Code is not a business: what 20 paying customers taught me

Muhammad Gulzaib
https://orcid.org/0009-0000-8709-6989
SynergyFlow Labs · Islamabad, Pakistan

Published 2026-09-14
URL https://gulzaib.vercel.app/blog/code-is-not-a-business-what-20-paying-customers-taught-me
Licence © 2026 Muhammad Gulzaib. All rights reserved. Code samples MIT.

ABSTRACT

I built a POS SaaS, sold it to about 20 shops, then saw it solved the wrong problem. How I validate a SaaS idea now, and the checklist I use before coding.

Keywords: validation, saas, lessons, pos

> **TL;DR**
>
> I spent months building a point-of-sale SaaS, sold it to around 20 shops, and then watched usage stall because the market needed **offline-first**, not more features. Code is the easy part. Now I validate the *problem* before I write the first migration.

For most of my career, my instinct when I saw a problem was to open an editor. That instinct made me a good engineer. It also cost me the better part of a year on my first real product.

The product

The idea was simple: small retailers in Pakistan still run on paper registers or pirated desktop software. Give them a clean, cloud-based POS with inventory, receipts and daily sales reports, priced for a small shop. I knew Laravel and React well, so I could build it fast, and I did.

~20Paying shopsmostly from my own network

1Engineerme, nights and weekends

0↓Customer interviewsbefore writing code

That last number is the whole story.

What went wrong

Shops signed up. A few even paid annually. Then I started getting the same message in different words: *"The internet went out and we couldn't make a sale."*

Power cuts and patchy connections are normal for a lot of small businesses here. A cloud POS that freezes when the connection drops isn't a minor bug for a shop like that. It means a queue of customers at the counter and the owner going back to the paper register. Once that happened twice, they stopped trusting the system, and usage quietly fell off.

> The market didn't need a better POS. It needed one that worked when everything else didn't.

I could have found this out in a week by sitting in three shops and watching a normal day. Instead, I found it out after building authentication, multi-store inventory, role permissions and a reporting dashboard that almost nobody opened.

> **The real lesson**
>
> Customers judge software by how it fails as much as by its features. If the product can't survive the worst normal day in their business, the features don't matter.

Why engineers fall into this

Building feels like progress. Every merged pull request is a small win you can see. Talking to customers feels slow, awkward and uncertain, so we put it off and tell ourselves we'll "launch and learn".

The trouble is that by the time you launch, you've already made the expensive decisions: the architecture, the data model, the pricing, who the product is for. Offline-first, for example, is not something you add later. It changes how you store data, sync it and resolve conflicts from day one. (If I rebuilt it today I'd start with a local-first store (SQLite or IndexedDB) and a sync queue, and treat the server as a backup and reporting layer, not the source of truth at the counter.)

What I do now

Before I build anything for a new market, I run through this list. It takes one or two weeks, not months.

1. **Watch a normal day**

   Sit with three to five potential users while they do the job you want to improve. Don't pitch. Write down every workaround, every piece of paper and every moment they sigh.

2. **Find the worst normal day**

Ask: *"What happens when things go wrong?"* Power cuts, staff turnover, end-of-month rush. The worst normal day defines your real requirements.

3. **Sell before you build**

   Describe the product in one paragraph and ask for a commitment: a pre-payment, a letter of intent, or at least a scheduled pilot. Polite interest doesn't count as validation.

4. **Build the smallest thing that survives**

   Not the smallest thing that demos well. The smallest thing that keeps working on the worst normal day.

Decision · 2025 · Accepted

Validate the problem before writing the first migration

Context. I'm a fast builder, which makes it tempting to skip discovery.

Decision. No new product gets a repository until I've watched at least three real users and have one written commitment.

Consequences. Slower starts, far fewer dead products. Oxygen LMS and ZeroExam both started from school visits, not from code.

What the POS still gave me

It wasn't wasted. Twenty paying customers taught me billing, onboarding, support at 10 pm and how to explain software to people who don't care about software. Those lessons are the reason I could start SynergyFlow Labs, and they shaped everything that came after.

> **Key takeaways**
>
> - Code is the easy part. Distribution and problem fit are the hard parts.
> - Design for the worst normal day, not the demo.
> - Some requirements, like offline-first, are architectural. Find them before you build.
> - Count commitments, not compliments.

Next in this series: how cold-calling schools taught me more than any framework did.

*[Figure: image]***Distribution beats features: notes from cold-calling schools:** What cold-calling school principals taught me about selling software to schools: the script, the numbers, and why a rejected call beats a landing page.

Questions readers ask

How do you validate a SaaS idea before building it?

Watch three to five potential users do the job, find their worst normal day, and ask for a real

commitment such as a pre-payment, a letter of intent or a scheduled pilot before writing code. It takes one or two weeks, not months.

Why did the POS SaaS stall after 20 paying customers?

The shops needed offline-first software that kept working through power and internet cuts. That is an architectural requirement, so adding more features could not fix it.

What counts as validation for a startup idea?

Commitments, not compliments. Polite interest does not count; money, signed intent or a scheduled pilot does.

CITE AS

Gulzaib, M. (2026, September 14). Code is not a business: what 20 paying customers taught me. Gul's Notebook. https://gulzaib.vercel.app/blog/code-is-not-a-business-what-20-paying-customers-taught-me

© 2026 Muhammad Gulzaib. Quote short passages with a link and credit; please ask before republishing the full text.