SACRA

EXPERT INTERVIEW

UPDATED
02/23/2022

Jamund Ferguson, senior engineer at PayPal, on using Jamstack in the enterprise

TEAM

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

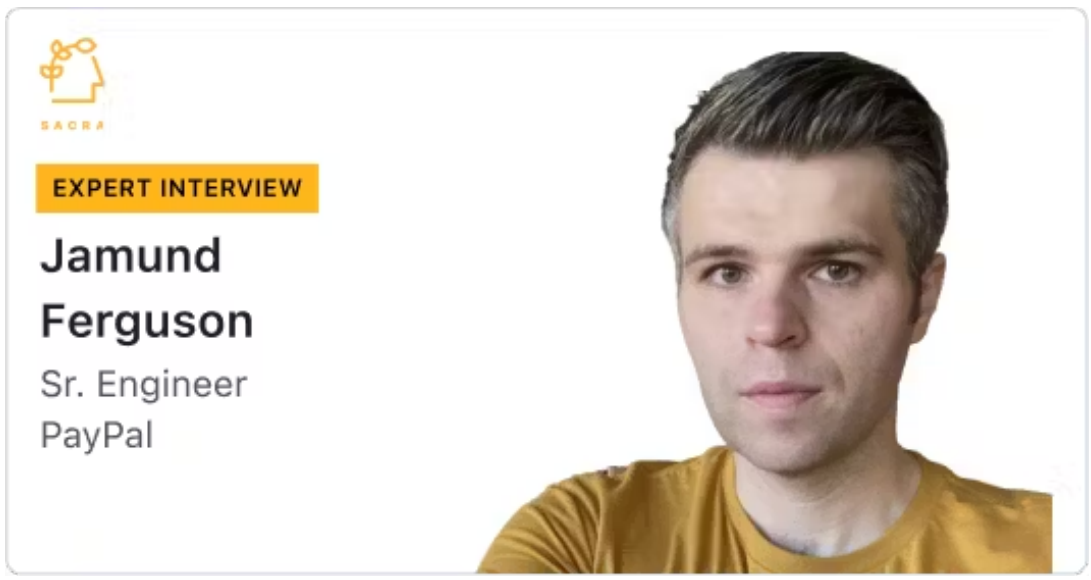
DISCLAIMERS

SACRA

Published on **Feb 23rd, 2022**

Jamund Ferguson, senior engineer at PayPal, on using Jamstack in the enterprise

By **Jan-Erik Asplund**



Background

Jamund Ferguson is a Senior Staff Developer Relations Engineer at PayPal. We chatted with Jamund because of his background of working with deployments of Jamstack in the enterprise, as well as front-end development more generally.

Interview

Could you start by giving us a definition of Jamstack?

Jamstack is an acronym that was started by Netlify for “JavaScript,” “APIs,” and “markup” -- basically, HTML websites that use JavaScript to add interactivity and call APIs to get that data. Another term for it that I prefer is “static web apps.” You have these apps that are interactive websites that are formed with static markup, as opposed to being generated on the server. Most of the websites out there are dynamically generated on the server. When you make a request, these static web apps or Jamstack apps are built in advance, so all

the HTML is generated in advance, and they're usually delivered with a content delivery network. The idea is that they're going to be very, very fast because all the content is sitting as close to the consumer as possible, as opposed to in some data center -- maybe in AWS or wherever -- but not necessarily as close to your customer as it could be. Jamstack I think was somewhat of a marketing term, but it really caught fire. A lot of engineers, especially front-end engineers and React developers, were like, “Yes. Give me the power I need as a front-end developer to be able to deliver websites and not have to worry about servers and all that complicated business.” That's roughly my take on it, anyway.

Why now? What are the underlying developments and trends that have emerged that made this happen in the last couple of years?

I think one of the biggest things is the growth of these content delivery networks. Obviously, old school CDNs like Akamai serve media assets and have done that for a long time. But there's been a lot more interest -- companies like Cloudflare and Netlify and others that are pushing out more and more data to the edge, closer to the users. This is an ongoing trend. You also have this serverless trend of being able to deploy advanced websites and APIs and technology like that without necessarily having to be worrying about maintaining the servers.

And then a couple other forces are happening. One, front-end website development is getting more advanced and more complex. You have tools like React that are being embraced wildly, and people are creating really complex and interesting websites with them. So you have these front-end web developers that are really, really good at learning all the ins and outs of the frontend and maybe have very little backend experience and very little experience with PHP or Java servers. They built their front-end app all using React, and they just want to get that out there to the customers.

And then one other thing that has come along is a push for web performance and a demand that our websites get faster, especially as there are more mobile users and more users as the web continues to grow throughout the world. You want these websites to be accessible and fast to everybody out there. We've seen that the traditional model of building

websites maybe isn't quite as fast or as optimized as it could be. So there's been a lot of push, at least in the web development community, about how can we get things to users as fast as possible? And then, how can I build out these sites as easily as possible?

How did you get into Jamstack, and what kinds of projects do you use it for?

I gave a talk about this at QCon a couple years ago. Essentially my interest in Jamstack started with a frustration with our Node.js application architecture. At PayPal, we were pretty early adopters of Node.js at the enterprise level. I mean, we have thousands and thousands of servers and at least hundreds of applications out there powering all sorts of different user experiences based on Node.js and the Kraken framework that we use, and it has worked fairly well.

Previously when I was working at PayPal -- I've rejoined since then -- I was working on our Send Money page. What I found was that deploying that application to hundreds of servers through multiple data centers ended up taking a really, really long time. And many, many times the stuff that I was deploying was essentially just updates to the website, so just JavaScript assets or UI assets, HTML, CSS. That required redeploying these whole servers across many different places, which was a multi-hour process. We had a slow deploy process. Not necessarily unique to our situation -- it was fairly standard at the time, but unfortunately it was not very fun, so I was frustrated with that.

The other thing that frustrated me with our setup at the time was that we had a lot of server crashes. Oftentimes we would run into problems, and there'd be all sorts of different things that could cause these. PayPal is running in many different languages, but there'd be certain times that someone would request something in a language that we didn't support. In some cases that would crash the server. We have hundreds of these servers running, so one crashing isn't causing any significant downtime, but it would bring down that request, as well as any other requests being handled on that particular server instance. So we were running into problems maintaining Node.js at that huge scale.

There were other people trying to figure out how to make Node.js and our implementation more stable, and we've

definitely made progress there. And some teams were working on how to make our deploys faster, and they've since made progress there as well. I was looking at it really from the perspective of a front-end engineer saying, "I mostly just want to update front-end assets. I don't even care about the backend or the APIs. Those can be separate. Can we decouple our APIs from our markup -- the sort of Jamstack idea -- and maybe quickly deploy our markup and static assets and then deal with the APIs separately?"
For me, the focus was partially on the developer experience. As a front-end engineer, I don't want to have to have these slow deploy times for just updating some UI assets. And then also the stability factor. With Jamstack, you pre-compile your stuff and put it up on the internet as just static HTML for the most part, so it's very stable. It doesn't go down, really. It's just a file that someone downloads. So that's what got me into it. And then, as I shared in that talk, we did a lot of exploring of it at PayPal. We looked at our paypal.me site, which is a super popular way for people to send money, and we did some experiments there, and it was very interesting.

There's a lot of details that I don't want to get into necessarily here, but I will say that, overall, I think that the company ultimately found other ways to address the problems that drew me to Jamstack. I didn't end up staying for that full process of testing that out. But it was really cool. I was really happy with our experiments. We ended up using the Gatsby framework to test this out, and what we found at PayPal at least was we were able to deploy things really, really quickly like I wanted, and the instance of paypal.me, for example, that was running on this platform was super fast. So the websites were faster, it was deployed faster, and the developer experience and the user experience benefits were there.

For a whole bunch of reasons, it didn't end up getting full adoption within the company. I think enterprise adoption of Jamstack is actually tough, and maybe we can talk about that. But overall it was a great experiment. I'm really happy that we were able to go down that path. I ended up, like I said, leaving PayPal around that time. I came back later and was a little sad that it never took off as the primary way we were building websites internally. We do still have some Jamstack sites. Our developer documentation is built on Gatsby using a similar technique. But it didn't take off as a primary platform for us. We're still mostly using Node.js for our application layer.

SACRA

We've heard a couple times now that people love being able to start out projects and have them be performant really quickly within Jamstack or using Vercel or Netlify. This might tie into your point about enterprise adoption, but can you build fully featured apps as easily? And if not, what prevents that from happening?

I think that there are some interesting trade-offs when it comes to Jamstack. The static web app approach means that, if you're having a fully dynamic thing, that's going to be harder. Let me take you to another example. I left PayPal and went to a startup for a while. I was still really excited about Jamstack there. At the startup, we were using headless WordPress, which is still kind of Jamstack-ish stuff. I didn't last there very long and ended up at Amazon in the seller central org. There I was working on building sites to help people sell their products on Amazon.

It was the most interesting thing to me. Basically, the Jamstack technique that I had proposed at PayPal -- we already were doing the same thing at Amazon. It wasn't widely celebrated as Jamstack, but it was essentially Jamstack. We were able to deploy static apps and deploy APIs separately. There was a layer there that essentially said, “Look, you could be a static app. You could be a server app. It all runs through this single interface that basically sets up your web server for you.” So the technique that I had been trying to pioneer within PayPal, we ended up using at Amazon.

What I found that was really, really frustrating is: it turns out at Amazon and a lot of places, we do a lot of dynamic stuff, so every user experience is very custom to a specific person. What that means is, instead of having a static site where you just immediately get the data because it's at a local CDN to you, we deliver to every single person a static outline of a page, and then we hit an API trying to get content. So you just got a lot of loading spinners. I ended up focusing a lot of my time at Amazon working in web performance, and it was basically how to get rid of these loading spinners and make stuff faster.

In many cases that meant trying to nudge people away from that Jamstack approach and toward a more traditional server-rendered approach, hopefully without losing too much of the developer experience wins because front-end developers

absolutely love the Jamstack. It's so much easier for them than having to think about a server the whole time. By the way, that's where I think Next.js has done a great job of saying, “We'll basically give you the developer experience of Jamstack and you only have to think about building your front-end web app, but also server capabilities that are pretty easy and seamless to use.”

What is the path forward that you see for enterprise Jamstack? What would have to change in order for Jamstack to start eating up enterprise use cases?

Let me give you a very niche issue that we actually ran into. I don't know if Amazon ran into it, but we definitely ran into it at PayPal. And I think it is kind of related, which is that both companies -- PayPal, Amazon -- have our own infrastructure. In most cases, we're not just going to be like, “Oh yeah, I'm just going to send this over to Vercel or Netlify.” We've got really good internal services -- obviously Amazon has all of AWS at its disposal. That being said, a lot of the specific features you need to do Jamstack well require very specific CDN capabilities, so your CDN has to be able to handle authentication and do all this other stuff at the edge.

If you look at some of the more modern stuff -- like Cloudflare's putting out additional edge computing and even storage without sacrificing the simple development experience -- I think that's where we struggled a little bit at PayPal. We had a traditional CDN infrastructure, and it worked really well for just plain static assets. But when you're putting up a website that needs authentication, and it needs certain security -- you know, CSRF tokens, all this kind of security stuff that you'd put in a server-generated website -- you think, “How do I do that if I'm on a traditional CDN that doesn't have the special capabilities that maybe Netlify or some of these other providers have available there?”

So I think those companies that have focused on Jamstack are actually a little bit ahead of the game. I'm sure other places are starting to play catch up and realizing that these are really important features. With a number of these hosting providers, there's a simple configuration that you can make that's just very easy to do and that's way harder in a traditional Amazon S3 bucket or something like that. There's a number of little details that I think Netlify got right, and I'd argue Vercel and others have figured out, to make it that much easier to do

Jamstack well, whereas if you're in a big company and you're just trying to use traditional infrastructure, you don't have access to the cutting-edge stuff, and it doesn't really work very well. Both at Amazon and PayPal, we had a Jamstack-like development experience, but the actual processing of that stuff wasn't happening in the ideal way -- on the edge near the customers -- but instead in our data centers in a more traditional server format. We gave our developers a good experience, but it ended up not necessarily being the optimal experience for the users. That's where I think those cutting-edge hosting providers are still continuing to be ahead. If you look at what they're trying to do with Amplify at AWS, they're trying to be like all these other companies that have kind of innovated before them. Firebase for example had a lot of these features long before they were acquired by Google.
The way that I think about this is: these companies are oftentimes based on top of AWS and existing cloud providers, but what they've done is figure out a way better developer experience, and they're essentially making that developer experience an add-on that's worth paying a lot of money for. I don't think there's anything super proprietary in the technology from any of these providers. I do want to mention Cloudflare again, because I think they've been really innovative about how they're doing edge workers and things, which are really interesting. I think there's some cool technology innovation. But for the most part, I think they're just ahead of the curve in thinking about what makes it easy for developers to be successful quickly.

If you look at offerings like CloudFront from AWS, which are more basic, or just generally putting data in an S3 bucket and serving that, my guess is that all of those providers will eventually have a very similar feature set as Netlify and Cloudflare and other things. But it's taking a little while for the big providers to get there. I think it'll be interesting once all the big providers have the same features. Then it'll be easier for a large enterprise company to say, “Well, look. We already pay for this feature. We can just opt in.”

I know I went around the bush a lot on that, but I think right now it's a lot easier for startups to say, “Let's go with Netlify, let's go for Vercel,” and just let that scale them up to a big size. But if you're already a big company with a lot of built-in infrastructure, I think you'll find that the existing tool you have

might miss some of the key parts to make Jamstack successful for you.

We've seen some parallels in how Vercel and Netlify build on top of AWS to how Heroku set out to do something similar twelve years ago but for a server world versus serverless. Some consider Heroku to have failed. I'm curious if you see those as similar and if there are mistakes that Heroku made that Netlify and Vercel are managing to avoid or to improve on?

I'll be honest. I'm not a great business analyst, that's not my take. So I'm not going to comment too much on that. I will say, from a technical perspective, I think you hit the nail on the head -- just focusing on different audiences, right? Heroku did a fantastic job nailing developer experience for those Ruby apps and Node.js and these other apps. They just did a great job of making it easy to deploy those things. And they're still somewhat the gold standard for just a simple deploy process. People still respect what they're doing and what they've done. As an engineer that's used Heroku a few times, I don't think that they're completely flawless or whatever, but they're pretty good and much easier than trying to go on AWS and build out my own thing. Any day I'll choose Heroku.

I think that they did well for the audience they were serving. You could see that they started by focusing on startups -- I think they came out of Y Combinator and all that stuff. So they were living in that world the same way that Vercel and Netlify I think are living in that world, focusing on startups, and you get that same growth where, if you start with a startup and the startup grows really big and you can continue to support that growth, I think you're going to be able to grow tremendously. I don't have any great business feedback there. To me, I think they're basically just a modern version. I don't see any major differences at a high level. I think they're vaguely targeting the same areas. They're looking at basically selling great developer experience to startups and then helping those startups grow and grow with them.

My only thing with Heroku -- and to be honest, same a little bit with Vercel -- is maybe you start with them as you start up, and then as your startup grows, it seems like everyone ends up on AWS or on the cloud, whatever. I couldn't see many big companies being like, "Well, let's ditch AWS and just try out this newcomer," whatever that newcomer is. So I do think that

there may be an issue at a certain scale. How do you grow your startup-focused business to support enterprises? That is a challenge. Maybe Cloudflare has innovated enough and has enough unique features that they may be doing well at that, because I think they have some amazing stuff that they're doing with being able to prevent denial of service attacks and things like that. But honestly, I do think that that is a challenge. Growing from small startup to medium business, sure. But when you get to be a pretty large company, are you still going to stay on a smaller platform versus building your own on top of solid cloud infrastructure? I don't know.

People sometimes argue that Jamstack, perhaps partially because of the marketing origins of the term, is a transient trend. What I'm getting from this conversation, though, is that underlying Jamstack is this architecture and this way of building apps that is being adopted by AWS and Cloudflare.

Azure has this big static web apps push right now. They've got a whole DevRel team focused on static web apps. They have events. Static web apps aren't going anywhere. The basic technology there isn't going anywhere. I think where things are interesting right now is there's so much innovation around edge computing. Cloudflare is doing that with edge workers. Vercel is doing that by making it easier to use edge workers and that technology. And then Netlify is making it easier to use serverless functions. So there's a lot of interesting technology around edge computing and serverless broadly. I think Jamstack started as this basic static web apps thing, and then even Netlify started allowing serverless functions on their platform, and Vercel is doing their edge middleware. So you've got a lot of innovation happening in the space still. I don't see it as done.

I think the core architecture of “I have a static website that I want to put up and make it really, really fast” is a solid idea. If you look at where people are using this technology, mostly it's blogs, because it's static content. It works really well for blogs and marketing kinds of pages, like a landing page. I want that page to be as fast as possible. And then ecommerce as well, where performance is really, really important. When you don't have so much dynamic stuff in your ecommerce and it's not super tailored to individual users, you can take advantage of this to give your users really fast content. I think that's just a

few of many places that could continue to benefit from Jamstack moving forward.
There's a little bit of interesting pushback right now, I'd say, within the JavaScript community, because Gatsby was really a forerunner. Everyone was adopting that, and they were the main framework for building Jamstack apps. Then Next.js sort of supported most of those same features plus additional server rendering, and that overtook Gatsby in terms of popularity in the community. Now there are several other contenders. There's Astro, which is trying to be super fast and fully static, but with support for progressively enhancing the page with additional functionality. There are also static site generators like 11ty that are quite good. Then you've got Remix, which is being built by some really popular React community leaders, and includes full server rendering capabilities. We're seeing a little bit of pushback from static site generation with heavy client-side applications back into server-side rendering. But I think there's always going to be a place for those static, Jamstack style apps.

To try and organize my thoughts a little bit, you have continuing innovation in the edge space. Edge computing is something that's still a little bit mystical to people, but it provides power and server process and capabilities close to where your users are, just like how CDNs put your content closer to users. There's a lot of innovation happening there that's really cool. You've got new frameworks coming out that are challenging the notion of, “Well, maybe we want a little bit of server capability alongside your static apps.” There's constant innovation in the web space, but I think that static apps absolutely have a place in the future. But it will be interesting to see how these other forces take shape.

Looking at Blitz.js and RedwoodJS, it seems like what's emerging is a hybrid Jamstack or post-Jamstack. Is that legible to you as a category?

I agree with you. I haven't used those specific tools, but you have that. You also have the emergence of people that are saying, “You know what? All these React-based frameworks, we're sick of them. We want things to be even faster.” A lot of the folks that push web components and really hardcore web performance are saying, “Can we get these static site generators to not generate any JavaScript?” Because the problem is, even with these Jamstack sites, a lot of times

they're still very JavaScript-heavy, even if they're just displaying content. It's all client-side rendered oftentimes, and it can take longer than it really needs to just to display some content on a page. So you do have this server-rendered set of tools that you mentioned.
But there's also another set of tools that are static site generators offering interactivity without paying as much of a cost in terms of your web performance. Astro is one that I'm very interested in right now. It'll fully static site generate your page, but it still allows you to add interactivity without paying as much of a performance penalty as maybe a Gatsby-type tool would do. I think there's a little bit of renewed interest in, “How can we do this Jamstack stuff without paying as much client-side app penalty?”

Like I said, web performance has been a really interesting driver of a lot of these trends, because Google is pushing their web vitals as part of some of their SEO. You see a lot more people -- especially in the ecommerce world and other worlds that are really SEO-driven -- doing everything they can to make their websites a little bit faster. So that's another trend that I think is driving tools in a few different directions.

You've touched on ecommerce a few times. Do you see this as potentially encouraging people to build Jamstack ecommerce sites, maybe using Shopify as a back-end API or some other API to host their products, and as a headwind for Shopify itself? Does it seem like people will start building their own stores using Jamstack versus using Shopify?

I follow someone from the Wix team, @DanShappir, for example. I know Wix isn't Shopify, but it’s a similar space. That team is super focused. All those teams know that they need to be really, really fast, so they're doing everything they can to continue to optimize their stuff while allowing users to upload horrendously oversized images or make whatever mistakes that someone maybe less familiar with the space would do. Then those tools try to do a pretty good job of optimizing them.

I think there is a continued interest in using Shopify or WordPress or any kind of CMS as kind of a backend. And you've got Contentful, of course, and other tools to allow you to fully programmatically build your site and then drive content from these easy to use CMSs. To me that seems like an

obvious continued area for growth. Essentially, you always are going to have this disconnect between the people building your website and the people generating the content or updating the website. So thinking of ways to continue to allow you to build amazing, fast, really cool looking apps and sites, but also making it easy for people to update the content -- I think that's clearly a trend that's going to continue to grow.
I don't have specific opinions about the ecommerce space. All I can say is that I've seen a lot of posts recently from developers sharing, “Here's how to build an ecommerce site using whatever new technologies.” There's a lot of interest in that right now. So I do think that what you're implying -- maybe having some sort of backend to manage products and then building custom UI using Jamstack technologies -- sounds like kind of a winner, but we'll have to see. I don't have a ton of specific experience in that area, even though I work at PayPal and feel like I should.

JavaScript had a reputation for being rather hard to work with. I'd love to hear your take on how it has changed. Also, what are some of the key things that make Next.js different from just vanilla React?

If you look at the web development space over the last 15 years, in general you have one trend, which is absolutely everyone is being more online. Every company is a tech company these days. Everyone has to have a web interface. There's just this constant growth of web development. At the same time, you've seen React as a paradigm for building websites just completely dominate the mind share of developers. Many, many, many new websites are built in React. I get that WordPress still is the complete leader of the way that most of the sites out there are. That's fair. But in terms of modern web development, you see React absolutely dominating the way that many folks are building sites. React is continuing to grow and has grown tremendously.

What I think was always the missing story was: how does that React frontend connect with the backend of your website? We had a lot of experience with this at PayPal, and I'm sure many people have had experiences, where you have your Node.js backend and then your React frontend, and they sort of tie together. Your backend might be rendered using a templating language and your frontend with React. It's like, “Oh, could I

use those same templates to render on the frontend and the backend?”
There are all sorts of stabs at these isomorphic frameworks, as they were called, where you could share code between your frontend and your backend, and the same developer could be building your frontend as well as your server. And I think Next.js just really nailed the developer experience on that. They catered the whole thing towards front-end developers, of which there are just so many of these React developers, and they made building a full stack website feel exactly like just building the frontend that people were already familiar with. I think that's how they've gained a lot of popularity -- just making that experience really easy.

I think that what you see here with Vercel, Netlify and all these things is that they're not necessarily doing anything that wasn't already possible before. They're just nailing that development experience. Developers are thinking, “When I'm doing something hard, if there's an easy way to do it, I'm just going to flock to the easier way to do it.” Next.js nailed that developer experience for building full stack websites with server components, and it's obviously taken off.

Then they've been able to effectively leverage that. People in the Node.js community have known Guillermo and some of the folks that work at Vercel forever, and they have a ton of community respect, so people are already listening whenever they're announcing things. I've been following them since the early days of Zeit and before, when they worked on Cloudup (which was acquired by Automattic) and Learnboost. When people in the community are doing interesting things, we're listening up, and we're definitely like, “Okay, this is probably something I should check out.”

Disclaimers