WebSharthi

A WEBSHARTHI FIELD GUIDE

How Modern Companies Build Digital That Moves Industries

A practical guide to the decisions that separate software built to last from software built to demo — for founders and operators evaluating a technology partner.

Prepared by
WebSharthi

Based in
Delhi, India

Edition
2026

INTRODUCTION

Most software fails quietly, long before launch day

It rarely fails because the design was ugly or the pitch was weak. It fails because of decisions made in the first two weeks of a project — decisions that are easy to skip and expensive to undo.

This guide is a distillation of what we've learned building production systems for real businesses: hyperlocal commerce platforms, school ERPs, queue management systems for the grain trade, and internal tools that have to keep working long after the first invoice is paid.

It isn't a sales pitch dressed up as research. It's the same list of questions we ask ourselves before we write the first line of code on any project — and the same list we'd suggest you ask any technology partner before you sign with them.

> Who this is for: founders, operators, and product owners who are about to commission a website, an internal system, or a customer-facing application, and want to know what actually separates a durable build from an expensive-looking one.

Five ideas run through everything that follows. None of them are exotic. All of them are easy to skip when a deadline is close.

01 — PROCESS, NOT SCREENS

Start with the workflow, not the wireframe

It's tempting to begin a project by designing screens — they're the part everyone can see and react to. But a screen is just a window onto a process, and if the underlying process hasn't been mapped out, the screens end up rearranged three or four times before anyone touches them again.

What this looks like in practice

Before any interface work starts, we write out the actual sequence of steps a real user follows today — including the messy parts: the phone call that happens outside the system, the spreadsheet someone still keeps as a backup, the approval that technically has three people involved even though only one is "the user."

> Why it matters: systems designed around the paperwork trail instead of the real workflow tend to get abandoned within months — everyone quietly goes back to WhatsApp and Excel because the "digital" version didn't match how work actually happens.

Questions worth asking a technology partner

- Did they ask about your current process before proposing a solution, or did they go straight to a features list?
- Can they explain, in your own terms, what happens when something goes wrong mid-process — not just the happy path?
- Is there a plan for the edge cases that don't fit the main flow, or only for the ideal case?

02 — DATA REALITY

Build for the data you'll actually have

Most system designs quietly assume clean, complete, consistently-formatted data. Real businesses rarely have that — phone numbers with and without country codes, addresses that are really just landmarks, records that were entered twice by two different people last year.

What this looks like in practice

We ask for a real export of existing data — even a messy spreadsheet — before finalizing a database structure. It's the fastest way to find out that "customer name" is sometimes a shop name, sometimes a person's name, and sometimes both crammed into one field.

> Why it matters: a system built for perfect data breaks the first week it meets real data — and by then, the fix means migrating live records instead of adjusting a design document.

Questions worth asking a technology partner

- Did they ask to see your real, current data before designing the data model?
- Is there a plan for migrating your existing records, or does the new system assume a clean start?
- Who is responsible for data cleanup — you, or them, and is that written down anywhere?

03 — SEQUENCING

Ship a working core before adding features

Feature lists grow fastest in the planning stage, when everything is still theoretical and cheap to imagine. The projects that stay on budget are the ones where a small, real, working version ships first — even if it only handles the single most common case.

What this looks like in practice

We define the one workflow that, if it worked end-to-end tomorrow, would already be useful on its own — and we build that first, in front of real users, before adding the second and third features on top of it.

> Why it matters: a system that does one thing reliably is more useful — and more trustworthy — than a system that does ten things inconsistently. It's also the only way to find out early whether an assumption from the planning stage was wrong.

Questions worth asking a technology partner

- What's the very first version you'll actually be able to use, and when does that land?
- Is the plan to build everything and launch once, or to launch something small and expand it?
- How will you find out if an early assumption was wrong, before the whole system is built around it?

04 — AFTER LAUNCH

Make maintenance a first-class deliverable

A launch date feels like a finish line, but for any system that handles real business — orders, records, payments, approvals — the work that matters most starts the week after. Dependencies age, traffic patterns change, and something you didn't think to test in month one will surface in month four.

What this looks like in practice

Before a project ships, there should be a clear, written answer to: who gets notified if something breaks at 2am, how quickly, and what happens next. If that answer is "email us and we'll see it eventually," that's worth knowing before launch, not after.

> Why it matters: the gap between "the demo worked" and "it kept working for a year" is almost entirely about whether anyone was actually watching after launch.

Questions worth asking a technology partner

- What does support actually look like in month three — not in the sales conversation, in practice?
- Is monitoring/alerting built in, or is "someone will notice" the plan?
- Who owns the system's documentation, and can you get a copy of it?

05 — INFRASTRUCTURE

Choose infrastructure you can actually operate

It's easy to over-engineer a small business's first real system — microservices, multiple databases, elaborate cloud architecture — because it looks impressive in a proposal. Most of that complexity becomes a liability the day something needs fixing at 11pm and the person on call has to relearn the whole stack first.

What this looks like in practice

We size the infrastructure to the actual, current scale of the business — not the scale it might reach in five years — and design it so it can grow in stages, rather than guessing the end state upfront and paying for it from day one.

> Why it matters: the right architecture for a business processing a hundred orders a day is not the right architecture for one processing a hundred thousand — and building for the second when you're the first mostly just adds cost and fragility.

Questions worth asking a technology partner

- Is the proposed infrastructure sized for where you are today, or where a pitch deck imagines you in three years?
- Can it scale in stages, or does growth mean a full rebuild?
- Do you actually own your infrastructure and data, or is it locked into the vendor's own accounts?

W

WebSharthi

About this guide

WebSharthi is a Delhi-based development studio building web platforms, mobile apps, ERP systems, and AI-assisted tools for growing businesses. This guide reflects the questions we actually ask on our own projects — not a marketing checklist, a working one.

If you're evaluating a build right now and want a second opinion on a proposal, a data model, or an architecture decision before you commit to it — that conversation is free, and it comes with no obligation attached.

GET IN TOUCH

WebSharthi.com/connect

BASED IN

Delhi, India