Cofounder

FOUNDER FIELD MANUAL

How To Start A Company

A four-part founder guide to starting, building, selling, and scaling a software company.

Edition: How To Start A Company
Andrew Pignanelli
Abhishyant Khare
Kat McGuire
John Hutton
The General Intelligence Company Of New York
Published by Cofounder
May 2026
cofounder.co/how-to

Inside the guide

A page-native edition of the four-part founder playbook: decide what to start, build the first product, create distribution, and scale the operating loop.

Chapter I. How to Start a Company

Chapter II. How to Build a Company

Inside the guide, continued

A page-native edition of the four-part founder playbook: decide what to start, build the first product, create distribution, and scale the operating loop.

Chapter III. How to Sell

Chapter IV. How to Scale

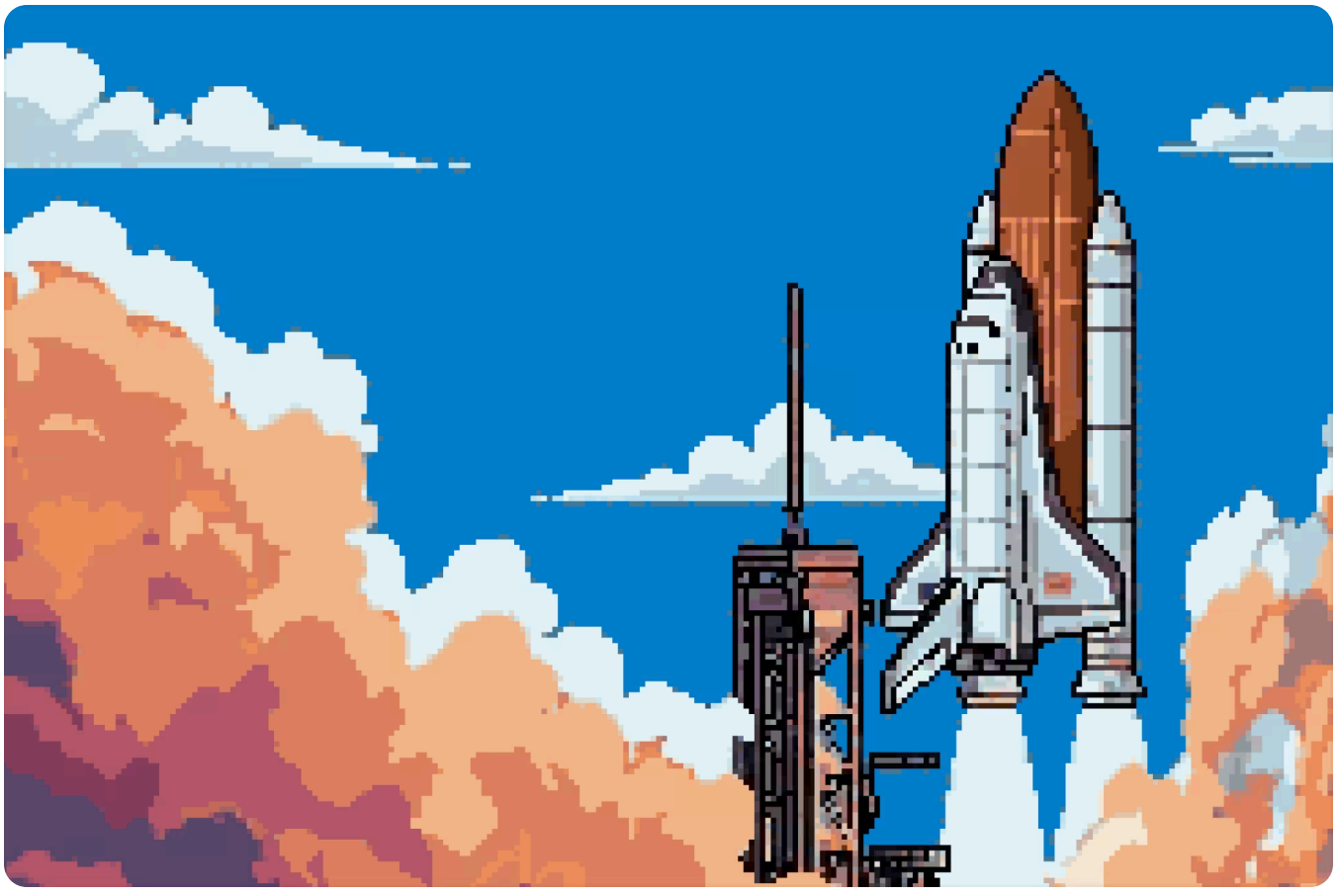
CHAPTER I

I

How to Start a Company

Turn a vague pull toward entrepreneurship into a specific problem, market, name, and next step.

- Start from a painful problem, not a clever solution.
- Choose a narrow wedge inside a market that can become large.
- Make enough progress that the next week is obvious.



CHAPTER I / 01

Introduction

At zero, the job is not to look like a company. The job is to find motion.

Great companies usually start with a simple belief: something should exist that does not exist yet.

At zero, you do not need a perfect plan, a team, funding, or a polished product. You need a problem worth exploring, a customer you can learn from, and enough momentum to take the next step.

This chapter is about what you do from literal zero.

FOUNDER MOVE

Write the problem in one sentence before you write the solution.

CHECKLIST

- [ ] Name the customer.
- [ ] Name the painful workflow.
- [ ] Name the bad current alternative.
- [ ] Name the smallest useful next step.

CHAPTER I / 02

Should You Start a Company?

The question is not whether starting is hard. It is whether this is the right kind of hard for this season of your life.

Everyone should try to start a company at some point in their lives.

It's hard, takes a while to pay off, and sometimes comes with significant risks. But you end up with something you built yourself. The real question is whether it's the right time for you to make an attempt.

A GOOD STARTING POINT IS IF:

- You have seen a painful problem up close.
- You understand a customer or industry better than most people.
- You have access to a market other founders cannot easily reach.
- A new technology makes something possible that was not possible before.
- You keep coming back to the same problem even after trying to ignore it.

WATCH OUT

Do not confuse a desire to be a founder with a customer problem. The company starts when the problem gets specific.

CHECKLIST

- [ ] I have seen the problem directly.
- [ ] I know who feels it.
- [ ] I can reach at least five people who might care.
- [ ] I am willing to spend a few months learning before it feels obvious.

If any of these are true, and you're in a place where you've got extra time (for a side project) or want to take some risk, it's probably the right time to start a company.

CHAPTER I / 03

How to Come Up With Startup Ideas

Most good ideas begin as problems. The solution usually arrives later.

Startup ideas can come from many places. Most good ones start with a problem.

A problem gives you something real to test. If people already feel pain, already spend money, already use awkward workarounds, or already complain about the current options, you have signal. You may still be wrong about the solution, but you are pointed at something that matters.

FOUNDER MOVE

Ask: what became newly possible that customers already wanted?

BETTER FRAMING

"We use AI" is not a startup idea. "We help paralegals review discovery documents in half the time" might be.

MOST STARTUP IDEAS COME FROM THREE SOURCES:

Problems you understand

Here's where to look for problems to solve.

The best source is a problem from your previous work. You were inside a company or industry, watched people struggle with the same workflow over and over, and realized the current tools were not good enough. These ideas are strong because they come with context. You know the customer. You know the language they use. You know what they have tried before. You know which parts of the problem are annoying and which parts are expensive.

A second source is your personal life. You notice an annoyance in the world and wonder why it has not been fixed. This can work especially well when the pain is frequent, obvious, and shared by many people. Uber is the classic example: getting a taxi used to be unreliable, opaque, and frustrating. The problem was not hidden. It was sitting on the street.

A third source is customer research. You pick a market, interview dozens of people, and look for repeated problems. This can work, but it is harder than it sounds. People are not always good at articulating their own problems. They describe symptoms, not causes. They ask for features, not solutions. Customer interviews are still useful, but they are best for sharpening a problem, not replacing your own judgment.

New technology

The iPhone created the mobile app boom. Cloud infrastructure made it easier to start software companies without buying servers. Modern AI is creating new ways to build, search, automate, generate, and interact with software.

When a new technology arrives, it changes what is possible. Things that were too expensive, too slow, or too awkward suddenly become practical. That creates startup opportunities. But it needs to be mapped to a problem space.

The important thing is to connect the technology to a real customer pain.

"We are using AI" is not a startup idea. "We help paralegals review discovery documents in half the time" might be. The technology is the unlock, not the reason customers care.

A useful question: what became newly possible that customers already wanted?

Specific opportunities

Some startup ideas come from a specific opportunity you have unusual access to.

You might know a market because of your family, your job, your location, your relationships, or your reputation. You might have access to a distribution channel others cannot easily copy. You might understand a regulatory change before the market reacts. You might know how a niche industry actually works because you have lived inside it. This is called alpha — opportunity you have that's higher than the market has access to.

These opportunities can be powerful because they give you a head start.

Use alpha as a wedge to build something durable. A relationship can help you win the first customer, but the company still needs repeatable value. A regulatory change can create urgency, but the company still needs a product. A unique distribution channel can get attention, but the product still needs to retain customers.

CHAPTER I / 04

Evaluate Startup Ideas

A startup idea is not good because it sounds exciting. It is good when it survives pressure.

Not all startup ideas are worth pursuing. Whenever you want to go for something, evaluate the idea critically across four lenses.

Market Size

The market should be big already or small but likely to become huge. A big existing market means people already spend money in the category — demand is proven. The opportunity is to serve a specific segment better than the current options.

A small but growing market can be even better. New technologies, regulations, platforms, and behaviors often create markets before they look large on a spreadsheet. The risk is timing, but the upside is that you can enter before the answer is obvious. Do not choose a market solely because it is large. A market is useful when you can identify a reachable customer with a painful problem and a budget. If your market is too small, you will run into serious issues growing revenue.

Founder/Market Fit

Founder/market fit means you are able to understand and serve the market. Maybe you worked in the industry. Maybe you are the customer. Maybe you have relationships that help you get early meetings. Maybe you understand the culture and constraints of the buyer better than outsiders do.

What this means in practice is: do you have a realistic shot at actually starting this company? With software, in 2026, the answer is probably yes. But if you're not a lawyer trying to start a law firm, it's probably not.

WATCH OUT

Do not wait for the perfect idea. A month of real customer conversations beats a year of abstract brainstorming.

CHECKLIST

- ○ The customer already spends money or time on the problem.
- ○ The current workaround is expensive, slow, risky, or hated.
- ○ You have a realistic way to build and reach the first customers.
- ○ You believe something specific that the market has not fully priced in.

Pain

The best startup ideas solve painful problems. Pain means the customer already cares — they are losing money, wasting time, missing opportunities, taking on risk, or bound by a workflow they hate. A nice-to-have product can work, but it is much harder to sell. A painful problem pulls the product into the market.

A good test: what does the customer do today because your product does not exist, and would they pay you to fix it? If they are using spreadsheets, hiring people, duct-taping tools together, paying consultants, or tolerating a bad incumbent, that is a strong signal. Workarounds are evidence of demand.

Unique Insight

A unique insight is something you believe that most people in the market have not realized yet. It might be that a customer segment is underserved because incumbents are focused upmarket. It might be that a workflow can now be automated because of a new model. It might be that buyers dislike the current tools more than outsiders realize. It might be that a product that worked in one country or industry can work in another.

Without a unique insight, you are probably entering the market the same way everyone else sees it. That does not mean you cannot win, but it makes the path harder.

SOME EXAMPLES OF POSITIVE SIGNALS:

- You are building something you personally want.
- It only recently became possible.
- There are successful analogs elsewhere.
- Customers already hacked together a solution.
- The market pulls the product from you.

COMMON MISTAKES:

- Avoiding ideas that seem too hard. Hard ideas can be good because fewer people attempt them and it is easy to get people and capital to rally around hard ideas.
- Avoiding boring industries. Many of the best startup opportunities are not glamorous. They live in logistics, insurance, construction, compliance, accounting, healthcare administration, manufacturing, legal workflows, and other places where software is old and customers have real budgets. Boring problems often make excellent businesses.
- Being too afraid of competitors. Competition usually means the problem exists. The question is not "are there competitors?" The question is "why will customers choose us now?"
- Waiting for the perfect idea. You will learn more from a month of serious customer conversations and prototyping than from a year of abstract brainstorming.

CHAPTER I / 05

Pick a Wedge Market

The wedge is the first customer group you can find, understand, sell, and use as proof.

A startup idea is the wedge. The market is the world around it. Picking a market matters because it determines who you learn from, how you sell, how much customers can pay, how large the company can become, and what kind of product you need to build.

A GOOD EARLY MARKET HAS A FEW TRAITS:

- The customer is easy to describe.
- The pain is frequent or expensive.
- The buyer has budget.
- You can reach customers without needing a giant sales team.
- The current alternatives are bad, expensive, slow, or outdated.
- The market is changing in a way that creates an opening.

Start narrow. This feels counterintuitive because founders want the company to be big. But the best way to become big is often to start with a small group that cares intensely.

Do not say your customer is "small businesses." Say your customer is "independent dental practices with three to ten locations that struggle with insurance claim denials." Do not say your customer is "creators." Say your customer is "YouTube educators selling paid cohorts who need to convert viewers into students." Specificity helps you build, sell, and learn. You know where to find customers. You know what language to use. You know which features matter. You know who to ignore.

FOUNDER MOVE

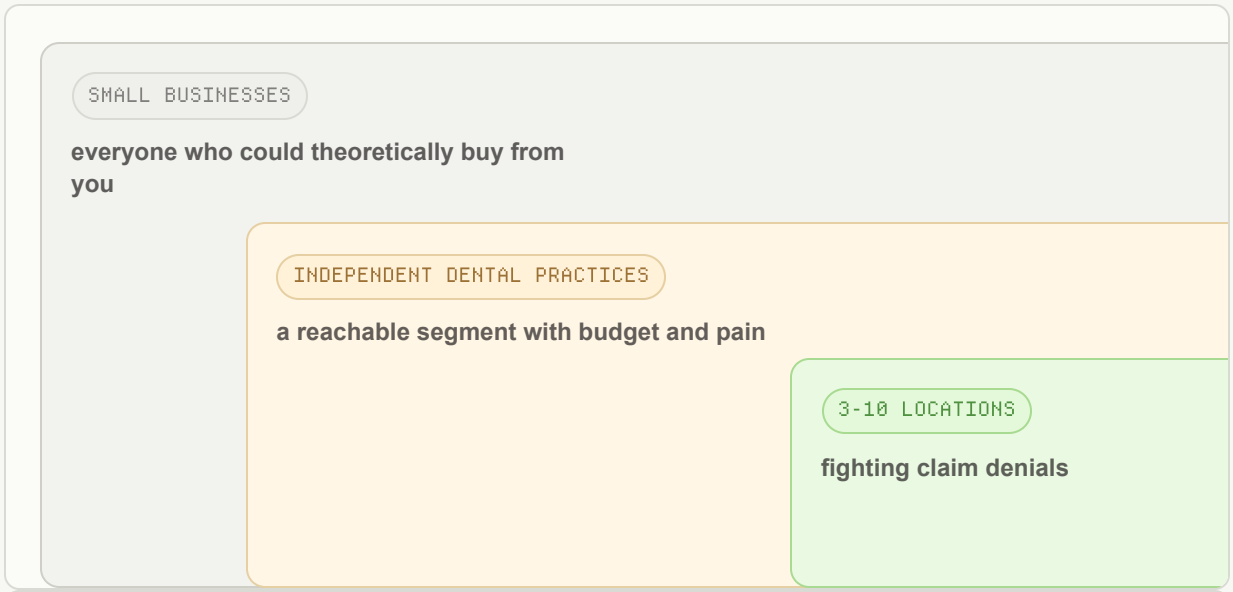
Write three versions of the customer: too broad, reachable segment, first wedge.

CHECKLIST

- Easy to describe.
- Frequent or expensive pain.
- Budget exists.
- Reachable without a giant sales team.
- Current alternatives are bad or outdated.

MARKET SIZE

The first wedge should be small enough to reach, inside a market big enough to matter.



A wedge is not the final market. It is the first customer group you can find, understand, sell, and use as proof for the larger opportunity.

CHAPTER I / 06

Pick a Name and Domain

A name should help the company move. It should not become the company.

Naming feels important because it is visible — and it is. You're building a brand. People will know your company by this brand. A good name should be simple, memorable, easy to spell, and flexible enough to survive changes in the product. You do not need a perfect name. You need a name that does not get in the way.

A FEW RULES:

- Avoid names people cannot spell after hearing once.
- Avoid names that lock you into a tiny feature.
- Avoid names that sound too similar to existing companies.
- Avoid cleverness that only makes sense after an explanation.
- Check basic trademark risk before you get attached.
- Make sure you can get a reasonable domain and social handles.

A dot-com domain is ideal, but not required at the very beginning. A clean alternative domain is fine if it lets you move. Do not spend months negotiating for the perfect domain before you have talked to customers.

Other options include .co, .app, .io, .ai, .dev, and .tv (if in media), or .xyz (if you're in crypto). Do not use other TLDs — they rank lower and confuse customers. Startups will sometimes pick a good name then add "try" or "use" to the beginning of a common domain name.

WATCH OUT

Do not spend months negotiating for the perfect domain before you have talked to customers.

CHECKLIST

- [ ] Easy to spell after hearing once.
- [ ] Not locked to a tiny feature.
- [ ] Not too similar to another company.
- [ ] Basic trademark risk checked.
- [ ] Reasonable domain and social handles available.

CHAPTER I / 07

Take the Leap

The beginning of a company should turn uncertainty into motion.

The beginning of a company should turn uncertainty into motion. Do not try to solve the next five years. Solve the next week. Start the company.

WHAT COMES NEXT:

Once you have chosen a problem and are committed to testing it, the next step is to build and deploy your MVP. That's what Chapter II covers.

FOUNDER MOVE

Choose the smallest public commitment you can make this week: a landing page, a customer interview sprint, or a prototype.

CHECKLIST

- [ ] Problem chosen.
- [ ] First customer segment named.
- [ ] Domain direction picked.
- [ ] Next seven days planned.

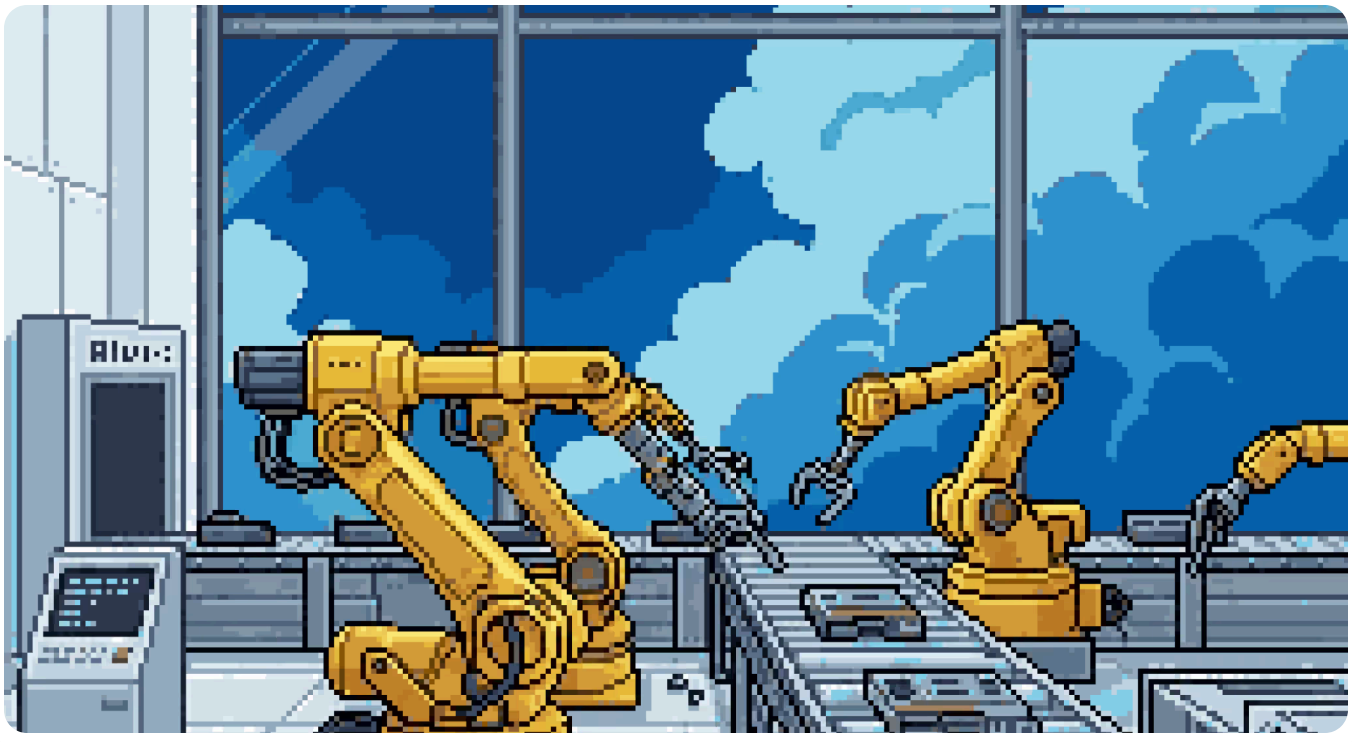
CHAPTER II

II

How to Build a Company

Move from idea to a real product: spec, repo, deployment, secrets, backend, frontend, tests, and production operations.

- A product plan is a learning plan.
- Set up production habits before the product feels mature.
- Build the smallest thing that can teach you what matters.



CHAPTER II / 01

Introduction

Building is not opening a code editor and hoping momentum appears.

Setting up an engineering department is more than just writing code. You need architecture, a database, infrastructure, testing, design, deployment, and a plan to keep it all running as you grow. Most first-time founders dramatically underestimate this; they think "building" means opening a code editor and hacking until something works. That approach gets you a prototype (which are still important when you start!). It does not get you a company.

This chapter walks through the full lifecycle of building a software product, from initial spec to production deployment to ongoing infrastructure management. At each step, the goal is to understand what matters, what founders usually get wrong, and what you need to own as the company becomes real.

FOUNDER MOVE

Define the first product as a testable promise, not a full vision.

CHECKLIST

- ○ Spec
- ○ Repository
- ○ Deployment
- ○ Auth and database
- ○ Frontend
- ○ Tests
- ○ Production monitoring

CHAPTER II / 02

Come up with a spec

The spec is how you decide what not to build yet.

Before you write a single line of code, you need a plan: a clear, minimal specification of what your product does and who it's for.

The goal at this stage is to define your minimum viable product (MVP). This is the smallest version of your product that delivers real value to a real user. It is not a landing page. It is not a demo. It is a working piece of software that solves a specific problem well enough that someone would actually use it.

TO DEFINE YOUR MVP, ANSWER THREE QUESTIONS:

- ○ **What is the core problem you're solving?**
  Be ruthlessly specific. "Helping people manage their finances" is not a problem statement. "Freelancers lose track of quarterly tax estimates because existing tools assume W-2 employment" is. The tighter your problem definition, the easier every subsequent decision becomes.

- **What is the minimum set of features needed to solve that problem?**

  List every feature you think you need, then cut half of them. Seriously. If your MVP has more than three to five core features, it's too big. You're not building the final product; you're building the first thing real users will test.

- **What platform are you building for?**

  Web, mobile, or desktop? For most startups, start with web. It's the fastest to build, the easiest to deploy, and the simplest to iterate on. You can always go native later once you've validated the product.

Once you have answers, write them down. This is your spec. It doesn't need to be formal — a one-page document or even a detailed bullet list works. What matters is that you've thought through the scope and committed to a boundary.

WATCH OUT

A spec should create clarity. If it becomes a graveyard for every possible feature, it is not doing its job.

CHECKLIST

- Who is the user?
- What job are they trying to do?
- What is broken about the current workflow?
- What are the three to five essential features?
- What must be true for the MVP to be worth continuing?

CHAPTER II / 03

Create a repository

A repository is the home base for the product, even when one person is building.

Your code needs a home. That home is a Git repository, hosted on GitHub.

If you're not familiar with Git, here's the short version: Git is a version control system that tracks every change you make to your code. GitHub is a platform that hosts your Git repositories in the cloud, lets multiple people collaborate on the same codebase, and provides tools for code review, issue tracking, and automation.

Why does this matter? Because without version control, you're one bad edit away from losing hours (or days) of work. With it, you can experiment freely, roll back mistakes, and work with teammates without stepping on each other's code.

SETTING UP A REPOSITORY INVOLVES A FEW THINGS:

- Initialize the repo with a sensible folder structure for your chosen tech stack.
- Set up branch protection on your main branch so nobody (including you at 2 AM) can push broken code directly to production.
- Configure a README that explains what your project is, how to set it up locally, and how to contribute.
- Add a .gitignore so you don't accidentally commit secrets, node_modules, or other garbage.

FOUNDER MOVE

Keep main deployable. Use branches for work in progress.

CHECKLIST

- Private repo created
- README started
- Main branch protected when needed
- Deployment connected
- Environment variables documented

VERSION CONTROL

A repository is the source of truth for your product.

Every meaningful change should move through a branch, a review, and a protected main branch before production sees it.

MAIN

Protected branch

Only reviewed code lands here.

FEATURE BRANCH

Work in progress

Changes happen away from the deployable version.

PULL REQUEST

Review and merge

Discuss, test, and merge when the change is ready.

Repo hygiene: README, .gitignore, branch rules, environment docs, and clear project structure.

A quick note on terminology you'll encounter throughout this chapter: a branch is a parallel copy of your code where you can make changes without affecting the main version. Think of it like a Google Doc suggestion — you're proposing changes that can be reviewed before they're accepted. A pull request (or PR) is how you propose merging a branch's changes into the main codebase. It's a formal "here's what I changed, please review it" request. These two concepts are fundamental to how modern software development works, and you'll see them referenced throughout this guide.

If you want to learn more about Git and GitHub fundamentals, GitHub's own Getting Started guide is excellent. But honestly, for most founders, you don't need to become a Git expert. You need a repo that's set up correctly, and then you need to write code.

CHAPTER II / 04

Set Up Deployments

Deploy before the product is interesting. It changes how you build.

Before you start coding, set up deployment. Yes, before. This is counterintuitive but important.

Here's why: if you wait until you've written a bunch of code to figure out deployment, you'll inevitably run into environment-specific bugs, configuration headaches, and "it works on my machine" problems. By setting up deployment first, every change you make is immediately visible in a real environment. This tightens your feedback loop dramatically.

We recommend Vercel for deployment. It's built for modern web frameworks (especially Next.js), handles scaling automatically, and has an excellent developer experience. Vercel is free to start with a generous hobby tier, and paid plans start around $20/month per team member when you need more.

YOU NEED THREE ENVIRONMENTS:

1. Deployment — traditionally, this is your local machine where you write and test code. In a traditional setup, this means installing Node.js, a code editor, and running your app locally with commands like npm install and npm run dev. This can be a pain to configure, and "it works on my machine" is one of the most common problems in software development.
2. Staging — a cloud environment that mirrors production. This is where you test changes before they go live. Staging should be connected to your main development branch.
3. Production — the live environment your users see. This should only receive code that has been tested in staging.

You should also set up preview environments. Vercel does this natively — every time you push a new branch or open a pull request, Vercel creates a unique URL where you can see exactly what your changes look like. This is incredibly useful for reviewing features before merging them.

Preview environments become even more powerful when you're working with AI agents. In a traditional development workflow, you'd review code in a pull request, try to visualize the changes in your head, and maybe run the branch locally to actually see them. With preview environments, every PR an agent opens is immediately deployed to its own live URL. You can see exactly what the agent built, click through it, and decide whether to merge — all without pulling a single branch to your local machine.

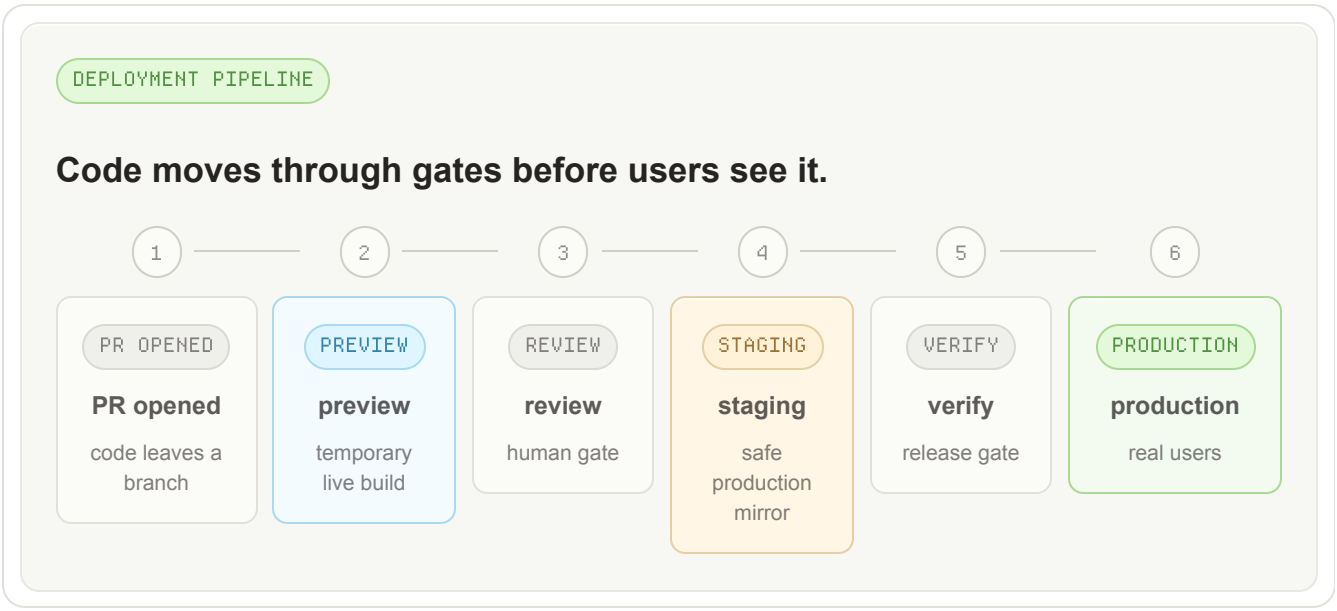
This matters because agents work fast. A single agent can open multiple PRs in the time it takes you to review one. Preview environments let you keep up: you can review several changes in parallel, each in its own live environment, and merge the ones that look right. It turns code review from a bottleneck into a lightweight quality gate. Instead of being the person who writes every line, you become the person who steers — directing agents, reviewing their output, and shipping the good stuff. This is what agent-native development looks like, and preview environments are the infrastructure that makes it practical.

WATCH OUT

If deployment is hard at the end, every product decision starts carrying operational fear.

CHECKLIST

- ○ Production project created
- ○ Preview deploys enabled
- ○ Custom domain planned
- ○ Environment variables separated by environment
- ○ Rollback path understood



CHAPTER II / 05

Secret Management

Secrets are part of the product, but they should never live in the product code.

Your app will need API keys, database connection strings, auth secrets, and other sensitive configuration values. These should never be hardcoded in your codebase — not even in a private repo. If a secret ends up in a Git commit, it's in your history forever (or at least until you go through the painful process of rewriting history to remove it).

The right approach is to use environment variables. These are values stored outside your code that get injected into your app at runtime. Vercel has built-in support for environment variables, and you can set different values for each environment (development, staging, production). This means your staging app can talk to a staging database while your production app talks to the real one, all without changing a line of code.

FOUNDER MOVE

Add a sample env file with names and descriptions, not values.

CHECKLIST

- [ ] No secrets committed
- [ ] Local env file ignored
- [ ] Preview and production values separated
- [ ] Required variables documented
- [ ] Rotation plan known

A FEW RULES OF THUMB FOR SECRETS MANAGEMENT:

- Never commit .env files. Add them to your .gitignore immediately.
- Use different secrets per environment. Your staging Supabase keys should not be the same as production.
- Rotate secrets periodically, especially if someone leaves the team or a key is accidentally exposed.
- Limit access. Not everyone on your team needs access to production secrets.

CHAPTER II / 06

Start Building: Scaffold Your App

Scaffolding is where the product gets a shape users and engineers can both understand.

Now it's time to actually write code. The first step is scaffolding — setting up the foundational structure of your application.

Any modern web app has two sides:

- The frontend — what your users see and interact with. Buttons, forms, pages, layouts.
- The backend — what happens behind the scenes. Storing data, authenticating users, running business logic, serving API responses.

FOR YOUR TECH STACK, WE RECOMMEND:

- Next.js for your frontend and API layer. Next.js is a popular web framework that lets you build both the pages your users see and the server-side logic that powers them — all in a single project. Instead of maintaining two separate codebases (one for the website, one for the server), everything lives together. It's built on React, the most widely-used library for building user interfaces.
- Supabase for your database and authentication. Supabase gives you a database (where all your app's data lives — user accounts, content, settings, everything), built-in login and sign-up, and file storage. Think of it as the back office of your app. It's open source, has a generous free tier, and scales well.

This stack — Next.js + Supabase — is battle-tested, well-documented, and one of the fastest ways to go from zero to a working product.

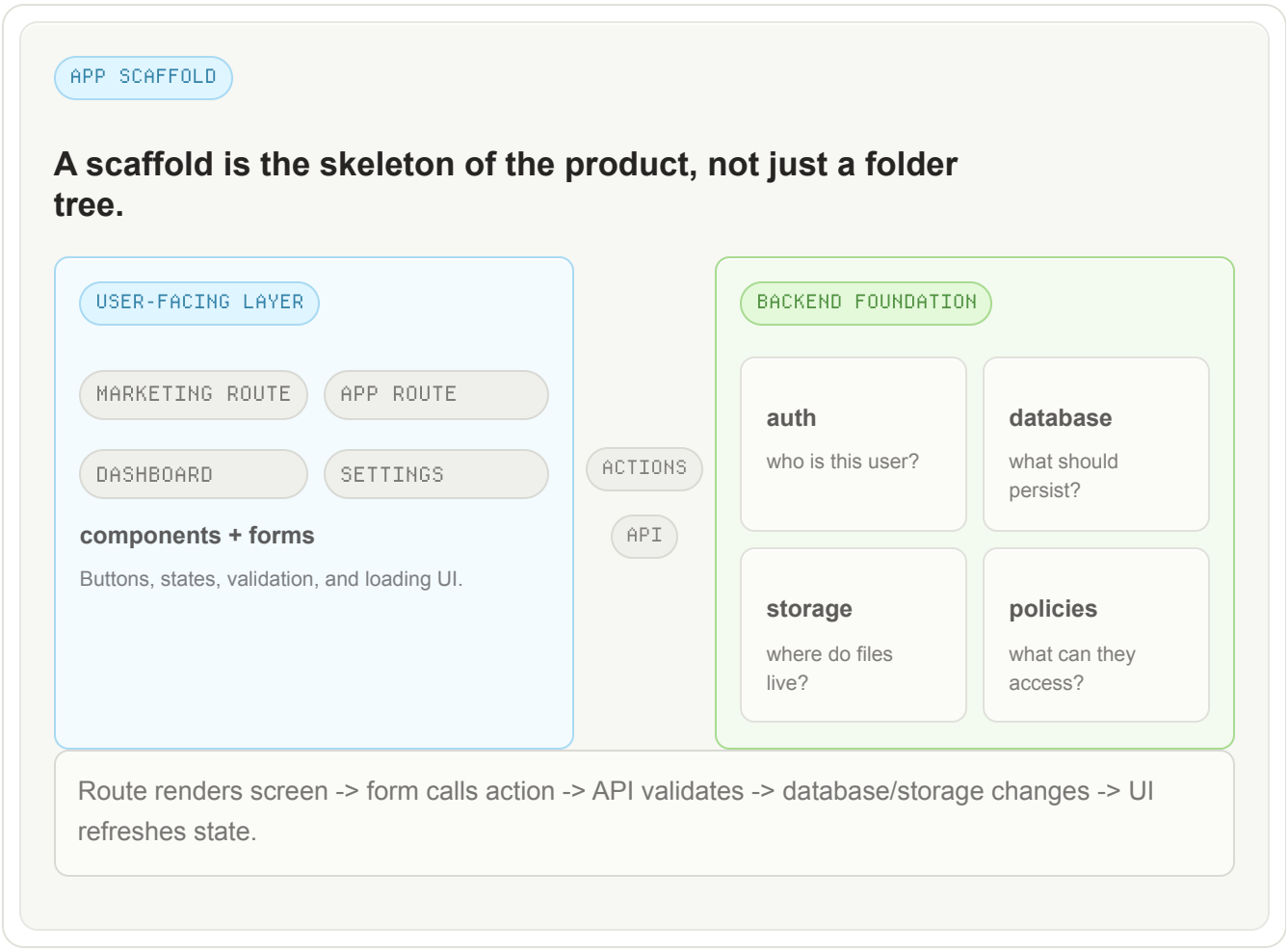
On cost: GitHub is free for private repositories. Supabase has a free tier that's more than enough for development and early users, with paid plans starting at $25/month when you need more storage or compute. Between Vercel and Supabase, you can realistically run a production app for under $50/month until you have meaningful traffic. Don't let infrastructure costs scare you — the expensive part of building a startup is your time, not your server bill.

WATCH OUT

Do not build clever abstractions before you have repeated pain. Early abstractions often preserve the wrong assumptions.

CHECKLIST

- [ ] Routes mapped
- [ ] Layout shell built
- [ ] Auth screens planned
- [ ] Core data model sketched
- [ ] Empty and loading states included



CHAPTER II / 07

The Backend: What You Need at Minimum

The backend is where trust, data, and business logic live.

Your backend is the foundation everything else sits on. Get it wrong and you'll be dealing with security vulnerabilities, data loss, and architectural rewrites down the line. Get it right and it quietly does its job while you focus on the product.

At minimum, your backend needs four things:

Authentication

Authentication is how your app knows who a user is. It covers sign-up, login, password reset, session management, and (increasingly) social login via Google, GitHub, etc.

Do not build your own authentication system. This is one of those areas where rolling your own solution is almost always a mistake. Auth looks simple on the surface — it's just a login form, right? — but under the hood, there are dozens of security concerns: securely storing passwords,

managing login sessions, preventing brute-force attacks, handling "forgot password" flows, and more. Getting any one of these wrong can expose your users' accounts. Use a proven solution.

Supabase Auth handles all of this out of the box. It supports email/password login, magic links (passwordless login via email), and social login with providers like Google, GitHub, and Apple. It also integrates with Supabase's database security so you can control what data each user is allowed to see — for example, ensuring users can only access their own records, not anyone else's.

A Database

You need somewhere to store your data. Supabase gives you a full database that can handle everything from simple user profiles to complex data with many relationships between different types of records. It's the same type of database (PostgreSQL) that powers some of the largest apps in the world, so it won't be something you outgrow.

Think carefully about your data structure early. What are the main things your app tracks? Users, orders, messages, projects? How do they connect to each other? For example, a user has many orders, and each order has many items. Getting this structure right early saves you from painful restructuring later.

That said, don't overthink it. Your data structure will evolve as your product evolves. Start with what you need for your MVP and iterate.

A note on database migrations: when you need to change your database schema after you've already launched (adding a new column, renaming a field, changing a relationship), you can't just edit the schema like you would a document. You need to write a migration — a set of instructions that transforms the database from the old structure to the new one without losing existing data. Supabase has built-in migration support, and this is one area where being careful pays off. A botched migration on a production database with real user data is one of the worst situations you can be in as a founder.

An API Layer

Your API layer is how the pages your users see communicate with your server and database. When a user clicks "submit" on a form, or a page needs to load a list of items, those requests go through your API. In Next.js, you define these directly in your project — no separate server needed.

Your API handles things like: loading data for a page, saving form submissions, running business logic ("when a user does X, also do Y and Z"), and talking to third-party services like Stripe or Postmark.

Keep these routes clean and well-organized. Make sure they check that incoming data is valid before doing anything with it, and that they return helpful error messages when something goes wrong. These are the boring fundamentals that separate a production-quality app from a hackathon project.

Third-Party Integrations

Almost every real product needs to integrate with external services. Two are nearly universal:

Payments. If you're charging users, you need a payment processor. We recommend Stripe. It handles credit card processing, subscriptions, invoicing, and tax compliance. Stripe's API is well-documented and battle-tested — it powers payments for companies from early-stage startups to Fortune 500s. Stripe charges 2.9% + 30¢ per transaction with no monthly fee, so you only pay when you make money.

Setting up Stripe involves creating an account, adding Stripe's code library to your app, building the checkout experience your users will see, and setting up webhooks. Webhooks are how Stripe tells your app about events — "this payment succeeded," "this subscription was cancelled," "this card was declined." Your app needs to listen for these notifications and respond appropriately (e.g., granting access after a payment, or sending a follow-up email after a cancellation). This is one of those integrations that seems simple but has a lot of edge cases: failed payments, refunds, subscription upgrades, partial charges, and so on.

Transactional email. Your app will need to send emails — welcome emails, password resets, receipts, notifications. We recommend Postmark for this. Postmark is fast, reliable, and focused specifically on transactional email (as opposed to marketing email). It has excellent deliverability, which means your emails actually land in inboxes instead of spam folders. Postmark offers a free tier for development, with paid plans starting at $15/month for 10,000 emails.

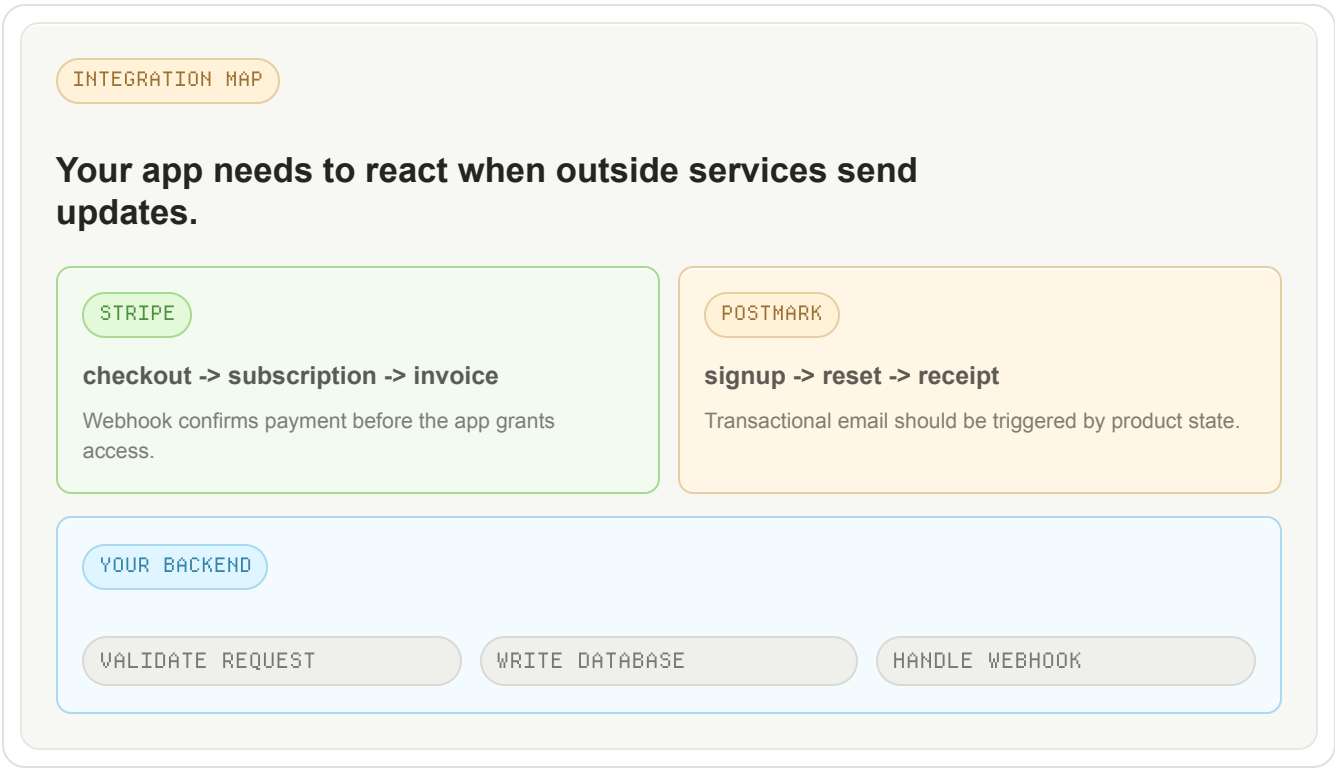
Other integrations you might need depending on your product: file storage (Supabase includes this), analytics (covered in the Scale chapter), error monitoring, and various SaaS APIs specific to your domain.

WATCH OUT

Security is still your responsibility even when tools make setup easier. Know what data each user can access and why.

CHECKLIST

- [ ] Authentication
- [ ] Database
- [ ] API routes or server actions
- [ ] Payment provider
- [ ] Transactional email
- [ ] Authorization rules



Security

This is the one that trips up most first-time founders. You are responsible for making sure your app doesn't leak user data. This means:

- Only send users the data they need. If someone requests their profile, don't accidentally include other users' data in the response.
- Don't trust anything that comes from the browser. Always double-check data on the server before saving it. Users (or attackers) can send anything they want — your server needs to verify it's valid.
- Make sure users can only see their own data. Supabase has a feature called row-level security that enforces this at the database level, so even if your code has a bug, users still can't access each other's records.
- Keep secrets out of your codebase. API keys, database passwords, and other credentials should live in environment variables (covered in the Secrets Management section above), never in your code.
- Use HTTPS everywhere. This encrypts data in transit so it can't be intercepted. Vercel handles this automatically.

This is not optional. A data breach can kill a startup — both legally and reputationally. Even if you use tools to help set things up, it is still on you as the founder to understand your security posture and ensure user data is protected.

The Frontend: Building What Users See

The frontend is where users decide whether the product feels obvious.

The frontend is where your product comes to life. It's what users interact with, and it's often the difference between a product people love and one they abandon after thirty seconds.

Frontend development in 2026 is primarily done in TypeScript (a typed superset of JavaScript) using component-based frameworks like React (which Next.js is built on). The good news: this is an area where AI agents excel. Translating designs and user flows into working UI code is one of the things agents do best.

That said, the agent is only as good as your direction. You need to deeply understand your UX and business logic:

- What are the core user flows? Map out every path a user takes through your app, from sign-up to their first meaningful action to ongoing usage. Think about this before you build anything.
- What happens at each step? What data is displayed? What actions can the user take? What happens when something goes wrong? What are the edge cases?
- What does it look like? You don't need pixel-perfect designs, but you need a clear sense of layout, hierarchy, and interaction patterns. Wireframes or rough sketches are fine.

Design Without Designers

Traditionally, building a frontend meant hiring a designer to create mockups in Figma, then handing those designs to a developer to implement. This was slow, expensive, and created constant back-and-forth between design and engineering.

In 2026, agents can design directly in code. Instead of creating a static mockup and then translating it, you describe what you want and the agent builds a working version immediately. You can see it, click through it, and iterate on it in real time. This collapses the design-to-code pipeline into a single step.

This doesn't mean design thinking is irrelevant. You still need to understand your user flows, information hierarchy, and interaction patterns. But the artifact you produce is working code, not a Figma file. And because iteration is nearly instant, you can try ten variations in the time it used to take to get feedback on one mockup.

Once you have this clarity, building the frontend is largely execution. You're translating your flows and product thinking into components, pages, and interactions.

FOUNDER MOVE

Use the product on a phone before you call the first version done.

CHECKLIST

- Clear navigation
- Primary action visible
- Empty states helpful
- Errors understandable
- Mobile checked
- Core flow tested end to end

FRONTEND LOOP

Use the app like a customer, then fix the parts that feel off.

1

test the app

Click through the real flow.

2

find odd parts

Confusing copy, dead ends, awkward states.

3

fix and repeat

Ship the smallest useful improvement.

Keep cycling until the product flow feels obvious to someone using it for the first time.

CHAPTER II / 09

Deploy to Production

Production is not a ceremony. It is the first honest environment.

You've got a working app in staging. It's time to go live.

Deployment is the process of taking your tested code and making it available to real users. If you set up Vercel earlier (which you should have), this process is straightforward:

1. Quickly test in preview environments. Before anything gets merged to staging, review it in its preview environment. Click through the changes, test the flows, make sure it looks and works the way you expect.
2. Test thoroughly in staging. Once branches are merged, test the integrated result in staging. Use your app like a real user would. Try to break it. Click things you're not supposed to click. Enter data you're not supposed to enter. Find the bugs now, not after launch.
3. Merge to main. Once you're confident staging is solid, merge your code to the main branch. This should go through a pull request with a code review (even if you're reviewing your own code).
4. Deploy. Vercel automatically deploys your main branch to production. If you've set things up correctly, this is a one-click operation.
5. Verify in production. After deployment, check that everything works in the live environment. Sometimes things that work in staging break in production due to environment differences (different API keys, different database, different domain configuration).

WATCH OUT

Do not let the word production make the product precious. Ship small, then improve with evidence.

CHECKLIST

- Domain connected
- Production env set
- Database migration applied
- Core flow works
- Analytics or logs available
- Rollback path ready

A note on backend deployment: Vercel handles your frontend and API routes natively. For anything that needs to run in the background or run longer than a standard API request — things like sending a batch of emails, processing a file upload, running an AI pipeline, or handling a webhook that kicks off a chain of operations — you'll want Vercel Workflows.

Vercel Workflows lets you run background processes alongside your app. Here's the problem it solves: a normal API request needs to respond in a few seconds, but some operations take much longer — sending 1,000 emails, processing a large file, or running a multi-step AI pipeline. Workflows lets you kick off these long-running tasks, and it handles the hard parts automatically: if a step fails, it retries it; if the process is interrupted, it picks up where it left off; and you get a dashboard showing exactly what ran and when. Unlike older approaches that required setting up separate infrastructure, Workflows lives inside your existing Vercel project — no extra services to manage.

CHAPTER II / 10

Testing and Quality: Avoiding AI Slop

Testing is how you protect momentum from your own speed.

Here's an uncomfortable truth: AI-generated code can be mediocre. It works, technically, but it's often verbose, poorly structured, and riddled with subtle issues that only surface under real-world conditions. The industry has a term for this: AI slop.

Avoiding slop requires discipline. Here's how:

FOUNDER MOVE

Write tests for the path that creates value, not every implementation detail.

CHECKLIST

- Core flow test
- Lint command
- Build command
- Rules file for agents
- Manual QA checklist
- Bug capture process

QUALITY CONTROL

Generated code needs to pass three checks before it ships.

BEHAVIOR TESTS

Does the core flow work?

CODE HYGIENE

Is it consistent and low-risk?

HUMAN REVIEW

Does it feel right in the app?

Bad tests are another form of slop. They must prove user outcomes, not merely confirm implementation details.

Write Tests

Automated tests verify that your code works as expected. There are several types:

- Unit tests verify individual functions and components in isolation.
- Integration tests verify that multiple parts of your system work together correctly.
- End-to-end (E2E) tests simulate real user interactions — clicking buttons, filling forms, navigating between pages.

You don't need 100% test coverage on day one. Start with tests for your core flows — the critical paths that, if broken, would make your app unusable. Then expand coverage over time.

A crucial warning: when using agents to write tests, verify that the tests actually test something meaningful. A common failure mode is agents writing tests that are circular — they essentially check "does this code do what this code does?" instead of "does this code do what the user needs it to do?" Even worse, agents sometimes write tests that are rigged to pass no matter what, or that skip testing the actual important part. Always review test logic manually. If a test looks too simple or too clean, it probably isn't testing anything useful.

Set Up Linting and Formatting

Linters are tools that automatically scan your code for common mistakes and style inconsistencies — think of them like spell-check and grammar-check for code. The standard tools for JavaScript/TypeScript projects are ESLint (catches potential bugs) and Prettier (keeps formatting consistent). Set them up to run automatically every time code is saved or committed. This is especially important when AI agents are writing code, because different agents (or the same agent on different days) might use slightly different styles. Linters keep everything consistent.

Configure Agent Rules

If you're using AI agents to write code (and you should be), set up rules files that guide their behavior. These are typically agents.md or .cursorrules files in your repo root that tell the agent about your codebase conventions, tech stack, and patterns.

Good rules files include:

- Your tech stack and versions.
- Coding conventions (naming, file structure, import patterns).
- What libraries to use (and not use) for common tasks.
- Examples of well-written code in your project.
- Common pitfalls specific to your codebase.

The better your rules file, the better your agent output. Think of it as onboarding documentation for an AI team member.

Test Like a User

Finally, use your own product. Constantly. Every feature you build, use it the way a real user would. Click through the flows. Try edge cases. Use it on your phone. Use it on slow internet. This kind of hands-on testing catches things that automated tests miss — awkward flows, confusing copy, interactions that technically work but feel wrong.

When Things Break: Debugging in Production

Most bugs are not mysterious. Finding them is the skill.

Your app will break. Not if — when. A database query will time out. An API integration will return unexpected data. A user will find a flow you never tested. This is normal, and how quickly you can diagnose and fix issues is one of the most important skills you'll develop as a founder.

WATCH OUT

Do not debug by changing random things. Form a hypothesis, test it, then move.

CHECKLIST

- Console checked
- Network request checked
- Server logs checked
- Recent deploy reviewed
- Env vars verified
- Database query inspected

The first question when something breaks is: where do you look?

Vercel logs show you what's happening in your API routes and serverless functions. If a page isn't loading or an API call is failing, start here. Vercel's dashboard shows real-time logs, error rates, and response times.

Supabase logs show you what's happening in your database. If data isn't saving correctly, queries are slow, or authentication is failing, check Supabase's dashboard. It shows query performance, auth events, and database errors.

Browser developer tools are your frontend debugging toolkit. Every browser has built-in tools (usually opened with F12) that let you inspect network requests, see JavaScript errors, and examine the page structure. For a non-technical founder, the "Console" tab (which shows errors) and the "Network" tab (which shows what requests your app is making) are the most useful.

For more serious monitoring, consider adding error tracking with a service like Sentry. It automatically captures errors in your app, shows you exactly where they happened, and groups similar errors together so you can prioritize fixes.

The key insight about debugging: most bugs aren't mysterious. They're usually one of a few things — a typo, a missing environment variable, an API that changed its response format, or a database query that doesn't account for a new edge case. The hard part isn't fixing the bug; it's finding it.

CHAPTER II / 12

Infrastructure Management

After launch, the job is reliability and cost.

Congratulations, your app is live. Now you have to keep it running.

Post-deployment, infrastructure management is primarily about two things: reliability and cost.

Scaling Your Database

As your user base grows, your database needs to handle more data and more people using it at the same time. Supabase manages most of this for you — it can automatically scale its resources up and down based on demand.

What you need to watch for:

- Slow data lookups. When your app had 100 users, loading a list was instant. With 100,000 users, the same lookup might take seconds. The fix is usually adding "indexes" — think of them like a table of contents for your database that helps it find things faster. Supabase's dashboard shows you which lookups are slow so you know where to add them.
- Storage growth. Database storage costs money. If you're storing data that you rarely need to access (old logs, archived records), consider moving it out of your main database. Set up data retention policies early.
- Too many simultaneous users. Every active user takes up a connection to your database, and there's a limit. Supabase handles this reasonably well out of the box, but if you're seeing connection errors, you may need to upgrade your plan.

FOUNDER MOVE

Review infrastructure once a week before it becomes urgent.

CHECKLIST

- [ ] Database dashboard reviewed
- [ ] Slow queries watched
- [ ] Storage growth understood
- [ ] Compute costs checked
- [ ] Backups enabled
- [ ] Incident notes kept

Scaling Compute

Vercel handles frontend and API scaling automatically — it spins up more resources as traffic increases and scales them back down when it drops. This is great for handling traffic spikes without manual intervention.

The catch: this can get expensive quickly at scale. Vercel charges based on usage, and if you're not careful, a traffic spike (or a bot repeatedly hitting your app) can run up a significant bill.

Strategies for managing compute costs:

- Cache aggressively. Caching means saving the result of an expensive operation so you can reuse it instead of re-computing it every time. Vercel has built-in tools for this — for example, serving a pre-built version of a page instead of regenerating it for every visitor.
- Rate limit your APIs. Put a cap on how many requests any single user (or bot) can make in a given time period. This protects against both abuse and runaway costs.
- Monitor your usage. Set up billing alerts in Vercel so you're never surprised by a bill.
- Optimize your functions. Shorter execution times = lower costs. If an API route is taking 3 seconds when it could take 200 milliseconds, that's money.

There's a classic founder dilemma here: do you throw money at infrastructure to move fast, or do you optimize ruthlessly for efficiency? The answer depends on your stage. Early on, optimize for speed — your time is more valuable than your server bill. As you scale, efficiency becomes critical because infrastructure costs can eat into your margins.

CHAPTER II / 13

What Comes Next

The best product plan is worthless if it stays in a document.

Building is not a one-time event. It's a continuous cycle of shipping, learning, and iterating. Once your MVP is live, the real work begins — watching how users interact with your product, identifying what's broken or confusing, and improving relentlessly.

In the next chapter, How to Sell, we'll cover how to take what you've built and get it in front of users — from building a brand to running your first sales campaigns to setting up marketing that actually works.

But first: ship something. The best product plan in the world is worthless if it stays in a document. Get your code into production, get it in front of real people, and start learning. Everything else follows from there.

FOUNDER MOVE

Pick five people in the wedge market and ask them to react to the live product.

CHECKLIST

- ○ Live URL
- ○ Demo path
- ○ First outreach list
- ○ Feedback capture
- ○ Next product iteration

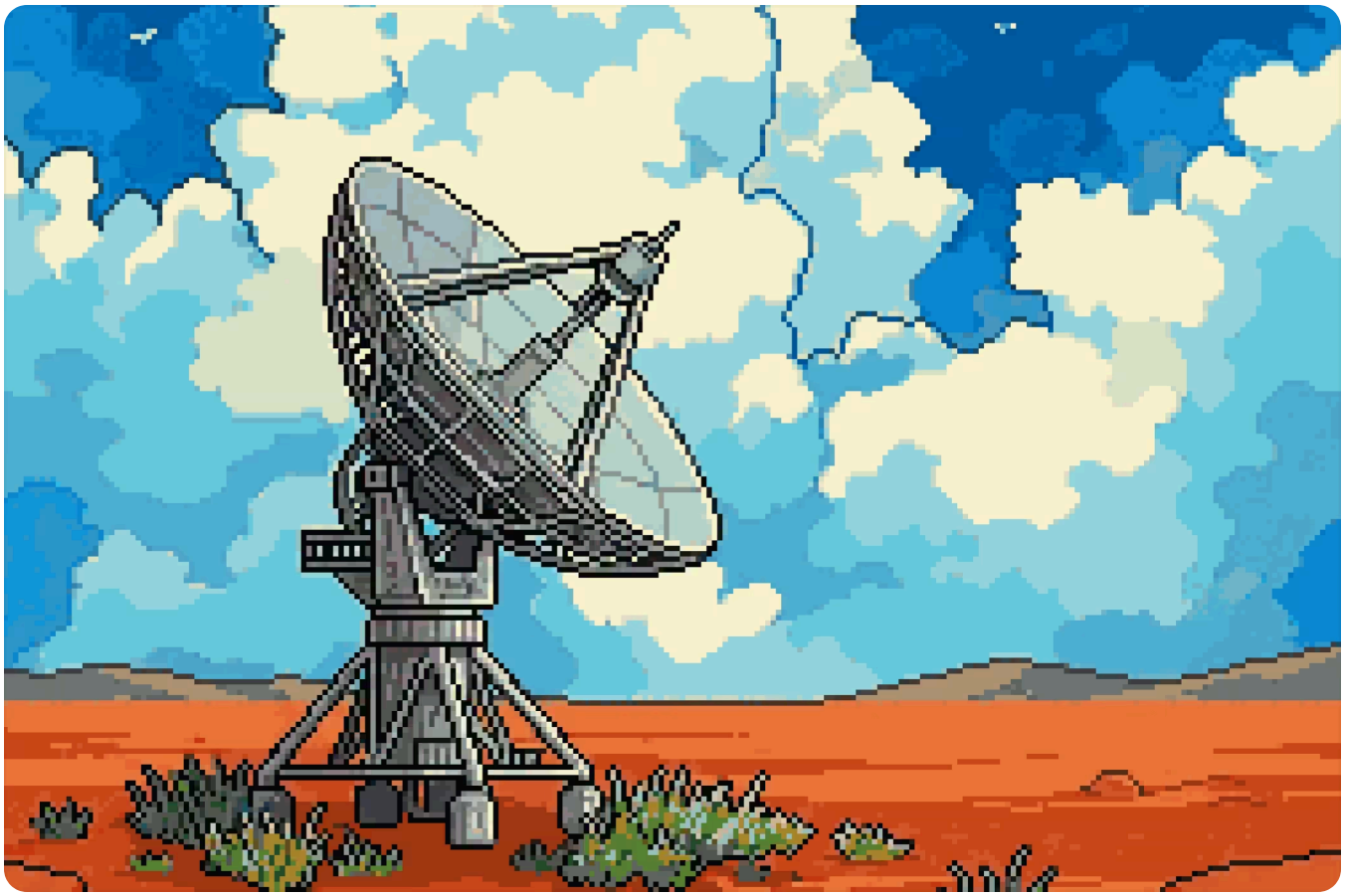
CHAPTER III

III

How to Sell

Build distribution: brand, website, first users, ICP, competitors, sales motion, workflow, process, marketing, and story.

- Marketing helps people discover you. Sales turns the right people into customers.
- Specificity beats volume early.
- Every customer conversation should improve the product and the story.



CHAPTER III / 01

Introduction

Distribution is everything once the product exists.

Now that you've built an MVP, you need to get users.

Distribution is everything. You cannot launch a product, send a few cold emails, and expect users to show up. You need a brand, a website, a sales process, and a way to create demand over time.

Sales and marketing are the two sides of distribution. Marketing helps people discover you. Sales turns the right people into customers.

This chapter is about building that system.

FOUNDER MOVE

Treat distribution as a product you build, test, and improve.

CHECKLIST

- ○ Brand
- ○ Website
- ○ First user plan
- ○ ICP
- ○ Competitor map
- ○ Sales workflow
- ○ Marketing loop

CHAPTER III / 02

Build a Brand

A brand is the story people attach to your company.

You need a brand.

A brand is not just a logo. It is the story people attach to your company. It tells customers why you exist, what you believe, and whether they should trust you.

A BRAND HAS THREE PARTS:

1. The story of the company. Why does the company exist?
2. A set of values. What do you believe that should show up in the product?
3. A visual identity. What does the company look and feel like?

WATCH OUT

Do not write values that sound like corporate posters. Values should shape product and marketing decisions.

CHECKLIST

- ○ Origin story
- ○ Problem belief
- ○ Values
- ○ Visual direction
- ○ Voice
- ○ Proof points

The story

Start with story — start with why. Uber's early story was simple: someone waited 20 minutes for a cab and realized the system was broken. The problem was obvious, emotional, and easy to repeat.

Your story doesn't need to be dramatic, but it does need to be a story.

Example — General Intelligence brand story

"Our branding — with the sunflowers, lush greenery, and people spending time with their friends — reflects our vision for the world. That's the world we want to build. A world where people actually work less and can spend time doing the things they love.

We're going to make it easy for anyone to start a company and build that life for themselves. The life they want to build, and spend every day dreaming about."

Maybe you watched finance teams spend every Monday reconciling spreadsheets. Maybe you saw clinics lose money because insurance claims were impossible to track. Maybe you kept building the same internal tool at every job because the market options were terrible. That is the seed of the brand.

Your brand values should be just as clear. They should not sound like corporate posters. They should shape product and marketing decisions.

EXAMPLES OF BRAND VALUES:

- Nike: Sport should move the world forward and be accessible to everyone.
- Airbnb: Travel should help people feel like they belong anywhere, not just book a place to sleep.
- Patagonia: A company can build great products while putting the planet first.
- Stripe: The internet should make it easier for more businesses to participate in the global economy.
- Shopify: Anyone, anywhere should be able to start and grow a business.
- Apple: Privacy should be built into products because it is a fundamental human right.

Visual identity

The visual identity is how the brand shows up — your vibe. It does not need to be perfect. It needs to feel intentional and consistent.

ELEMENTS OF A VISUAL IDENTITY:

- A logo
- A color palette
- Fonts
- Button and layout styles
- Image, illustration, or screenshot style
- Basic rules for how everything should look together

CHAPTER III / 03

Build a Website

Your website is the first salesperson that works while you sleep.

Once you understand your brand, you need a website.

Your website is the home base for distribution. Sales emails link to it. Customers use it to understand what you do. Marketing campaigns send people there. Investors look at it before they take you seriously.

The first thing your website should do is tell any reader what your company does. After that, it can represent your brand and is often the "front door" of your company.

A GOOD EARLY WEBSITE SHOULD ANSWER:

- Who is this for?
- What painful problem does it solve?
- What does the product do?
- Why is this better than the current way?
- What should the visitor do next?

FOUNDER MOVE

Read the homepage aloud. If it could describe ten companies, make it sharper.

CHECKLIST

- [ ] Specific headline
- [ ] Customer pain
- [ ] Product promise
- [ ] Screens or workflow
- [ ] Proof
- [ ] CTA
- [ ] Contact path

For a live product, the next step might be Sign up, Book a demo, or Log in. For an earlier product, it might be Join the waitlist or Talk to us.

Do not overcomplicate the first version. You need a clear homepage, a simple product explanation, a few screenshots or visuals, and a call to action. If you run a software company, it should also convert readers into signups (self serve) or demos (enterprise).

CHAPTER III / 04

Get Your First User

The first user is not a metric. It is a learning event.

If you followed along with chapter 2, you should now have a product. Since you have a product, you need someone to use it.

Start with people close enough to give real feedback — friends, former coworkers, industry contacts, advisors, or people you know through your network. The important thing is that they understand the problem and will not immediately shut you down.

Do not just send a link and ask, "What do you think?" Watch them use the product if you can. Ask them to share their screen. Sit next to them. Pay attention to where they hesitate, what they misunderstand, and what they try to do that your product does not support yet.

YOU ARE TRYING TO LEARN:

- **Do they understand what the product is for?**
  Watch for confusion in the first 60 seconds. If a user cannot explain what the product does after a brief look, the positioning is unclear.
- **Can they reach the first valuable moment?**
  Identify the moment the product first delivers real value. Then count how many steps it takes to get there. Every extra step is a drop-off risk.
- **Where do they get confused?**
  Note every hesitation, misclick, and misread label. Confusion is data — each instance points at copy, UX, or model mismatches you can fix.
- **What would make them use it again?**
  Retention starts on day one. Ask what would pull them back tomorrow, next week, next month — and whether the product currently delivers that.
- **Would they pay for it?**
  Willingness to pay is the strongest signal. Ask directly. A polite "maybe" is a no — push for a real answer.
- **Who else has this problem?**
  If they say yes to paying, ask who else they know with the same problem. That is your first lead list — and the start of an ICP.

WATCH OUT

A polite compliment is not validation. Look for behavior: time, money, introductions, or repeated usage.

CHECKLIST

- [ ] List 20 reachable prospects
- [ ] Write a plain-language ask
- [ ] Offer to show the product
- [ ] Watch usage
- [ ] Capture objections
- [ ] Ask for referrals

Take the feedback seriously, but do not obey every request. Users are good at revealing problems. They are not always good at designing the solution.

Building a good company is a function of how well you solve your users' problems. Your first users also help refine your ICP.

CHAPTER III / 05

Define Your Ideal Customer

Good sales starts with good targeting.

Most people start too broad. They say they sell to founders, marketers, realtors, doctors, developers, small businesses, or enterprises. These categories are too vague for sales. You cannot write good outreach, build a strong pitch, or prioritize leads if your customer is "everyone who might maybe use this."

You need an Ideal Customer Profile, or ICP — the highly specific kind of customer you are trying to sell to first. It is not every possible customer. It is the customer most likely to feel the problem, understand the value, and buy early.

FOUNDER MOVE

Write the ICP so clearly that you know where to find 50 of them.

FROM VAGUE TO USEFUL

Too broad: ecommerce companies. Specific: DTC brands doing $5M-$50M in revenue that rely on repeat purchases and have a small retention team.

TOO BROAD

We sell to ecommerce companies.

SPECIFIC

We sell to DTC brands doing $5M–$50M in revenue that rely on repeat purchases and have a small retention team.

The second version tells you where to look, who to message, what pain to lead with, and what language to use.

To define your ICP, start with the broad category, then narrow it by company size, industry, growth stage, geography, workflow, budget, and trigger event. A trigger event is something that makes the problem urgent: a company just raised money, a new leader joined, a team is hiring, a regulation changed, a bad process finally broke.

Good sales starts with good targeting. If your ICP is wrong, everything downstream gets harder.

CHAPTER III / 06

Understand Your Competitors

Competitor research is not about copying. It is about finding the opening.

Competitors are not proof that you should quit. They are proof that customers already care about the problem. The mistake is copying them.

If you copy their website, features, pitch, and customer list, you become a weaker version of something that already exists. The goal of competitor research is not to blend in. The goal is to find the opening.

Search the main keyword in your category and study the first few serious companies you find.

LOOK AT:

- Who their homepage speaks to
- What pain they lead with
- Which features they highlight
- What industries they mention
- How they price
- What their reviews complain about

Then ask: what are they missing? Maybe they serve enterprises and ignore smaller teams. Maybe they are powerful but hard to use. Maybe customers like the outcome but hate the workflow. That gap is where you should position yourself.

WATCH OUT

Being different is not enough. Be different in a way the customer values.

CHECKLIST

- [ ] Positioning
- [ ] Pricing
- [ ] Features
- [ ] Customer segment
- [ ] Reviews
- [ ] Weaknesses
- [ ] Switching trigger

CHAPTER III / 07

Choose Your Sales Motion

The sales motion should match the price, pain, buyer, and trust required.

A sales motion is how you sell. The right motion depends on your customer, price, product complexity, and how much trust the buyer needs before they pay.

THERE ARE THREE QUESTIONS TO ANSWER:

- [ ] **Transactional or consultative?**
  A transactional sale is simple — the buyer understands the product, the price is low enough to decide quickly, and they can usually buy without talking to you. A consultative sale requires a conversation.

- [ ] **Product-led or relationship-led?**
  A product-led sale depends mostly on the product experience. A relationship-led sale depends more on trust.

- [ ] **Inbound or outbound?**
  Inbound means customers come to you. Outbound means you go to them. Most startups are outbound first because nobody knows they exist yet.

A SIMPLE RULE:

- Cheap, obvious, low-risk products can be self-serve and product-led.
- Expensive, complex, high-trust products need consultative sales.
- If nobody knows you exist yet, you need outbound.

Choose the simplest motion that can close your first customers.

FOUNDER MOVE

Match the sales process to the customer's risk, not your preference.

CHECKLIST

- Price point
- Buyer seniority
- Sales cycle
- Trust required
- Demo needed
- Implementation effort
- Expansion potential

SALES MOTION

Choose based on how hard the purchase is and whether buyers are already looking.

	SIMPLE / LOW RISK	COMPLEX / HIGH TRUST
BUYERS ARE SEARCHING	PRODUCT-LED INBOUND **Simple product, low risk, buyers already search.** Clear website, pricing, signup, content.	SALES-ASSISTED INBOUND **Complex product, high trust, buyers already search.** Demo requests, discovery, proof, proposal.
YOU NEED OUTBOUND	LOW-TOUCH OUTBOUND **Simple product, clear ICP, buyers are not looking yet.** Targeted emails, short calls, trial links.	FOUNDER-LED CONSULTATIVE **Complex product, expensive or risky, demand must be created.** Named accounts, warm intros, deep discovery.

Startups usually begin in the outbound row because nobody knows they exist yet. Move toward inbound and product-led when the product, market, and trust signals support it.

CHAPTER III / 08

Set Up the Sales Workflow

A sales workflow turns conversations into a system you can improve.

Sales needs a system. The first version does not need to be complicated — it needs to help you track who you are selling to, what happened, and what happens next.

WATCH OUT

If a deal can sit in a stage forever, the stage is not objective enough.

CHECKLIST

- [ ] CRM created
- [ ] Stages defined
- [ ] Lead source captured
- [ ] Next step required
- [ ] Objections logged
- [ ] Closed-lost reason captured

OUTBOUND CADENCE

A sales workflow turns one lead into a sequence of clear next actions.

Day 1	EMAIL	Context, problem, solve
Day 2	SOCIAL TOUCH	Become familiar
Day 3	EMAIL	Answer the likely objection
Day 4	SOCIAL TOUCH	Show real engagement
Day 5	EMAIL	Share the differentiator
Day 6	CONNECTION / DM	Make it relevant
Day 7	EMAIL	Soft walkaway

REPEAT RULE

Run the sequence up to 3x with different decision makers.

LATER FOLLOW-UP

After a month, use their news. After two, restart the cadence.

Set up a simple CRM

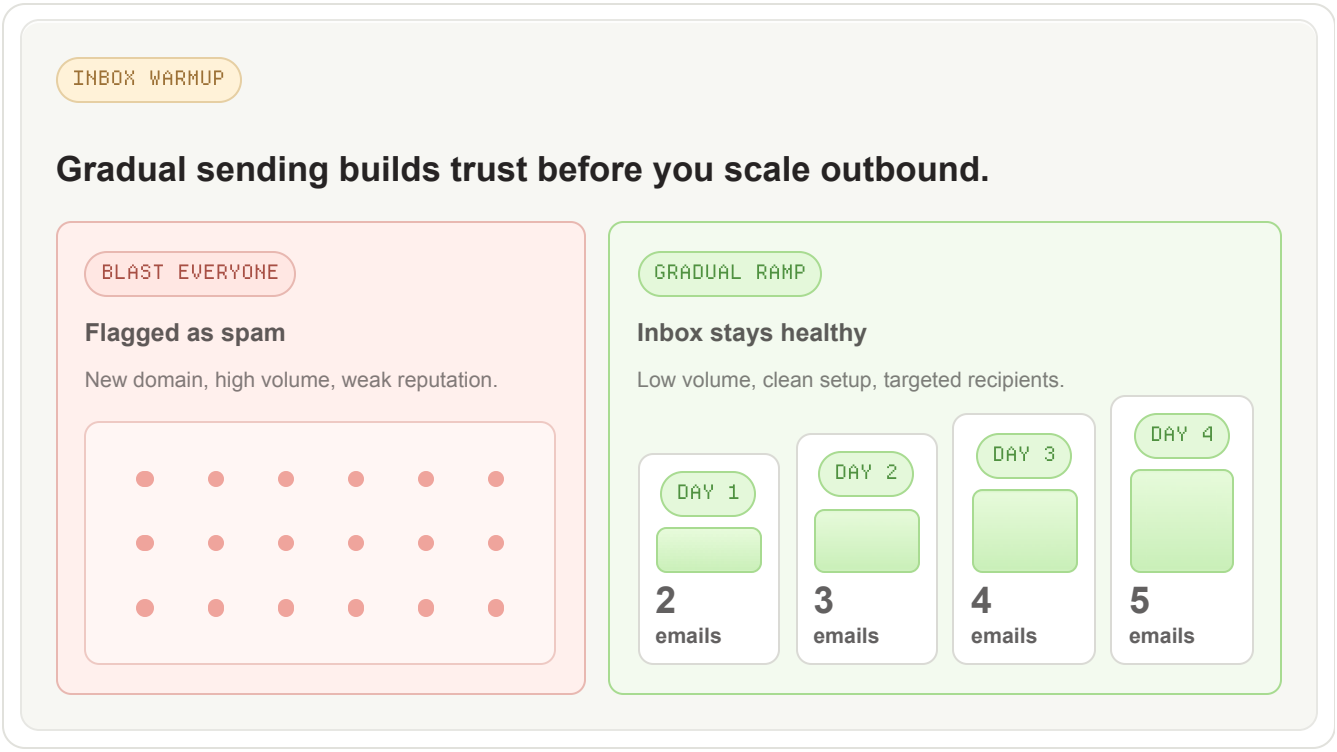
A CRM is where you track customers. A structured spreadsheet can work. A lightweight CRM can work. What does not work is trying to remember every conversation in your head.

YOUR CRM SHOULD ANSWER:

- Who are we selling to?
- Who is the decision maker?
- What stage is the deal in?
- What did we last discuss?
- When do we need to follow up?
- Why did we win or lose?

Warm up your inbox

Cold email only works if your email lands in the inbox. New domains have no reputation. If you create a domain and immediately send hundreds of emails, email providers may treat you like spam.



BEFORE OUTBOUND, SET UP THE BASICS:

- Use a real business domain.
- Configure SPF, DKIM, and DMARC.
- Send a small number of targeted emails first.
- Ramp volume slowly.
- Keep bounce rates low.
- Make it easy for people to opt out.

Build a lead list

A lead list is a list of companies or people who might buy. Start with the ICP, then look for evidence that a company has the problem now. Hiring posts, funding announcements, product launches, new executives, regulatory changes, industry news, and public complaints can all be useful signals.

A GOOD LIST IS:

- Relevant. The leads match your ICP.
- Repeatable. You can build the same kind of list again.
- Reachable. You can find a real person and a real channel to contact them.
- Rare. It is not the exact same list every other seller is blasting.

THEN SCORE THE LEADS:

1. Tier 1: Strong ICP fit, clear trigger, likely to buy soon.
2. Tier 2: Good ICP fit, some signal, worth contacting.
3. Tier 3: Possible fit, but unclear need or long sales cycle.

Spend most of your time on Tier 1 and strong Tier 2 leads.

Choose the right decision maker

The ICP tells you which companies to target. The decision maker tells you who to contact. The right person is not always the most senior person — it is the person who feels the pain, understands the workflow, and can influence the purchase.

LOOK FOR:

- Problem fit. Who actually experiences the problem?
- Budget or influence. Who can approve, recommend, or block the purchase?
- Timing. Has something changed that makes them more likely to act?
- Familiarity. Have they used a similar product before?

Do not spend forever finding the perfect contact. Pick the best person, test the message, and move.

Do outreach that does not suck

Nobody wants to engage with spam. Good outreach is casual, clear, concise, and contextual.

TOO BROAD

Hi Michael,
I hope this email finds you well. I am reaching out because our revolutionary platform helps modern organizations unlock operational efficiencies across the enterprise.

SPECIFIC

Michael — saw your team is hiring three lifecycle marketers.
Usually that means retention is becoming a bigger priority, but the reporting stack gets messy fast.
We help DTC teams see which campaigns actually drive repeat purchase without stitching together five dashboards.
Worth a quick look next week?

This works because it references a trigger, names a problem, says what the product does, and asks for a small next step. Cold email is useful, but it works best when the prospect can also find a real website, a real point of view, and real signs that your company understands the market.

CHAPTER III / 09

Run the Sales Process

Sales is a sequence of specific next steps.

A pipeline is the path a deal moves through from lead to customer. The stages should reflect real progress — a deal should only move when something objective has happened.

A SIMPLE EARLY PIPELINE:

1. Open — lead added.
2. Trying to Contact — outreach started.
3. Contacted — prospect replied.
4. Consult — discovery call scheduled or completed.
5. Pitch — you presented a specific solution or proposal.
6. Verbal Commit — buyer agreed in principle.
7. Closed Won — contract signed or payment received.
8. Closed Lost — deal is not moving forward.

Review the pipeline weekly. Ask what needs follow-up, what is stuck, what should be closed lost, and which deals deserve more attention.

Run a consult call

A consult call is where you learn enough to decide whether you can help. Do not treat it like an interrogation. The best consult calls feel like real conversations — you are listening for pain, urgency, authority, budget, and fit. You are also qualifying them. Not every prospect should become a customer.

YOU WANT TO LEARN:

- What problem are they trying to solve?
- Why does it matter now?
- How do they handle it today?
- What is broken about the current approach?
- What happens if they do nothing?
- Who else is involved in the decision?
- What would a successful outcome look like?

Develop a strong pitch

A pitch is not a deck. A deck can help, especially in a consultative sale. But the pitch is the story you tell about the customer, their problem, and the better outcome you can help create.

A STRONG PITCH USUALLY FOLLOWS THIS STRUCTURE:

1. Here is what we heard. Repeat the customer's problem in their language.
2. Here is why it matters. Connect the pain to time, money, risk, growth, or opportunity.
3. Here is what changes. Show the better workflow or outcome.
4. Here is how the product helps. Explain the product only as much as needed.
5. Here is proof. Use examples, testimonials, data, or a clear demo.
6. Here is the next step. Make the path forward obvious.

Talk about outcomes more than features. Customers do not buy a dashboard because it has filters. They buy it because it helps them see which campaigns are wasting money.

Close the deal

Closing is not a trick. If you have targeted the right customer, understood the pain, built trust, and pitched a useful solution, closing should feel like the next logical step. A good closing process makes buying feel safe.

MAKE SURE YOU HAVE:

- A clear price
- A clear scope
- A clear buyer
- A clear approval process
- A clear timeline
- A clear onboarding plan
- A clear next meeting or signature date

Re-establish the business case before talking terms. Then make the next step concrete.

FOUNDER MOVE

Every conversation should end with a calendar event, a clear no, or a concrete next action.

BETTER CLOSE

Weak: "Let me know what you think." Strong: "Can we send the order form today and schedule onboarding for next Tuesday?"

NOT A CLOSE

"Let me know what you think."

A CLOSE

"Can we send the order form today and schedule onboarding for next Tuesday?"

CHAPTER III / 10

Improve the Sales Process

The sales process is a diagnostic system for the company.

Once you have a few dozen real conversations, start looking at the process. Your CRM is how you diagnose where sales is breaking.

LOOK AT THE CONVERSION BETWEEN STAGES:

- Leads to contacted — tells you whether your list, copy, and sending setup are working.
- Contacted to consult — tells you whether the message is relevant enough to earn a meeting.
- Consult to pitch — tells you whether you are reaching the right ICP and finding real pain.
- Pitch to verbal commit — tells you whether the story and offer are strong.
- Verbal commit to closed won — tells you whether pricing, procurement, trust, and onboarding are clear.

Do not obsess over tiny numbers. Early data is noisy. But review the process monthly and look for patterns. Sales improves the same way product improves: observe, learn, change one thing, and measure again.

WATCH OUT

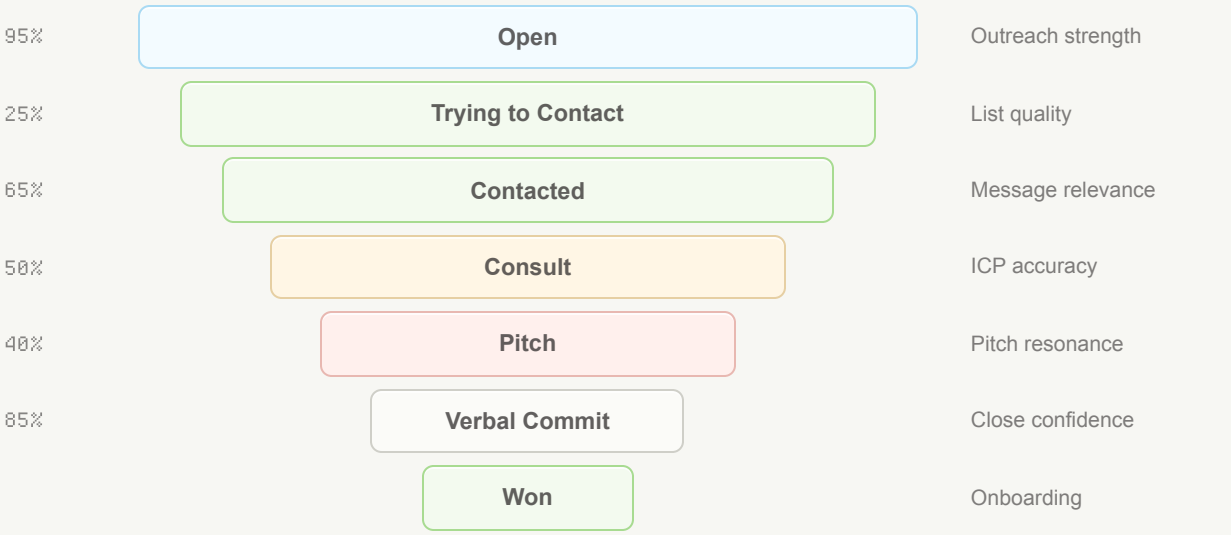
A full pipeline is not useful if the accounts are wrong.

CHECKLIST

- Reply rate
- Meeting rate
- Qualified rate
- Close rate
- Sales cycle
- Top objections
- Best segments

CONVERSION FUNNEL

Stage conversion shows which part of the sales process needs work.



WATCH THE LEAKS

A weak stage points to one sales habit to improve.

FIX ONE THING

Change one variable, then measure the next month again.

How to do Marketing

Marketing starts by turning your point of view into useful content.

Marketing creates awareness and generates warm leads. Sales converts those leads into customers.

A successful business usually does both. In the beginning, consumer companies tend to be more marketing-heavy. Business-facing companies tend to be more sales-heavy. But even B2B companies need marketing. Cold outreach works better when people can see that your company has a point of view.

Marketing starts by turning your brand into content. Content is not just posting for attention — it is how you teach the market what you believe, explain the problem, show your product, and make the company easier to remember.

FOUNDER MOVE

Make one excellent piece for the ICP before trying to post everywhere.

CHECKLIST

- Audience
- Channel
- Point of view
- Content format
- Distribution plan
- Conversion path
- Learning metric

Choose the content

Before you make content, decide what kind of content fits your market. The right content depends on your customer. Developers may want technical guides. Operators may want templates. Executives may want benchmarks. Consumers may want short videos or social proof.

EXAMPLES OF CONTENT TYPES:

- Essays or blog posts
- Images and social posts
- Short videos
- Educational guides
- Templates and checklists
- Product demos
- Customer stories
- Benchmark reports
- Micro-apps — small free tools that solve a narrow problem and bring the right audience into your orbit

Make the content

Most startup content is bad because it is rushed, generic, or written for no one in particular. One useful guide that says something specific is better than ten vague posts about "the future of work."

Good content usually has one clear idea. It can teach something, show a workflow, explain a customer pain, compare approaches, launch a feature, tell a founder story, or give away a useful tool.

You can use AI tools to produce the assets. Image models like gpt-image-2 can help generate campaign visuals. Midjourney can help explore art direction. HyperFrames can help create motion videos from web-based compositions. HTML can be used to make social assets, product visuals, and decks.

The important thing is not the tool. The important thing is taste. Refine the output until it looks like something your company should actually publish.

Pick a strategy

Do not try every marketing channel at once. Pick a strategy that matches your market and your strengths.

- SEO works when customers are already searching for the problem. Slow, but compounds.
- Paid works when you know who the buyer is and can afford to test messages quickly. Fast feedback, but can burn money if the funnel is weak.
- Organic works when your market spends time on social platforms, communities, newsletters, or forums. Builds trust, but requires consistency.
- Influencer or partnership marketing works when trusted people or companies already have access to your audience. You borrow trust, but need to choose partners carefully.

Pick one primary channel and one supporting channel. Run them seriously for long enough to learn.

Distribute the content

Making content is not enough. You have to put it where people will see it — publishing on your blog, posting on X or LinkedIn, uploading to YouTube or TikTok, sending a newsletter, launching on a community, running ads, or giving partners assets to share.

The platform should match the customer. Do not post everywhere because you feel like you should. Post where your ICP already spends attention.

You will need to create the accounts yourself. Use the same brand identity across them so the company feels consistent.

Measure and refine

Marketing should create a learning loop. Do not confuse attention with progress. A post can get a lot of likes and produce no customers. A niche guide can get fewer views and produce three excellent leads.

TRACK THE BASICS:

- Which content gets attention?
- Which content brings qualified visitors?
- Which visitors become leads?
- Which leads become meetings?
- Which meetings become customers?

The goal is not to be famous. The goal is to create demand from the right people.

CHAPTER III / 12

Tell a Consistent Story

Markets remember repetition.

Sales and marketing both depend on story. Not fake storytelling. Not manipulation. A good story is a clear explanation of why the customer should care now.

The customer is the main character, not you.

A SIMPLE STORY HAS FIVE PARTS:

1. The customer has a goal.
2. Something is getting in the way.
3. The old way is no longer good enough.
4. Your product creates a better path.
5. The customer gets a better outcome.

WATCH OUT

If customers cannot describe what you do after a conversation, your story is not simple enough yet.

CHECKLIST

- Same customer
- Same problem
- Same promise
- Same proof
- Same next step

This story should show up everywhere: your website, outreach, consult calls, pitch, demo, proposal, content, ads, and onboarding.

If the story changes every time you tell it, the market will feel confused. If the story is clear, customers start repeating it back to you. That is a strong signal.

WHAT COMES NEXT:

Once you've got this down, you'll have to start scaling functions like customer support, product analytics, and others. That's what Chapter IV covers.

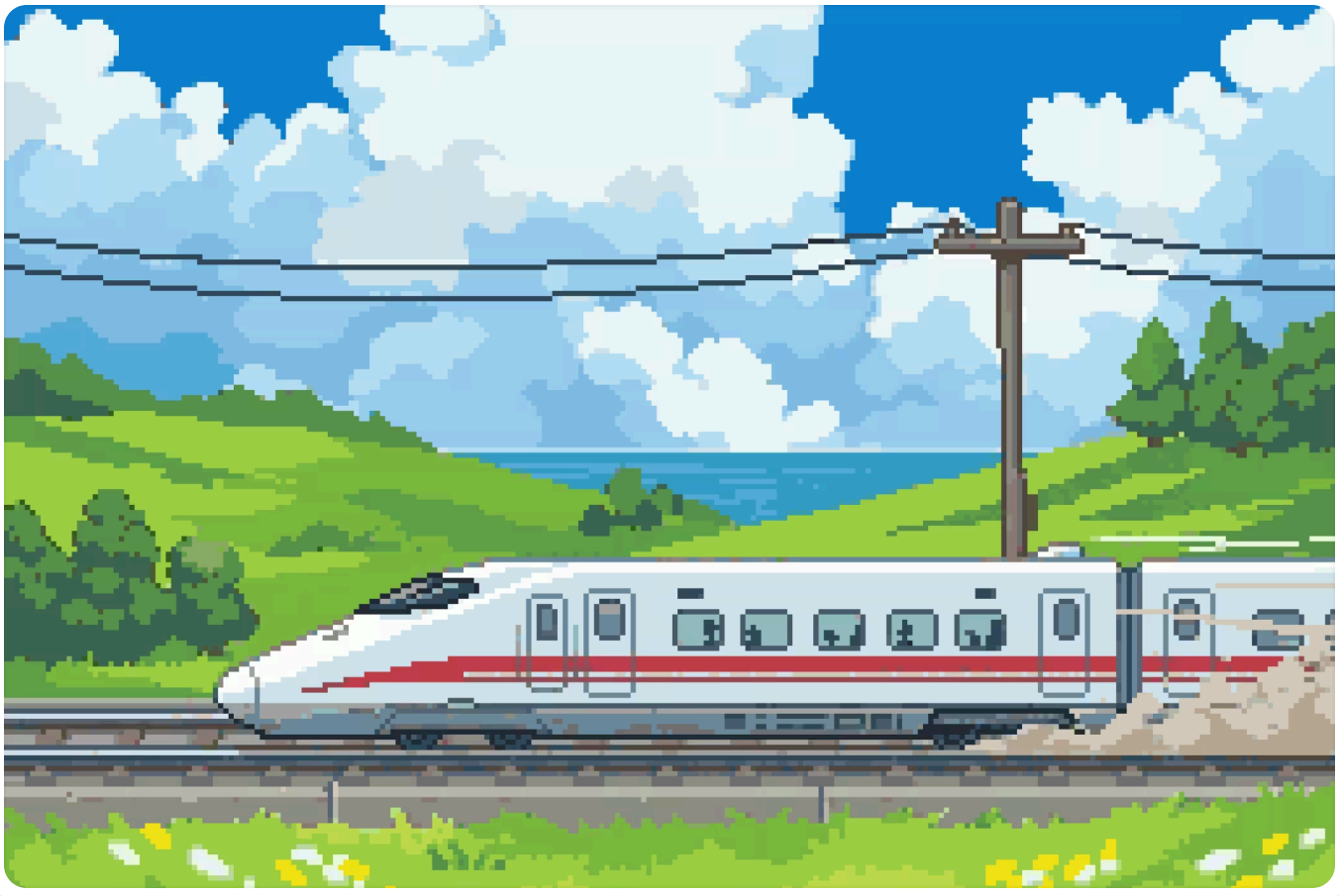
CHAPTER IV

IV

How to Scale

Turn a working product into a business by improving the loops: analytics, support, unit economics, expansion, and operating cadence.

- Scaling starts with the loop you already proved.
- Support, analytics, and revenue data should inform what you build next.
- Expansion should follow evidence, not boredom.



CHAPTER IV / 01

Introduction

Scaling starts with the loop you already proved: build, sell, learn, improve.

Now you've had some early success, have a product, and have some customers. Now it's time to scale.

Scaling starts with the loop you already proved: build, sell, learn, and improve. Your job now is to make that loop faster, cleaner, and more durable. You need to understand what users do, where they get stuck, why they leave, which customers are worth acquiring, and what to build next.

This chapter covers the systems you need once your product has real users: product analytics, customer support, unit economics, and expansion.

FOUNDER MOVE

Write the loop for your company in one line: build what, sell to whom, learn from what, improve which metric.

CHECKLIST

- ○ Usage signal
- ○ Customer feedback
- ○ Support themes
- ○ Revenue data
- ○ Product roadmap

CHAPTER IV / 02

Start With the Scaling Loop

Growth without a learning loop turns into noise.

A startup is not a static product. It is a loop.

You build something. You sell it. Users try it. Some get value. Some get confused. Some leave. Some ask for features. You take that signal, decide what matters, and build again.

THAT IS THE SCALING LOOP:

1. Build — ship product changes that solve real user problems.
2. Sell — get the product in front of the right people.
3. Observe — watch what users actually do.
4. Support — help users when they get stuck.
5. Learn — find the patterns behind the behavior.
6. Iterate — improve the product, positioning, pricing, or onboarding.

WATCH OUT

Do not scale acquisition before you know what kind of customer retains.

CHECKLIST

- ○ Acquisition
- ○ Activation
- ○ Core value
- ○ Retention
- ○ Expansion
- ○ Learning

Early on, you can run this loop manually. You can talk to every user, watch onboarding sessions, and fix bugs in real time. You should do some of this — it gives you a feel for the customer that no dashboard can replace.

But manual observation eventually breaks down. You need systems that capture signal as usage grows. Not just servers and databases, but the operating systems that help you understand, support, and improve the business.

CHAPTER IV / 03

Add Product Analytics

Analytics keeps you honest about what users do, not what they said they would do.

You cannot iterate well if you do not know what is broken.

Product analytics tells you what users are doing inside your app: how many sign up, how many complete onboarding, which features they use, where they drop off, and whether they come back. Without analytics, you are guessing. Analytics does not replace user conversations, but it keeps you honest.

We recommend PostHog for this. It gives you funnels, custom events, retention analysis, and session replay in one place.

FOUNDER MOVE

Pick one drop-off, watch sessions, talk to users, ship a small fix, then measure again.

CHECKLIST

- Core events defined
- Funnels created
- Retention viewed
- Replay sampled
- Segments compared
- Metric owner named

Set Up PostHog

Setting up product analytics has two parts: installing the tool and deciding what to measure. The installation is straightforward — create a PostHog project, add the library to your app, include the project key and host in your environment variables, and identify users after they log in.

The measurement part is where founders usually get it wrong. They either track almost nothing or track everything. Both are bad. Start with the few events that show whether a user is moving through your product successfully:

- signed_up
- onboarding_completed
- core_action_completed
- trial_started
- checkout_completed
- subscription_upgraded
- teammate_invited
- support_ticket_opened

Use clear names, keep staging and production data separate, and do not track sensitive information like passwords, payment details, private documents, or API keys.

Funnels, Events, Replay, and Retention

- A funnel is a sequence of steps you expect users to complete. A funnel shows where people drop off. If 1,000 users sign up and only 150 create a first project, the problem is activation, not traffic.
- A custom event is a meaningful action you decide to track. Track outcomes, not just page views: project created, report generated, file uploaded, teammate invited, invoice sent.
- A session replay is a recording of how a user moves through your app. Analytics tells you where something broke. Session replay often shows you why.
- Retention tells you whether users keep coming back. The question is not "do users come back every day?" The question is "do users come back at the natural frequency of the problem we solve?"

A GOOD ANALYTICS LOOP:

1. Find a drop-off or unusual pattern.
2. Watch replays for that segment.
3. Talk to a few affected users.
4. Form a hypothesis.
5. Ship a small change.
6. Measure whether the metric improves.

CHAPTER IV / 04

Add Customer Support

Support is not only a cost center. Early on, it is product research.

The users you fought to win have given you something valuable: trust. That trust has a limit.

There are only so many times a product can confuse someone before they leave. Customer support exists to catch friction before it becomes churn.

Early on, support should be founder-led. Watch your first customers use the product. Ask what they expected to happen. Fix bugs in real time. Listen to the words they use to describe the problem. This does not scale, but it teaches you what the product feels like from the customer's side.

Eventually, you need a system. Users will show up while you are asleep. They will ask questions you have already answered. They will hit bugs you cannot personally triage in real time. We recommend Fin by Intercom for this — an AI support agent that answers inbound questions using your documentation, product context, and support procedures, then escalates issues it cannot resolve.

WATCH OUT

If support themes never reach the roadmap, you are throwing away one of the best product signals you have.

CHECKLIST

- [ ] Support channel
- [ ] Help docs
- [ ] Common issue tags
- [ ] Escalation path
- [ ] Bug handoff
- [ ] Response expectations

How Fin Works

A support agent is only as good as the knowledge and procedures behind it. Fin can handle common questions like password resets, billing updates, teammate invites, import failures, plan changes, and basic troubleshooting.

It should not blindly handle refund disputes, security concerns, data loss, angry enterprise customers, or bugs affecting many users — those should escalate to a human or become product work.

Support automation is not about hiding humans from customers. It is about resolving common questions quickly and routing uncommon questions correctly.

Build a Knowledge Base

A knowledge base is the set of help articles, troubleshooting steps, and policies your support system uses to answer questions. Write docs in the language customers use, not the language your codebase uses. If users repeatedly ask the same question, either the product is unclear or the documentation is missing. Usually it is both.

START WITH THE BASICS:

- Getting started
- Core workflows
- Billing and plans
- Troubleshooting
- Account management
- Known limitations

Use Decision Trees

A support decision tree is a structured path for handling a category of issue. Decision trees make support consistent and make escalation cleaner — the person receiving the issue can see what has already been tried.

START WITH THE MOST COMMON AND PAINFUL CATEGORIES:

- Login problems
- Billing questions
- Import failures
- Permission issues
- Core workflow errors
- Cancellations
- Bugs that block users from getting value

CHAPTER IV / 05

Understand Unit Economics

Unit economics tell you whether growth is making the company stronger.

Revenue tells you how much money is coming in. Profit tells you how much you keep. Both matter, but they describe the business from far away.

To understand whether the business can scale, you need to look at the economics of a single customer: how much they pay, how long they stay, and how much they cost to acquire. This is called unit economics.

Growth can hide a broken business. You can increase revenue by spending aggressively on acquisition, but if every new customer costs more to acquire than they are worth, you are not scaling — you are buying revenue at a loss.

HERE ARE THE CORE METRICS:

WATCH OUT

A growth channel is dangerous if every customer costs more to acquire than they are likely to return.

CHECKLIST

- Monthly revenue
- Logo churn
- Revenue churn
- ARPU
- Estimated LTV
- CAC
- Payback period

Churn

The rate at which customers leave. Logo churn measures the percentage of customers who leave. Revenue churn measures the percentage of revenue that leaves. Be precise about the time period — a 5% monthly churn rate is very different from a 5% annual churn rate.

```
Revenue churn rate = lost recurring revenue during period / recurring revenue at start of period
```

Average customer lifetime

How long a customer stays before churning. The time period must match the churn rate — if monthly churn is 20%, average customer lifetime is about 5 months.

```
Average customer lifetime = 1 / churn rate
```

ARPU

Average revenue per user or customer.

```
ARPU = total revenue / number of customers
```

LTV

Lifetime value — how much revenue or gross profit you expect to earn from a customer over their lifetime.

```
LTV = ARPU × average customer lifetime
```

```
Gross margin LTV = ARPU × gross margin × average customer lifetime
```

CAC

Customer acquisition cost. Early CAC is easy to understate because founder time often looks free. It is fine to do founder-led sales early, but do not assume that motion will scale without cost.

```
CAC = sales and marketing cost during period / new customers acquired during period
```

LTV/CAC

Compares how much a customer is worth to how much it costs to acquire them. If LTV is lower than CAC, something is broken — you need to increase retention, raise pricing, improve margins, lower acquisition cost, or revisit your ICP.

```
LTV/CAC = lifetime value / customer acquisition cost
```

A useful related metric is payback period: how long it takes to earn back the money you spent acquiring a customer. If CAC is $600 and the customer generates $100 in gross profit per month, payback is about 6 months.

Do not obsess over precision too early. With small datasets, these numbers will be noisy. The point is direction. Are customers staying longer? Paying more? Costing less to acquire? If yes, the machine is getting healthier.

CHAPTER IV / 06

Expand the Business

Expansion should come from evidence that the current ceiling is too low or the adjacent opportunity is too strong.

Once the core product works, support is manageable, and your economics are improving, the next question is how to expand the ceiling of the business.

There are two broad paths.

Vertical expansion

Serve your existing customers more deeply. Add products, services, workflows, or infrastructure that make your product more central to the customer. Works best when customers already trust you, the adjacent pain is obvious, and the new product increases retention or ARPU.

Horizontal expansion

Serve a broader set of customers. Take what you built and adapt it to adjacent segments, use cases, or markets. Works best when the core technology applies to multiple customer types and the new segment has similar pain with a larger market.

The danger in both cases is distraction. Do not expand because you are bored. Expand because the evidence says the current ceiling is too low or the adjacent opportunity is too strong to ignore.

FOUNDER MOVE

Before expanding, write what gets weaker if the bet is wrong.

CHECKLIST

- Current segment retention
- Expansion request frequency
- Adjacent pain
- Revenue potential
- Operational complexity
- Strategic fit

BEFORE EXPANDING, ASK:

- **Where is demand already showing up?**
  Look at inbound requests, support tickets, and sales conversations. If customers keep asking for the same adjacent thing, that is a signal worth taking seriously.
- **Does this improve retention, ARPU, CAC, or payback?**
  Expansion should improve the unit economics, not just grow revenue. If it does not make customers more valuable or cheaper to acquire, it is probably a distraction.
- **Will it strengthen the core product or distract from it?**
  The best expansions make the core product better. The worst ones split engineering focus and slow down the thing that is already working.
- **Can we test demand before building the full thing?**
  Sell the expansion before you build it. Talk to customers. Get a verbal commit or a letter of intent. If nobody is willing to say yes before you build, the demand may not be there.
- **Does it meaningfully increase the total addressable market?**
  Investors call this TAM — the total revenue opportunity if you captured every customer in the market you are targeting. A small, healthy niche can be a good business. A large, expanding market can become venture-scale.

Fundraising may become relevant at this stage, especially if the expansion opportunity is clear but requires more capital than the business can fund on its own. Fundraising is not the goal of scaling. It is a tool. The goal is to build a durable business that creates value for customers and captures enough of that value to keep growing.

What Comes Next

A scaled company is still a company learning from customers.

Scaling is not a single milestone. It is a discipline.

Add analytics so you can see what users do. Add support so you can hear where they struggle. Understand unit economics so you know whether growth makes the business stronger or weaker. Then use that signal to build better product, sell to better customers, and expand with intention.

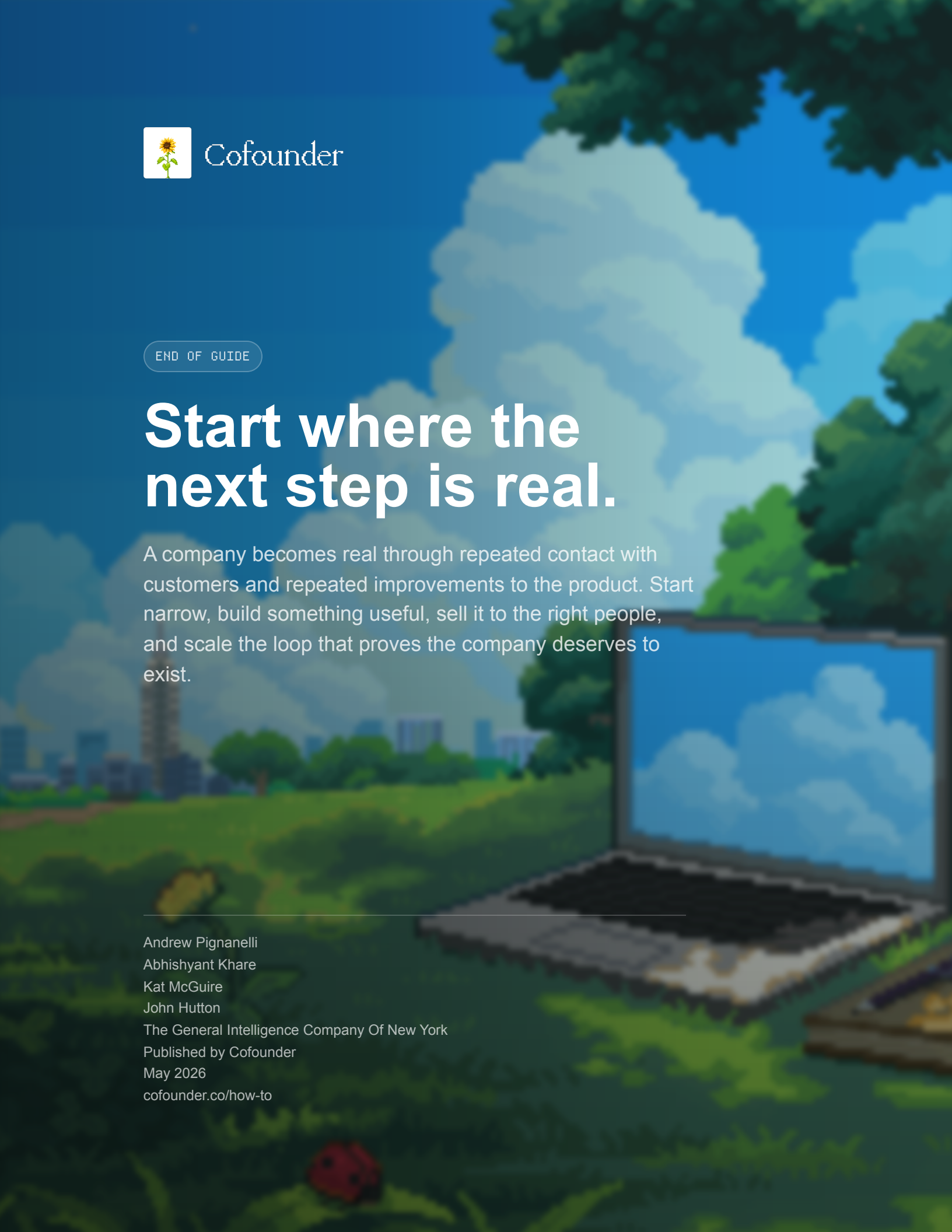
The company gets better when the loop gets better. Keep the loop moving.

KEEP IN MIND

The company gets stronger when every loop teaches the next loop what to do.

CHECKLIST

- Constraint identified
- Metric chosen
- Owner assigned
- Next experiment designed
- Review cadence set



Cofounder

END OF GUIDE

Start where the next step is real.

A company becomes real through repeated contact with customers and repeated improvements to the product. Start narrow, build something useful, sell it to the right people, and scale the loop that proves the company deserves to exist.

Andrew Pignanelli
Abhishyant Khare
Kat McGuire
John Hutton
The General Intelligence Company Of New York
Published by Cofounder
May 2026
cofounder.co/how-to