Breaking the rules: I threw away 10 months of work after 2 months on the job.

When I took over the team, they were in month 8 of a 3-month project to relaunch the company's ecommerce website. After 2 months leading the team, I decided to scrap it and start over. This is the story of how and why, and whether it all worked out.

Dan Cowell
Jul 1, 2023 • 6 min read

The relaunch of our ecommerce website had one goal: deliver blazing fast performance with Server-Side Rendering.

Our JavaScript bundle size was 168mb on a good day. The development environment took 3 minutes to become responsive on a top-of-the-line MacBook Pro. Engineers would regularly come in to work, type `npm run dev` and go for a coffee while their machine's fans tried and failed to keep up with the demands placed on them by our bloated project.

Where did it go so wrong?

The backstory

The company's first website was an AngularJS single-page application. AngularJS didn't support server-side rendering, so the website was slow to render on first load, but snappy once everything had downloaded.

On the web, milliseconds matter - and this goes double for ecommerce. We needed to reduce our time to first meaningful paint.

Angular 2 was on the team's radar, and although it was a substantial departure from AngularJS, it seemed like the logical foundation for the new website. There was even an experimental server-side rendering feature branch under active development! Surely it would be ready to go by the time they were ready to ship!

The team built the site, regularly merging the latest changes from the experimental branch into the code base and dealing with the breaking changes that turned up.

I joined the team about two weeks before the site was feature complete. The deadline was already well in the rear-view mirror, but most of the pages were there, and the critical functions all worked. My job was to get the project delivered.

The Angular team inched closer to shipping server-side rendering. We put the last bit of polish on the critical user journeys. Everything was going if not well, at least... going. Until it wasn't.

We had a working website. The only feature missing was full SSR support. Server-Side Rendering *worked*, but not with tree-shaking, and the AOT compiler (a critical step in the Angular build pipeline,) was spitting out enormous bundle sizes.

Three weeks were put towards shipping the "nice-to-haves" from our backlog, including a really impressive nginx image cache (that's another story.) We checked the Angular repo daily, hoping that the critical fix would land.

We ran out of nice-to-haves and spent two more weeks hacking on the Angular team's SSR branch, but the sprawling, unfamiliar code base and impenetrable errors made it an uphill battle. The experimental nature of the project meant documentation was scarce.

We were burning time and getting nowhere, yet it felt like such a waste to give up now when we were so close to the finish line!

After a week of no meaningful progress, and little action from upstream, I was at my wit's end. I was under increasing pressure from management, our competitors weren't waiting for us to catch up, and the business needed this project to ship.

We had no clear path to delivering the product we had, but could we afford to start again?

> 💡 We were suffering from the **Sunk Cost Fallacy**: throwing more money, time and resources at a failing project because we were afraid of losing the resources we had already invested.
>
> This inevitably leads to a greater loss when left untreated.

I spent a sleepless night questioning the decisions that had led to this point. I knew from experience that React & Redux had mature SSR support.

I had seen first-hand how fast a motivated team could move with that stack, but throwing away almost a year of work would be a major blow to the morale of the team, and a tough sell to upper management, who had been waiting for this effort to bear fruit.

It also went against every instinct I had about big bang rewrites.

Starting over

The hardest part of this effort wasn't going to be technical, but personal. The first challenge was getting buy-in from the founders. I achieved this by promising two things:

1. I would maintain situational awareness of the Angular SSR project, and if it shipped, we would immediately release the Angular website, and
2. The React site would be functional and performant in 6 weeks, and ship in 10. If it didn't, I would step down and recruit a replacement.

The first promise was common sense, the second was confidence born from experience. I had built a server-side rendered site in React, and knew it worked. From the founders' perspective, they had a clear stop-loss in place if things didn't work out.

The more critical challenge was getting buy-in from the team. They had spent months grinding on the project before I joined, and now I was proposing to throw away that hard work. Worse, I was setting a crazy deadline! They needed to believe it was possible to hit, and moreover, the right choice.

I had to sell the team on the React stack, and I figured the only way to do it was DevEx. I had to show them how productive they could be without their tools holding them back.

I bootstrapped a server-side rendered React project and chose a single flow to implement, setting the ambitious goal of delivering our homepage - fully responsive, with SSR support - within a day of mob programming.

Everyone was blown away by hot module reloading and real-time re-renders in the browser. After coming from a project where they had to wait up to a minute to see their changes (on a good day,) this seemed like magic. There was still some healthy skepticism around whether this performance would still be there once the rest of the site was built, but this demonstration was enough to get the team invested in trying it for themselves.

> 💡 Everyone loves a **fast feedback loop.** Waiting even a short time to see the impact of a change can kill flow.
>
> Investing in accelerating your test suite or finding a more performant build system tightens this loop and directly increases the engagement and productivity of your team.

On day 2 we chopped up the site into 20-30 key user stories - many reused from the first attempt - prioritized them and started to deliver.

The new staging site was online and open to the entire company, and at the end of the first week, any skepticism that may have remained had been squashed after seeing how far the team had come in such a short time.

Delivery

We delivered the new version of the site after 7 weeks of focused effort from the team, in time for one of our biggest shopping events of the year. It was blazing fast, used far fewer resources than our original AngularJS website, and the code we wrote is still in production several years later, although the site has changed a lot since then.

Angular Universal with first-class SSR support would eventually land in v5 of Angular, shipped April of 2018, about 6 months after we finished the rewrite.

This project broke two cardinal rules of software project management:

Don't build production applications with experimental tech.

The Angular version of the site was built on experimental, unproven technology still under active development.

This was one enormous external risk that went entirely unmanaged - the unknown ability of the Angular team, who had no particular accountability to us, to deliver their product.

We would have been spinning our wheels and delivering no business value for 9 months all told - doubling the length of the project - if we had waited.

Don't do "big bang" rewrites.

Deciding to rewrite the entire website in Angular 2, without the forcing function of incremental releases, led the team into the trap of implementing the entire site before confronting their biggest risk.

The topic of how to actually release the site would have come up a lot earlier in the development cycle if the requirement to incrementally release had been in the project's measures of success.

"Wait," astute readers might say, "wasn't the React project a big bang rewrite?!"

Bonus: Learn the rules, so you know when to break the rules.

I made the conscious decision to break this rule in order to maintain team morale.

I was banking on the developer experience that a state-of-the-art React project offered to keep the team motivated. Shoehorning React into a legacy AngularJS project was not going to achieve that goal, and returning to the old code after months of building The Next Thing would deliver a crushing blow to the team.

This was a very situational decision. If we hadn't been coming off the failed Angular 2 project, I would have pushed for using one of the React <> AngularJS compatibility shims available to gradually transition the site from day one.

> 💡 As a manager you have to **play the hand you're dealt, not the one you wish you were holding**. Every situation is unique, and every decision you make must take into account not only the product, but the people who are building it.
>
> Sometimes that means you break established rules, but they're rules for a reason, so don't make a habit out if it. 😉

There are many more stories like this one that I have to share, some of them are even deep dives into the more successful things we built while working on the Angular 2 project. Check back for more of those over the coming months!

Thanks for reading.

"balls": The day I locked everyone out of the company intranet.

I was just over a month into my first job in the tech industry - a lowly HTML jockey and frontline support drone at a local web agency. Everything that could have gone wrong this particular morning…

Jul 15, 2023 2 min read 1 comment