Your Path Into Tech

Explore the Field, Build Real Skills, Take Your First Steps

Chris Ayers

Principal Software Engineer | Speaker | Community Builder

Get the slides

Scan the code, or open:

chris-ayers.com/tech-career-toolkit

Follow along now, and keep every link, tool, and resource from this talk.

Chris Ayers

Principal Software Engineer
Azure EngOps AzRel
Microsoft

- BlueSky: @chris-ayers.com
- LinkedIn: chris-l-ayers
- Blog: https://chris-ayers.com/
- GitHub: Codebytes
- Mastodon: @Chrisayers@hachyderm.io
- ~~Twitter: @Chris_L_Ayers~~

Start with the honest version

No single route

Classes, clubs, online courses, camps, YouTube, or just teaching yourself - any
mix can get you started.

First steps are experiments

A project or club is a way to try something, learn, and meet people. It is not a forever decision.

Start where you are

Time, money, and access are different for everyone. Pick a pace you can keep alongside school.

> *The goal is not to have it all figured out. The goal is to make your **next step doable**.*

1 / Explore a direction, not all of "tech"

Pick one that sounds fun and one backup to explore.

Build and make

- Apps, games, and websites
- Testing and finding bugs
- Cybersecurity and ethical hacking
- Hardware, robotics, and IT

Analyze and connect

- Data, AI, and machine learning
- Design and user experience (UX)
- Product: how apps get planned and made
- Explaining tech: writing, video, teaching

Where do you want to go?

Development

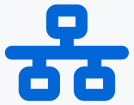
Write and ship software

Testing

Quality and automation

Security

Protect systems and data



Administration

Systems, networks, and IT

Data & AI

Insight, models, and ML

Product

Discovery and delivery

You already have useful strengths

Where it came from	Strengths you built	How it maps to tech
Games and hobbies	problem solving, patience	Debugging, learning systems
Clubs, sports, theater	teamwork, showing up	Collaborating, hitting deadlines
Class projects	research, explaining	Documenting, communicating clearly
A part-time job	responsibility, people skills	Reliability, knowing real users

Prompt: [something you did] + [where] -> [strength a tech project needs]

2 / Build real things

Small, finished projects are your proof.

Why build

- It turns learning into something real
- A finished project shows what you can do
- Building is the fastest way to actually learn

What counts

- A game, a website, or a small script
- A robot part, a bot, or a mod
- Anything you can show and explain

Build skills in a tight loop

The loop

1. Pick **one area** that sounds fun
2. Learn just enough to start
3. Build a **small, finished** project
4. Show it and ask for feedback
5. Do it again, a little harder

Guardrails

- Only learn what unblocks you
- Finish before adding features
- Keep it small enough to explain
- Jot down questions and decisions
- Shrink scope before adding hours

> *Learning gets real when it becomes something someone can play, run, or look at.*

Learn the tools around the code

Worth learning early

- The command line / terminal basics
- Your editor (like VS Code) and a debugger
- Git and GitHub: save and share code
- Reading errors and searching them well

Practice by doing

- Automate one boring, repetitive task
- Break something on purpose, then fix it
- Guess what's wrong before you Google it
- Explain why you picked a tool

> *Knowing the tools around the code makes projects way easier to finish.*

A good project is easy to check out

Explain it

- A README: what it is and who it's for
- Steps to run it that actually work
- What you chose and why
- What's not done yet, and your next step

Show it

- A screenshot, GIF, or short demo video
- A link to try it (game, site, or repo)
- Sample input and output
- What part you built

Match it to the area: a game (playable link), data (a chart), a website (a live link), hardware (a photo or video of it working).

Free places to learn (really free)

Start here

- Experience CS - CS units made for classrooms
- freeCodeCamp and Foundational C#
- Microsoft Learn - guided paths and badges
- GitHub Skills - hands-on, guided courses
- Scratch and MakeCode Arcade - build games fast
- Minecraft Education CS

AI is everywhere - learn to use it well

Why it's a big deal

- It's in apps, search, art, and code
- Almost every field uses it now
- Using it *well* is a real skill
- You don't need to be an expert

Use it with judgment

- Good for explaining and first drafts
- It can be **confidently wrong** - check it
- Don't trust it with private info or facts
- Play with it: test ideas, remix

> *A power tool: know what it's great at, where it breaks, and when to double-check.*

Learn AI for free - and run it yourself

Free courses

- Elements of AI - a gentle intro
- Generative AI for Beginners
- Kaggle Learn - hands-on lessons
- Hugging Face - a free course

Try it yourself

- Copilot - free for students
- Run models locally: Ollama, LM Studio
- Local = free, private, offline
- Build a tiny chatbot or classifier

> *Run real AI on your own laptop - free and private. Check age rules and ask a parent.*

Try contributing to open source

- Open Source Guides - how it all works
- goodfirstissue.dev - beginner-friendly tasks
- First Contributions - practice your first pull request safely
- Fixing docs, typos, or tests all count
- Real work that lives on your GitHub profile
- Free feedback from experienced developers
- See how teams actually work together
- Every fix is a story you can tell later

Share what you're learning

Show the process, not just the result

- Post what you built, learned, or fixed
- Make the guide you wish you'd found
- Keep a simple profile or portfolio page
- Show, don't tell - link to real work

Why it works

- People see real growth, not just claims
- Your projects speak when you're not there
- People remember and think of you
- Opportunities start to find you

> *Share only public, school-safe work - no personal info, and check with a parent about accounts. Inspired by Learn in Public (swyx).*

Keep a project journal

Jot it down as you go

- What you built, fixed, or learned
- The problem, what you tried, how it went
- Screenshots, links, and dates
- Nice feedback you got

Why it pays off

- Applications and resumes get easy to write
- Ready-made stories for interviews
- Proof for scholarships and programs
- Fixes "wait, what did I even do?"

> *A running list of your wins also quiets self-doubt - proof beats "I'm not good enough."*

3 / Find your people

Being part of a community is

- learning alongside other people
- showing up more than once
- helping out where you can
- saying thanks and following up
- becoming known as reliable

Where to find them

- school clubs: coding, robotics, esports
- FIRST Robotics teams and hackathons
- online communities for what you're into
- Discord servers, forums, and open source
- library and community center events

> *It's **not** collecting contacts or DMing strangers - it's showing up and being helpful.*

Where tech gathers near you

Places to look

- School clubs: coding, robotics, esports
- FIRST Robotics and competitions
- Student hackathons and coding competitions
- Library and Meetup events

Make it count

- Pick one and **keep showing up**
- Ask a question; thank the organizer
- Volunteer to help set up
- Even a 5-minute demo counts

> *Most are free and beginner-friendly. Bring a friend or trusted adult.*

Make it easy for someone to say yes

A short message template

> *Hi **[Name]** - I liked your **[video/project/talk]** about **[topic]**. I'm a student learning **[thing]** and building **[small project]**. Could I ask one question: **[question]**? No rush - thanks!*

Good asks

- their take on one decision
- feedback on one project
- a good next thing to learn

Stay safe

- use public, school, or club channels
- tell a parent or teacher
- never share private contact details

Ask someone about their path

The ask

- 15 minutes to hear how they got into tech
- You're curious, not asking for a job
- Offer a few times; make it easy to say yes

Good questions

- How did you get started?
- What does a normal day look like?
- What do you wish you'd learned earlier?
- What should a student like me try first?

> *Always follow up: one or two takeaways, a thank-you, and what you tried next.*
> *A teacher, family friend, or older student is a great first person to ask.*

Good things come through people you know

It's word of mouth

- A teacher mentions a program
- A friend invites you to a team
- A club leader hears about an internship
- People suggest you when they know your work

Be that person

- Show up and help out
- Let people see what you build
- Say what you're looking for
- Say thanks and pass it on

> *You don't get picked by staying invisible - being known and reliable opens doors.*

Different people help in different ways

Build a little support crew

- **Friends and peers** practice with you and keep it fun
- **Mentors** share advice and honest feedback
- **Teachers and counselors** know programs and opportunities
- **Older students and alumni** have just been where you are

> *You don't need all of them at once - friends and one caring adult are a great start.*

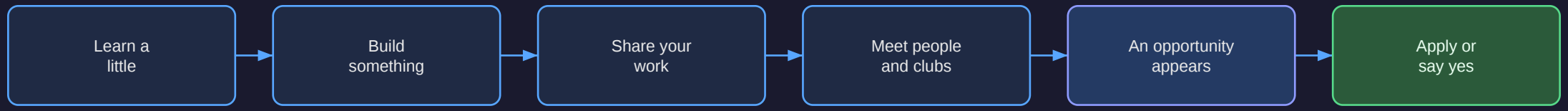
Be someone people want to help

Trust is built over time

- Do what you say you'll do
- Ask for specific feedback, not "will you mentor me?"
- Share what you tried after getting advice
- Help others and share credit

> *People go out of their way for someone who is reliable, curious, and kind - not for whoever asks the most.*

4 / Show up and apply



> *Most first opportunities - a club role, a summer program, an internship - come from having something to show and people who know you.*

A posting is a wish list, not a checklist

What really matters

- You understand the main thing they want
- You are eligible (age, grade, location)
- You can show something related
- The gaps are things you can learn

Often flexible

- The exact language or tool
- "Preferred" experience you don't have yet
- A perfect match to every bullet
- Years of experience for entry programs

> ***Apply when the main thing matches.***

Your first resume is short and real

Keep it simple

- One page, clear headings, easy to read
- Projects, clubs, and skills near the top
- Coursework and activities that fit
- It's fine to be new - show what you've done

Show what you made

- Things you built, not just "member of"
- What it does and your part in it
- Links to a project, game, or repo
- Honest skills you could actually talk about

Show your work, skip the fluff

✅ Show

- A GitHub, itch.io, or portfolio link
- One project you can actually explain
- A screenshot, demo, or live link
- A clear note on what you built

❌ Skip

- Fake "objectives" and skill bars
- Listing every app you've ever opened
- Buzzwords you can't talk about
- Typos - ask an adult to proofread

> *One good project you can explain beats ten half-finished ones.*

Turn "I helped" into evidence

❌ Vague

> *"Was in robotics club and helped with the code."*

✅ Specific

> ***Programmed** the arm controls for our team's robot and wrote a guide so the next team can reuse it.*

The pattern: action + context + what you made + result

> *No numbers? Name a real outcome - built, fixed, explained, or made something easier.*

Talking about your work is a skill

Tell it with STAR

- **S**ituation and **T**ask - set the scene
- **A**ction - what you did and why
- **R**esult - what happened and what you learned
- Have a story about a hard thing, a mistake, and teamwork

Ask them good questions

- What does a typical day or project look like?
- How would I get feedback?
- What could I learn here?
- What's the most fun part?

> *Practice out loud. A "no" is about the match, not about you.*

Tell your story simply

A simple arc

- **Spark:** what got you curious about tech
- **Explored:** classes, clubs, or tutorials you tried
- **Made:** a project you built and finished
- **Next:** what you want to learn or do next

Make it real

- Keep it short - a minute is plenty
- Point to one thing you actually built
- Use one real example, not a list
- End forward: what you'll try next

> *A small project you can show is the bridge - even a Scratch game counts.*

Technical interviews: think out loud

A coding challenge

- Restate the problem in your own words
- Say your plan, then start simple
- Test with examples; talk through edge cases
- Thinking out loud beats going silent

"How would you build it?"

- Ask who it's for and what it needs to do
- Sketch the main parts; name one tradeoff
- Don't overthink - a clear simple idea wins
- "I'd look that part up" is a fine answer

> *People care how you think, not whether you're instantly perfect.*

Use AI as a helper, not a crutch

Use it to learn faster

- Ask Copilot or a chatbot to explain and unstick you
- Then understand every line - don't just paste it
- Be ready to explain how it works yourself

Still do the reps

- Practice coding and problem-solving on your own too
- Follow your school's rules on AI and honesty
- As AI writes the easy parts, thinking clearly matters more

> *"I used AI to get unstuck, then I understood and finished it myself" is the goal.*

Choosing your next step

Look at the whole thing

- What will you actually learn or make?
- Who runs it, and are they supportive?
- Does it fit your schedule and school?
- Cost, travel, and whether you'll enjoy it

Decide thoughtfully

- Ask questions before you commit
- Talk it over with a parent or mentor
- Compare it to how you want to spend your time
- It's okay to say "not right now"

> *Pick the next class, club, program, or project on purpose - not just because it's there.*

A one-month experiment (a few hours a week)

Week	Focus	What to do
1	Choose	Pick one area; try a short tutorial; note a project idea
2	Learn	Learn just enough to start; sketch your project
3	Build	Build the smallest finished version; add a README or demo
4	Share	Post it on GitHub; join a club; find a program to try

Keep it sustainable: ~2-3 hours a week - homework and rest come first. Review each week and adjust.

"No" is normal - keep going

It happens to everyone

- Not making a team or program is common
- A "no" is about one match, not your worth
- Notice one thing to improve, then try again

Keep it steady

- Track what you tried and what you'll try next
- Notice what's working and do more of it
- Lean on friends and mentors when it's discouraging

> *Small, steady effort beats giving up after one "no." Everyone in tech has a pile.*

Starting points to explore

Explore and learn

- Code.org and Scratch
- Experience CS
- freeCodeCamp and Microsoft Learn
- The Odin Project
- roadmap.sh - pick a path

Practice and build

- GitHub Skills
- Exercism
- MakeCode Arcade - build games
- Minecraft Education CS

Starting points (continued)

Clubs and teams

- FIRST Robotics
- Girls Who Code
- CoderDojo

Online communities

- Hack Club - for teens
- freeCodeCamp forum
- dev.to - share your work

Practice and challenges

- Advent of Code
- Codewars & CodinGame
- Project Euler

Start with one club and one community. Keep it public and school-safe - loop in a trusted adult.

Fun ways to keep learning

Learn by playing

- 7 Billion Humans and The Farmer Was Replaced - programming games
- CodinGame and Codewars
- Advent of Code - a yearly December tradition

Build something you'd use

- A game in Scratch or MakeCode Arcade
- A tiny website about something you love
- A Raspberry Pi or micro:bit project
- A Discord bot or a simple phone app

> *The best way to keep going is to build things you actually think are cool.*

Most of this is free

Free ways to get involved

- School clubs, robotics teams, and CS classes
- CoderDojo and library workshops
- Student hackathons and coding competitions
- Volunteering to help at events

Free practice

- HackerRank and LeetCode
- Coding games: CodinGame, Codewars
- Project Euler puzzles
- Advent of Code each December
- Treat challenges as fun reps, not a scoreboard

Free and student tools

- Python, C#/.NET, and JavaScript - free to use
- Visual Studio Code - free code editor
- JetBrains IDEs - free for students
- A Raspberry Pi is a cheap, hands-on computer
- YouTube: free full courses
- GitHub Student Developer Pack - free tools for students
- Azure for Students - free cloud credits

Getting into tech is mostly free. The cost is your time and curiosity, not money.

Three things to remember

1

Pick a direction

Choose one area that sounds fun so your time has a focus.

2

Build real things

Small, finished projects show what you can do.

3

Find your people

Clubs, friends, and mentors open doors and keep it fun.

Your next step: pick one area and start one small project this week.

Questions?

Your path doesn't need to match anyone else's - start where you are.

Chris Ayers

Principal Software Engineer | Speaker | Community Builder

Blog | LinkedIn | GitHub

Bluesky | Mastodon

Get the slides

Scan the code, or open:

chris-ayers.com/tech-career-toolkit

Follow along now, and keep every link, tool, and resource from this talk.