SACRA

EXPERT INTERVIEW

UPDATED
02/15/2022

Cole Krumbholz, founder at Formspree, on the future of full-stack development

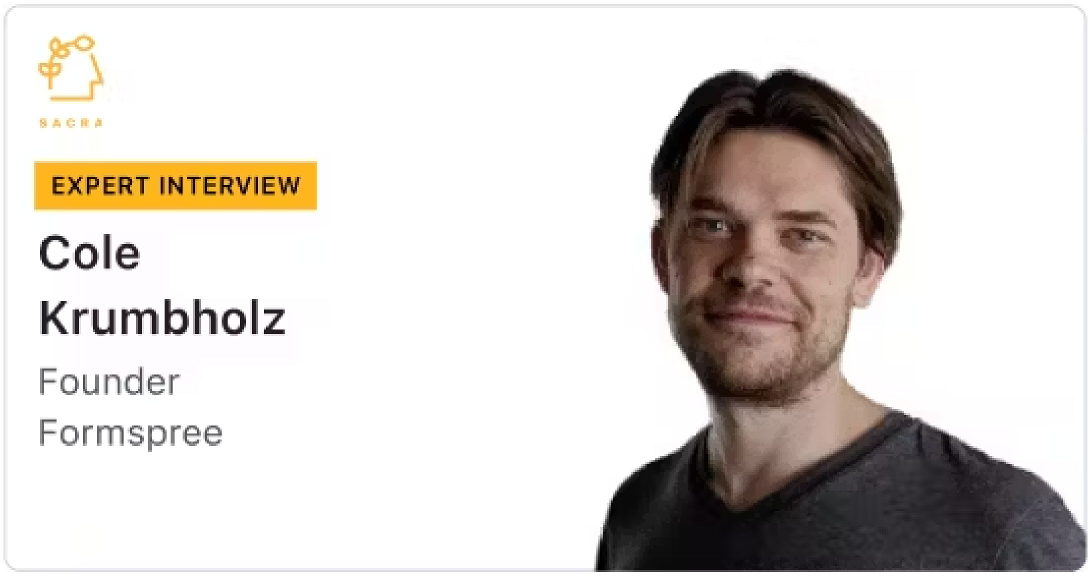
TEAM

Jan-Erik Asplund
Co-Founder
jan@sacra.com

Published on **Feb 15th, 2022**

Cole Krumbholz, founder at Formspree, on the future of full-stack development

By **Jan-Erik Asplund**



Background

Cole Krumbholz is the co-founder and head of product at Formspree. We talked to Cole because Formspree is an API product at the center of the growing trend towards headless and the Jamstack.

Interview

What has your experience with Jamstack been like as a developer, and how have you used Jamstack yourself?

There are two sides of it for me. I've used Jamstack to create websites. At Formspree, we've used Nextjs for our Vercel integration, and Hugo for our marketing site.

I've also been building products in this space for a long time. In 2012 I launched something called Backlift, which was a platform for single-page apps using Backbone.js. Then I transitioned to building Brace.io, which was a way to deploy

static sites using Dropbox instead of FTP when people still used FTP to upload code. Brace was acquired by Squarespace where I worked for a few years on their developer platform, and left in 2018.
I then focused on Formspree -- which we initially launched back when I was still working on Brace, in the early days of Jamstack (before it was called Jamstack). So I've been building services in this space for a while.

Why do you think Jamstack blew up so much around that time into now?

There are a lot of factors. I'm not sure that there was one obvious thing for me that made it blow up. But I do think that Netlify was a big part of it. They coined the term Jamstack, promoted it as a concept and proved it out. They spent a lot of effort to create an ecosystem that could compete with Wordpress. A big moment for the Jamstack ecosystem was Netlify scoring Smashing Magazine on their platform and showing that a big production content site could use this tooling. So I think a lot of it was them pushing the industry forward.

A few other enabling factors were static site generators becoming more popular, especially GitHub's Jekyll generator. The GitHub ecosystem in general becoming the default way to create and version code. Around that time Github’s API allowed Continuous Integration tools to hook into the workflow and run automations whenever code was pushed. So I think static site generators, GitHub and CI were important enabling factors too.

To give Netlify even more credit, I think that the build hook was a small but essential innovation that unlocked Jamstack as a workflow that could compete with Wordpress. Before the build hook, you needed a developer to compile and deploy website changes manually. If you had a non-technical content author that was creating markdown files, getting that person to commit to a Git repository was a big hurdle.

Having build hooks allowed a third-party API to trigger the rebuild. That opened the door for headless CMSs to offer a non-technical interface for writers to produce content. The CMS could automatically rebuild the website via build hooks whenever the content changed. That sorta made the whole thing work.

Could you talk about Netlify in comparison with Vercel and touch on why Next.js is important to understanding this?

When comparing Vercel to Netlify, my understanding is that they come from different places. When you look at Netlify, Mathias -- the co-founder -- came from the agency world. I think he had been working on a mix of content heavy projects including some WordPress projects. He launched something called BitBalloon, which was a way to deploy static websites, and a precursor to Netlify. He was focused on what I would call classic website development.

I think the Vercel product and team came more from the JavaScript community, and were trying to iterate on the developer experience for full stack JavaScript engineers. Historically you have two very different problem sets that web developers were tackling -- the server side and the client side -- and not really an ability to share code between them. In the Node.js ecosystem you end up with two different systems doing similar things. They're both essentially rendering HTML, whether it's rendering it on the server side or the client side via something like React. Trying to come up with ways to reuse that code across the two different environments and simplify the experience of writing JavaScript applications in general -- I think that's where Vercel had been innovating for a long time.

I think Next.js took the things people liked about static sites and also addressed a lot of problems that JavaScript engineers were facing. For example, rendering on the server and the client in the same way, incrementally adding functionality on the client so that you get fast load times. Those kinds of problems are the core of what Vercel started off trying to solve.

Are there certain kinds of teams where this Jamstack approach really makes sense? Does it not make sense for everyone?

I feel like Next.js is, in a way, the future of full stack development. When I started working on Next.js websites, I thought, "Oh, I get it. This is basically just like a full stack application." It can do everything I expect from a traditional MVC web framework. And it's packaging that along with the

tooling and build process that you need to create an interactive front-end application. That to me is an easier workflow than separately managing the frontend and backend code, which is what we had to do in the past. It blurs the line, and unifies the dev experience. So I see Next.js as the evolution of full stack development.
I think Netlify is trying to solve a different problem. They're evolving the platform. Especially for content-heavy sites -- the kinds of stuff that you might have used WordPress for -- I think that's where the Netlify platform and tooling is helpful.

It's a little hard to compare them because Netlify doesn't have a horse in the race of web frameworks. They've focused on improving the infrastructure and deployment experience regardless of your framework, even if you’re using Next.js. Vercel also has a platform, but I don't know how many people are deploying non-JavaScript or even non-Next.js applications to it. I could be wrong, but I feel like if I were to try and build something using a different static site generator than Next.js, I probably wouldn't try to deploy it on Vercel. But for Next.js it’s the way to go.

One idea that we’ve been talking about is that where the value is in the Jamstack ecosystem is less like Vercel and Netlify, and more all the APIs -- that's what you're really betting on, a future where there are just so many more APIs that are getting better constantly. Do you have any thoughts on that as a theory?

Well, I feel like APIs have been a part of the web development ecosystem forever. If you want to send text messages or collect payments from your website you use APIs. However, there are a specific set of APIs that grew up around Jamstack and are there to solve Jamstack problems. For example, headless CMSs are new. It's not something that people needed until they stopped packaging a database with their web server. It’s just a different way of managing content.

Formspree is somewhere in between -- we're an API that serves the Jamstack community. But we’re also serving a need that’s as old as the web. It used to be that when you built a website, you’d have a CGI-bin -- a way to run server-side scripts for things like collecting form submissions -- and every web host would give you access to that. But today’s hosting providers often don't support CGI-bin. Plus spam is a bigger

problem, sites must handle more traffic, marketing teams require more functionality and end users expect it all to just work. So when developers reach for an easy way to collect form submissions, it makes sense to use a service like Formspree instead of building it themselves.
I think we're part of a trend of developers moving away from the CGI-bin approach of building their own server-side functionality, and letting 3rd party APIs handle things. Jamstack puts those APIs at the center, but I think it goes beyond Jamstack. Anybody building any web application today is taking advantage of APIs.

One of the powerful things about Next.js is that it makes serverless functions first class citizens. That gives you more flexibility for how to incorporate those APIs. Serverless functions in general are useful for glue code that connects to an API on the backend, sends the appropriate credentials, and cleans up data before sending it to the frontend. I don't know for sure, but I would imagine that's the most common use case for serverless these days -- tying together all those APIs.

Could you talk about trade-offs to working in a Jamstack development mode? What are some of the key friction points that still need to be ironed out? One that we were thinking about is collaboration between engineering and content teams.

I think the whole idea of headless CMSs and build hooks allowing non-developers to collaborate was one of the biggest problems. I guess there may still be friction there in terms of immediacy. When building a site in Jamstack, you ship some content, but it still takes time to build and get distributed via CDN, the content delivery network that's an essential part of the Jamstack approach. As opposed to just making a change, saving it in a database, and next time you make a request it comes out of the database. I think there's perhaps a faster iteration speed there, and I'm not sure that Jamstack has 100% solved that. So that part of the collaboration may still be a challenge.

And then there's just people learning to do things a different way. That's always a friction point. At the end of the day, for somebody that's trying to create content, as long as the words are on the page, it looks good, and it's achieving SEO goals, it probably doesn't matter that much which approach is taken. So learning a different way of doing things, or even just learning a different CMS, can be a barrier.

Does anything come to mind that might serve as a useful analogy for what Jamstack represents? We were thinking of Agile, which was a new way of working where a bunch of tools emerged to support people working in an Agile way.

I think in some ways it's part of this larger trend pushing more functionality forward to the client. It is about technology: improving load times, improving security and making the end user's experience faster, better and more interactive. If it weren't for the constant pressure to innovate in terms of the user experience on the web, there wouldn't be so much froth in the JavaScript community and so many people trying to create new tools and changing practices every couple of months. It's just part of the constant evolution of the web, which is a very innovative and dynamic technology.

Disclaimers