How to keep engineers in their zone of excellence

Retool

Solving the frontend dilemma:

How to keep engineers in their zone of excellence

We know you've heard it before—that full-stack development as we know it is, actually, a myth because no dev is *really* an expert in the entire stack.

But that's an oversimplification.

What opinionated folks online miss *is* that there is a difference between an *intentional* full-stack engineer and a reluctant one: a full-stack engineer that lives to toggle between frontend and backend and a specialized engineer that's dragged from project to project, navigating a cognitive load that makes work feel ten times heavier.

Consider this scenario:

> "I'm a backend developer, currently constructing a frontend app in ReactJS... it took me 20 minutes to realize that a table with 1000 elements, each having edit buttons, led to the slow response of displaying a modal.
>
> Now, I'm tasked with finding an effective solution to address this issue—whether it's through pagination, using react-window, or exploring other alternatives. And I haven't even found the cause whether it's the amount of event handlers, the size of the virtual DOM, or something else.
>
> backend feels so much [more] straightforward."
> — erikvdven on Hacker News

When frontends feel so mired in complexity, even the most talented backend and full-stack engineers will struggle to keep up with the workload required.

We're not here to take a side on which discipline is harder or more worthwhile to learn. Whatever a developer's specialization, spending time outside of it is uncomfortable and inefficient (even for the most enthusiastic full-stack engineers). And yet, limited engineering headcount and misconceptions about frontend work being 'easier' means too many backend developers are still being asked to stretch their skills to time-consuming frontend work.

Teams are increasingly expected to do more full-stack work with fewer frontend resources.

For one recruitment firm specializing in seed-stage to Series C engineering recruitment, just 5% of roles they're hiring for in 2025 are pure frontend, vs 50% backend and 25% full-stack. Asher Hoffman, co-founder of Coastal Recruiting said, "The first three hires for a company with technical co-founders are almost always backend engineers. It's a bonus if they have some frontend skills, but I'd peg it at 80% backend, 20% frontend."

The pressure to make do with fewer resources has led to a rise in "quiet hiring"—encouraging internal talent mobility and upskilling to grow your teams' capabilities without hiring externally. All of this *can* be good news for team members who want to develop lateral skills and explore adjacent disciplines.

But some reluctant backend developers are expected to fill in regularly on frontend tasks—in some cases, on top of operations work they've already taken on. "You build it, you run it" has evolved to include "and create and maintain the UI."

When in-house frontend specialists are in short supply, they're often assigned to the highest-priority customer-facing apps. This pressures full-stack and backend developers to pick up the other frontend projects—leaving their zone of excellence, where they do their best work, behind.

The cost of forcing full-stack

But why is it bad that your backend-focused devs are building UIs? What might seem like a one-off project or a praiseworthy "above and beyond" effort have impacts that reverberate within your engineering org.

Unhappy and unfocused teams

When we talk about context switching in engineering, we tend to focus on the cost of workday interruptions—meetings and Slack notifications that pull a developer out of their flow state. But when you ask a backend engineer to take on frontend tasks, they're also constantly moving between different languages and parts of the codebase.

A survey from Cornell University found that it takes us an average of 9.5 minutes to return to focus after switching between applications. The same study also found that:

- 45% of respondents felt less productive from context-switching
- 43% of respondents felt fatigued from switching between tasks

Here's one full-stack engineer on how working across the stack is riddled with context switching:

> Here's what I'm dealing with:
>
> - Switching branches in my IDE multiple times a day
> - Sometimes even switching to a completely different IDE.
> - Closing and reopening web browser tabs for different projects.
> - Recreating environments and setups for each task.
> - jumping between Slack channels to follow project-specific discussions.

If this experience is painful even for a full-stack engineer who chose this life, it's not surprising then that dedicated backend engineers might be reluctant to spend time away from their chosen specialization.

Context switching and its cognitive cost doesn't just cause the frontend work to suffer. If your team is spread too thin, everything—including the engineer's specialty—is compromised.

Framework fatigue

Frameworks make some tasks easier, but each comes with its learning curves and tradeoffs. Rapid development has put more pressure on frontend engineers to keep up with the latest "best" framework—which is near impossible for anyone not working full-time in the space.

For organizations that *have* standardized, React has become the go-to framework and for good reason—it's a powerful abstraction for building UIs. But it's not the right solution for every use case, and even frontend developers are objecting to the Reactification of everything. Even in a standardized organization, building a full-blown React app for every use case puts your backend devs on the fast-track to frustration.

Bugs and delays

While it might be easy to dismiss frontend as window dressing, it is arguably just as—if not more—critical to the success of a business as backend work. Assuming you meet baseline performance requirements, a lot of the work that happens on the backend comes down to implementation details that don't directly impact customers' experience of your services.

We know that estimating engineering work is notoriously difficult, and it's only harder when asking someone to commit to development timelines including work outside of their core capabilities. If your team is taking on frontend tasks, it's easy to overlook things like accounting for different browsers or window sizes. The resulting delays to feature rollouts due to frontend bugs or an issue in the UI that affects a lot of customers can hurt the business's reputation.

Apps that miss the mark

A lack of frontend experience (or, let's be honest, interest) can mean that no one on your team is asking the right questions about what they're tasked with building. This can lead to apps that *technically* meet acceptance criteria, but frustrate your users.

This isn't only true for customer-facing apps. Sure, internal dashboards and interfaces may not require the same polish as your core product. But no one wants your sales team to be self-serving data via a dashboard that's prone to hallucinations, or an awkward and unintuitive interface resulting in engineering hours wasted on an app that its intended users don't use.

How to keep your devs in their zone of excellence

With the right team structure and tools, you can create an environment where your engineers can do their best work without having to be experts in every layer of the stack, and create a flourishing team in the process.

Encourage T-shaped skills

Specialization doesn't mean diminishing any full-stack skills that might exist on your team. Generalists bring fresh perspective and creativity to engineering problems (just read the comments on this post). But just because someone is full-stack doesn't mean they should have to be at all times.

The most resourceful engineers have T-shaped skill sets: deep specialization in one area complemented by a solid working knowledge of adjacent parts of the ecosystem. The idea of being T-shaped is not about being ready to replace or sub in for other roles. A well-rounded skill set makes them better collaborators with teammates by understanding the tradeoffs and considerations that engineers in other disciplines will be thinking about.

Exposure to related disciplines helps them avoid getting blocked on small tasks outside of their wheelhouse. Structuring your team around their T-shaped skills will ensure awareness and alignment without over-extension of skills.

Equip your team with the right resources

Even without the headcount to add the relevant expertise to your team, you can support specialization without cutting corners. Identify tasks with undifferentiated complexity that you can outsource to developer-first solutions and free up your team to focus on work that only they can do.

Investing in an application development platform relieves your teams of the work they're less qualified, inspired, or equipped to do, freeing them up to focus on what they're good at. Offloading the heavy lifting to a platform saves time and effort on both the building and ongoing maintenance.

Removing complexity this way means you can say yes to more high-impact projects (that your team *wants* to do), because your developers can build anything they can imagine, without worrying about limitations—whether that's their time, expertise, preference, or stack.

Know how and when to use AI

Yes, your backend engineers are resourceful, and they now have access to AI-assisted tools that can support frontend work, if not automate it completely.

But AI tools are most effective when outputs are combined with human judgment, and that judgment comes from deep knowledge and experience. Say your team is using an AI coding assistant to create a custom dashboard quickly. If your backend or full-stack engineers aren't as familiar with frontend languages or concepts, they might not know whether the code they're getting from tools like Copilot, Cursor, or Replit is any good. AI tools also usually default to fixing errors and problems with *more* code, while a human engineer might simplify instead.

If your backend engineers are leaning on AI to make major frontend decisions on frameworks and libraries, you won't be setting yourselves up for success. Without frontend-informed prompting, an LLM could recommend an obscure library that meets your needs at the moment but lacks any long-term support.

Without a deep knowledge of frontend, relying on AI may result in an app that meets initial acceptance criteria, but sets you up for significant tech debt down the line with code that's difficult for others to modify and maintain, or with performance implications for your frontend *or* backend, depending on how requests are made to your API.

Your backend engineer may well complete the task in front of them. But without knowing what they don't know, building with AI without understanding the output can leave teams trapped in a cycle where AI is the only entity capable of maintaining the app.

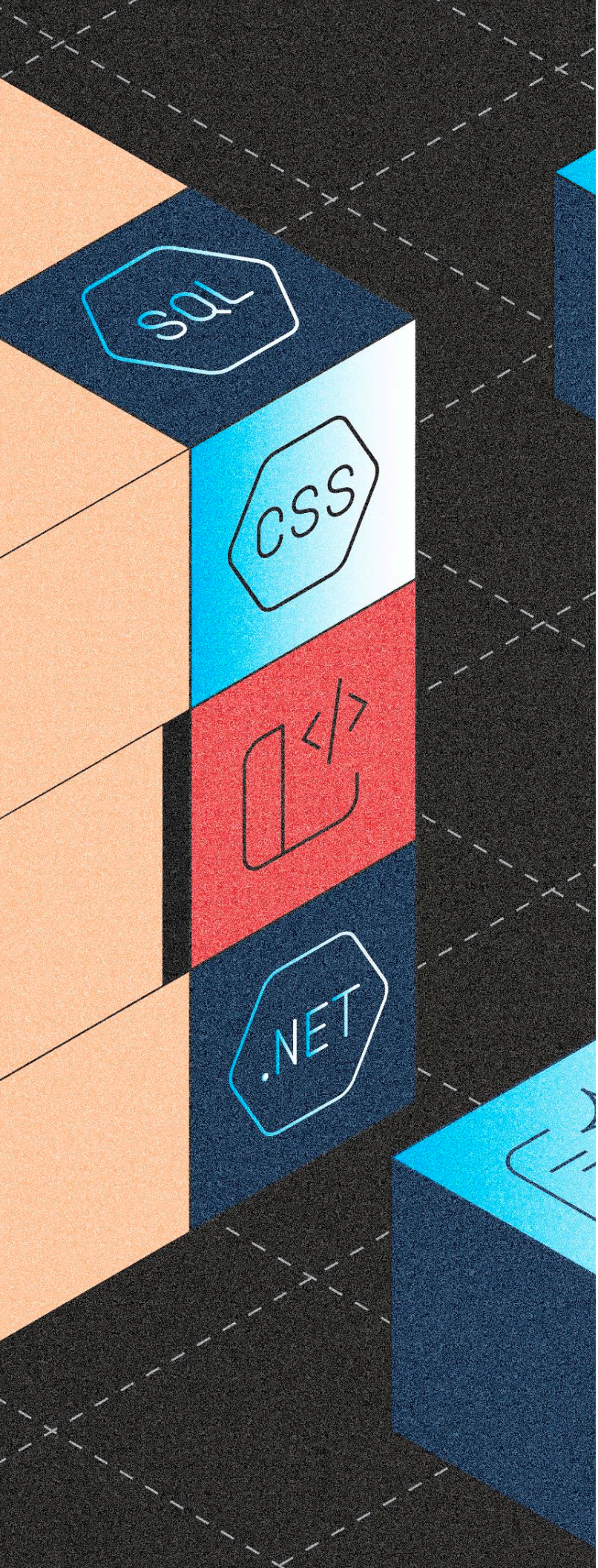
The future of frontend skillsets

Teams that stick with approaches and frameworks they know and love will produce better work in less time. Combining your specialists' T-shaped expertise with tooling that eliminates unnecessary complexity will make your engineering department unstoppable.

With the right solutions in place, engineers can concentrate their efforts on the work that they're best at and that they want to do—like building elegant, impactful solutions that people want to use. For engineers at Sage Home Loans, that meant offloading mundane and time-consuming day-to-day tasks like managing dependencies and handling permissions and authentication to Retool.

> "Convincing engineers to work on legacy admin tools is not the easiest thing. We've been able to obfuscate a lot of the functionality and say you're going to work on cutting-edge integration, not updating the button or grid layout on the dashboard. It's helped us attract and retain technical talent."
> — Chris Jaynes, a former Vice President at Sage.

Abstracting low-priority work isn't cutting corners—it's the best path forward for backend and full-stack developers who want a *better* way of working. It's what will allow them to prioritize meaningful, important projects (both internal and external), without the heavy cognitive load of moving around between different parts of the stack. It's how engineering teams will thrive.



The fastest way to build good software

Ready to unblock your team? See how Retool customers are saving engineering time and shipping internal tools without the heavy lifting. Book a demo to give it a whirl, or start for free today.

book a demo

start for free

Retool