.

From a PR to a public post

You'll take one example and share it a little wider each day: a PR description, a team demo, a blog post, then a social post.

Use: the proof trail on p. 7, PR and review shapes on pp. 9–10, and sharing route on pp. 11–15.

- [ ] DAY 1

Add the "why" to a PR description

Open a PR you own, or one you merged recently. Replace the empty or generated description with three short sections: why the change exists, the decision you made and how you know it works.

END RESULT: **an updated PR description.**

- [ ] DAY 2

Record and share a real AI-assisted demo

Record a few minutes of real AI-assisted work, or demo it live to your team. Show the first result, one correction and the final check, then ask what wasn't clear.

END RESULT: **a recording shared with your team.**

- [ ] DAY 3

Write up what happened in a blog

Turn the PR or demo into a post: the problem, AI's first result, what you changed and the evidence.

END RESULT: **a published blog post.**

- [ ] DAY 4

Post the lesson on X, LinkedIn, etc.

Sum up the lesson in a few lines and link back to the blog post. Add a screenshot or clip if it helps.

END RESULT: **a social post linking to your blog post.**

From a post to something people use

Build something small from the lesson, watch one developer use it, log what you learnt, then add it all to your profile.

Use: the reusable work guide on p. 17, profile route on pp. 18–19 and proof audit on p. 21.

- [ ] DAY 5

Publish one checklist, template or skill

Take a reusable part of your findings and turn it into something that saves other people (or future you) time. Add when to use it, how to use it and a worked example, then publish it.

END RESULT: **a public v1 someone else could use.**

- [ ] DAY 6

Give it to a teammate to run

Send v1 to one developer who has the same problem. Ask them to use it without your help, and note down where they got stuck. It's also worth posting it on your socials, too.

END RESULT: **real feedback from one developer.**

- [ ] DAY 7

Start a decision log

Look back over the week (or longer) and log each decision, with a link to its proof (the log shape on p. 21 works). Future interview answers and blog posts get much easier when you've written things down as they happened.

END RESULT: **a decision log with its first entries.**

- [ ] DAY 8

Update the profile people actually visit

Pick one place people look you up (your site, GitHub or LinkedIn) and link everything from the week.

END RESULT: **a profile that links to your proof.**

After eight days

When everything on this list exists, you've done something most developers never get round to: put real proof behind your skills.

- [ ] WORK **An updated PR**
  It explains the why, the decision and how you know it works.
- [ ] TEAM **A shared demo**
  A real person has seen how you work with AI.
- [ ] PUBLIC **A blog post and social post**
  The lesson, published where anyone can read it.
- [ ] REUSE **An improved tool**
  Improved because someone else used it.
- [ ] FIND **An updated profile**
  Everything above, linked in one place.

DON'T FORCE ANOTHER EXAMPLE

Keep this page up to date, and have another go at the plan when you have a genuinely different decision or result to prove. Rinse and repeat.

Where to Go Next

The work you've shared already says a lot. A certification adds an independent check on top, tied to you.

You've built the proof. The last page is about getting it stamped.

certificates.dev

Where a certification can help

A certification doesn't replace anything you've built in this book. But it adds an independent check of your skills for when an employer, client or team wants one.

AI DEVELOPER CERTIFICATION

Request early access

We're building the certification around practical AI-assisted development. It tests how you direct, review and improve AI output, and proves you're the engineer AI was built to amplify.

- BUILT BY INDUSTRY EXPERTS 01
- PANEL REVIEWED 02
- PROCTORED ASSESSMENT 03

SCAN TO REQUEST EARLY ACCESS

JOIN THE WAITLIST FOR EARLY ACCESS

ai.certificates.dev ↗