SACRA

EXPERT INTERVIEW

UPDATED
03/11/2022

Thom Krupa, co-founder of Bejamas, on building dynamic apps on the Jamstack

TEAM

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

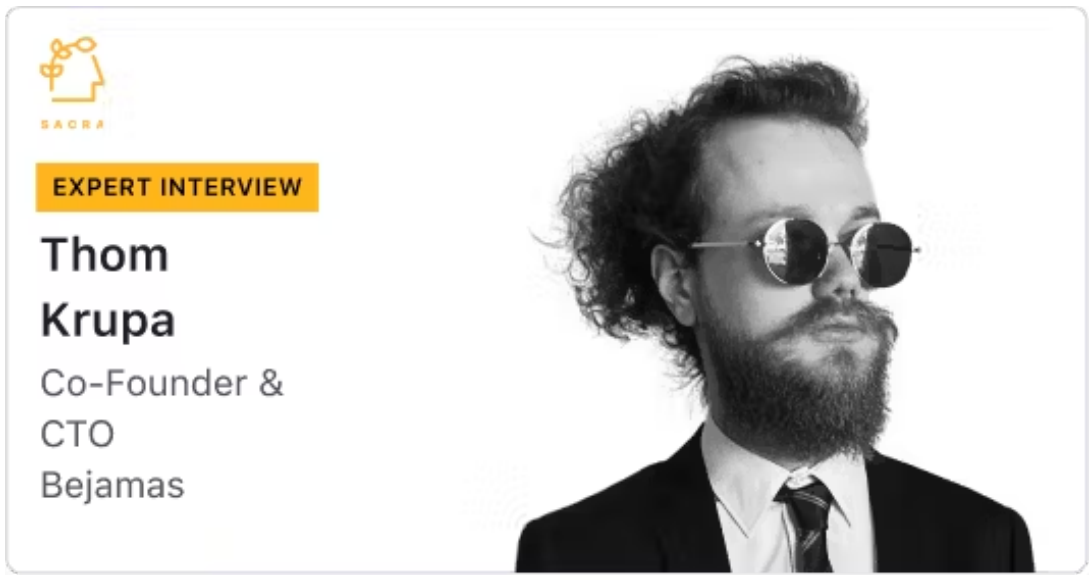
DISCLAIMERS

SACRA

Published on **Mar 11th, 2022**

Thom Krupa, co-founder of Bejamas, on building dynamic apps on the Jamstack

By **Jan-Erik Asplund**



Background

Thom Krupa is the co-founder and CTO of Bejamas. We talked to Thom because Bejamas is one of the biggest Jamstack agencies in the world, and we wanted to learn more about the practicality, use cases, and limitations of building with the Jamstack for businesses.

Interview

I'd love to hear about how you got started with Jamstack and why you chose it to be the foundation for how the agency builds stuff.

We started Bejamas three years ago, and in the beginning, we didn't know what Jamstack was. We -- Denis and I -- were just very excited about static sites, Hugo and Gatsby. We didn't know that the name for the whole thing was Jamstack. Once we started digging into it and learning more about the tech and other companies, we found Netlify and decided to position our

agency as a Jamstack agency. Basically, all we do is somehow related to Jamstack. That's our core technology and message. It wasn't just a marketing term. We found a lot of really exciting solutions to the problems we had. Before Bejamas, I was a front-end and WordPress developer, so I knew all the pain points of WordPress, and to be blunt -- I didn't like it. So, Jamstack was a cool alternative. That was just the beginning.

What were some of those problems that you saw as a front-end developer with WordPress? What were some of the biggest pain points?

I think the biggest pain points were performance and maintenance. It was hard to finish the project and just forget about it. You had to keep updating it, add different security patches and stuff like that. Then, if the website and business grew, you got more traffic, so you would need to upgrade your servers to manage that. Websites are full of different plugins, not every plugin gets updated, and not every plugin has a new version. Some plugins just don't work with newer versions of WordPress.

The big ecosystem of plugins and themes is a great idea, but at the same time -- it's a bit painful. You have good themes and bad themes, good plugins and bad plugins -- and that was not an ideal situation in some projects.

How does working in a Jamstack methodology help? Let's say I don't know anything about Jamstack, and you're saying, “Let's build this on Jamstack.” What's your pitch for that?

I think that the most interesting thing about Jamstack is its decoupled architecture. You don't have a monolithic WordPress system. You don't have one server -- actually, it's split backend and frontend. You have headless CMS -- for example, Contentful, Storyblok, Dato CMS -- or even headless WordPress is fine. You don't have to worry about scaling the backend, because it's mostly an API endpoint. It's separated from the frontend. If you want to build a website, it is run on Netlify and with static files, you're no longer connected. Users don't connect to the backend, they connect to the CDN, so the CDN is very close to the user wherever you are. It's just easier to scale, and it's cheaper, because CDNs are cheaper than servers.

Also, it's more secure. Because you don't have any direct connections from the user to the database, there's no security risk -- it’s all static, pre-rendered on CDN. This solves a lot of issues. You don't have to worry about backups, because every build is a separate environment. If you want to reverse some changes, you can publish some previous build just by one click on Netlify.

What kinds of websites do you build on Jamstack, and are there kinds of projects that Jamstack isn't that great for? We've heard some people say really fully featured apps are maybe harder, but others say it's fine to build them on Vercel or Netlify.

We mostly build websites that are static by nature. For example, landing pages, large corporate websites or large blogs -- the kind of websites where content doesn't change too often. We work on some marketing websites as well. We also build apps. Jamstack is for apps as well. There are different approaches to how you can build those apps these days. For example, you can just build a static skeleton, and the user can fetch data on the client's side and connect directly to APIs.

But that can be slow in some cases, since not every user has a good internet connection. So it's much better to render this on the server, at least the first view. If you enter some URL and hit the “go” button, then you'll get a fast response from the server. That can be done through the serverless functions, which both Netlify and Vercel have. Vercel recently released a beta of Edge Functions, which are even closer to the users. You can build dynamic apps with a Jamstack approach, but without the problems of having a single server.

Because the infrastructure has evolved, the term and the definition of Jamstack are evolving as well. And the way we build applications and websites is changing because of the new versions of Next.js and new frameworks like Remix. Since everything is so fresh, we are not sure if that's Jamstack or that's not Jamstack, because Jamstack in general is an umbrella term for a lot of things. But, it is possible to build fully featured, very dynamic applications in a modern Jamstack way.

I want to understand edge functions better. It seems like it's run off of CDNs, but CDNs previously were, as I

understand it, just for delivering static content, so this brings a dynamic element. Could you explain that?
Yeah, that's a good starting point. CDNs used to be just for serving content. You couldn't actually execute any code on CDN. If you have serverless functions, it usually means that there is a location -- maybe in Europe, maybe in the US -- but it's usually one location. Unless you're an Enterprise, in which case you could have multiple locations -- but at a higher cost. For example, Vercel has this feature but only available in their Enterprise account. Needless to say, it's not for everyone. Usually it's just only one location, and it can be slow because you can't be fast everywhere.

But edge is the opposite. You can run code and run functions on the CDN. Cloudflare Workers is a good example of edge functions. Actually, Vercel uses Cloudflare under the hood too. With edge, you can be fast everywhere, execute some code, render HTML and pages for every user. It can be highly dynamic and very scalable.

Do you reach a point on your projects where, in terms of performance and scalability, it's better for you to move off of Vercel or Netlify and onto AWS or Cloudflare? Or do you see that it could happen, but not at that scale usually?

No, we don't have such situations. Both Netlify and Vercel are very scalable. Basically, there are no limits on how big your app can be, because they both run on top of AWS or Azure and other bigger services like those. Usually if you want to switch, it's because of a specific feature -- maybe pricing, maybe you have better deal -- but it's not about limitations of the platform itself, like executions, bandwidth or stuff like that.

We've heard people talk about why you might switch on the feature side, and I'm curious what are some of the places where you might want that extra control or extra customization that comes from switching to AWS?

Maybe you want to switch from Vercel to AWS if you want to use some other product of AWS, like machine learning. AWS has a lot of products from audio, video, and if you want to keep your full stack on AWS -- if you want to have only one platform, only one provider -- those are situations where we've seen some companies prefer to be on AWS -- “Okay, we want to have everything on Amazon, because that's our policy and

because we already use some products of AWS. So we want to, for example, use S3 and CloudFront as CDN, because that's how we do this." It's very hard to compete with AWS on products. They have basically everything.
But Vercel competes, I think, on user experience, on developer experience. It's much easier to start on Vercel, and you don't have to worry about some bad configuration or pricing. Sometimes if you do something wrong on AWS, you can have a very unexpected bill.

Say a client says they want to do everything on Amazon or CloudFlare. For you, how much time or expense -- or both -- does that add, and how big of a downgrade in experience would that be from using your more preferred process?

I think the closest you can have on AWS to Vercel or Netlify is AWS Amplify. Basically, they want to have similar features, so it is CI/CD -- you can just connect your repository, build an app and deploy it to a CDN. I think that would be our preferred way, using Amplify.

If the client wants to have a very custom setup for whatever reason, then it would add some time: first to create all of this and second to maintain this. If something changes in our stack, we have to be sure that it'll work, so we need to have a person who really understands AWS and all those details about configuration and maintenance.

Do you have a preference between Vercel and Netlify, or how do you think about that?

As a company, we don't really have preferences. We are partners with both of them.

I can't say Vercel is better because I like it better. There are some differences: for example, how Vercel supports Next.js. Both products are created and maintained by Vercel in the company, so it's a smooth experience.

They have some smaller features, like analytics, that are different in how they work. It really depends on client requirements and expectations. In some cases, we recommend Vercel and in some cases Netlify, but both are really, really good products.

I'm curious if, as far as APIs go, there are ones that you will always recommend. Obviously, you work with a bunch of headless CMS partners, but do you have certain ones -- say, around authentication or search -- that you always use, or do you not care as long as it works for a particular project?

There's no one thing we always use, but there are some things we use more often in projects.

For search, it'll be Algolia, because it's just easy to develop things using Algolia. They have a lot of good documentation and so on.

For authentication, Auth0 is okay. Netlify Identity, it's okay in some cases. It really depends on the requirements and on the project.

But yeah, there are some things we use more and some things we use less.

One thing that we've heard is developers really like Next.js and Vercel because you don't have to learn any DevOps. I'm curious if, as a result of that, you're seeing more front-end developers who don't know the backend at all doing this work. If so, is there any potential problem with that?

I think that these days the front-end developer is really powerful.

The front-end developer becomes a full stack developer, because it's very easy to build an app using some managed back ends and managed database, like, for example, Firebase or FaunaDB. You don't really have to know very advanced back-end stuff to build an app. Of course, if you want to be a really good back-end developer and build custom backends, then I think it's not a job for a front-end developer. You need both of them, because a good back-end developer usually is not a very good front-end developer, and vice versa.

But the front-end developer these days can build a really powerful backend because of the tools we have -- the Jamstack ecosystem and all those different SaaS products that focus on really good developer experience.

I don't think that's a problem. I think that's an opportunity for front-end developers and for agencies like ours. We are mostly front-end developers, but we can build apps and we can build back-ends. For me, it's an opportunity and a way to build really powerful and fast experiences in products and apps.

Speed is something that comes up a lot, as well as stuff like Core Web Vitals. It seems like performance is really a key factor and key trend with Google and SEO. Could you say a little more about the importance of speed and performance, and why that's so important?

I think that's important for a few reasons.

First is a business reason. If your website is slow, you are simply losing your customers, because people don't want to wait for page. If something is slow, they will just leave your website. So that's an opportunity to provide a really smooth user experience and create a product that people really like, recommend to friends, and return to. Performance is just one part of UX. Core Web Vitals actually focus on general user experience. It's about how smooth your website is, if it's not clunky, if the content stays and doesn’t jump for whatever reason.

I think the second reason is because of the cost. If your website is fast, you don't have such a high cost for the server, because if everything is faster, then you need less power to execute all of this. It’s obvious, but if you do less, then you pay less. That's a win. Another reason why we like fast websites and fast apps is the environmental side. If you have a faster website, it will produce less carbon footprint.

Those three reasons are the most important for us. The user experience is definitely number one. If you can improve this, basically everything else will be better as well. Like A/B testing -- if it's faster, then it can be more accurate, because you are eliminating the factor of speed.

You mentioned AWS Amplify, and I'm curious about other ways that you've seen traditional cloud providers like Amazon adapt to Jamstack, either with new products or otherwise.

Microsoft released Static Web Apps as part of Azure. It's similar to Amplify. I think that's their answer to Netlify and Vercel. Basically, they do a very similar thing, but it's 100% on Azure. You can connect to your repository, build the app and deploy to a CDN. But I think it's not that powerful yet. They don't have that many features, but they're working on it and releasing new features pretty fast.

In the case of Google, I don't think they use something similar. You can for sure build this, because the Google Cloud platform is powerful, but they don't have anything out of the box. I think Firebase would be the closest, but it's not the same, really.

Do you see companies staying on Amazon but moving to a Jamstack setup via Amplify, or similarly on Azure going to Static Web Apps? And are you seeing new companies start out on these platforms, or is it mostly on Vercel and Netlify?

From our experience, the most popular are definitely Netlify and Vercel. We don't have a lot of cases where we are forced to use AWS or Azure. I'm not sure how it looks like in general, but that has been our experience. In some cases, we have even migrated from Amazon to Netlify. But the other way, not yet.

One sense we have is that big companies may be unlikely to switch from Amazon to Vercel or Netlify because of perceived risk -- for instance, you’re not going to get fired for using AWS. Do you see that, or are people pretty comfortable that Vercel and Netlify are big names and that there’s not much risk attached to them?

What we see in our case is big companies -- like enterprise companies -- will try out Jamstack on some kind of pivot project, like “Okay, let's do something small and just test how it works. Is it sufficient? Does it work for us?” We don't really see big migration, like “Okay, let's do everything on Vercel.” I think the approach is to try how it works and maybe then create new projects on this platform.

I think that enterprise adoption on Vercel and Netlify is growing, for sure. We see a lot of enterprise deals. But I wouldn't say it's like, “We have to move from Amazon this year”

or something like that. I think big companies like to split the risk and just try something small and see what happens.
So it's more likely that they would just use the two side-by-side, and they would potentially serve two different purposes, like Vercel and Netlify for starting new projects and then migrating over time to AWS if necessary.

Exactly. Or the other way. Maybe they decide Vercel is better because of the support. I don't think AWS has good support, unless you're really, really big. Because those companies are a bit smaller, I think they can provide better support than big players.

We spoke about pricing briefly earlier, but I'm curious for your thoughts on how Vercel's pricing scheme works and also how it compares to Amazon's pricing. We saw a graph that indicated that, once you're above ten -- or some relatively low number -- of developers on Vercel, the pricing gets dramatically more expensive.

I know they are experimenting with the pricing. There used to be a very large gap between the Pro Plan and the plan if you had more than ten developers -- the switch was pretty big. Personally, I think they can be flexible. If you need one more developer, I think that would be fine. But since most of these platforms are built on top of something like AWS, it doesn't make sense for them to be cheaper than AWS. If they were cheaper, it means that maybe they have AWS for free or something, and that's likely not the case.

So what a company does is outsource the maintenance. You are paying for being free of DevOps and to have people who take care of the infrastructure and the support. That's something. If you want to have everything on AWS, you will need to hire some person who does this or does Vercel for you.

Generally speaking, I think the Jamstack approach is cheaper, especially at a large scale. The scaling is cheaper because of the CDNs, and serverless is becoming cheaper and cheaper too.

We've been talking about how Jamstack is changing, and how the concept of it is becoming a little blurred with the edge and all that. I'm curious to hear your thoughts on

where it's going in the next five or ten years. What does that change look like, and what trends are important?
I think the biggest trend right now is the edge, it’s already very powerful. Personally, I think that we will see more edge databases. You could have the whole stack running at the edge. And Vercel and Netlify for sure have plans: Netlify has Edge Handlers, Vercel has Edge Functions. They're invested in this direction. I think that will be the next phase of Jamstack’s evolution.

We see new frameworks, like Remix for example. That framework focuses on running at the edge, and they don't have static side generation. Instead you can render everything on request. It's kind of back to basics but with that twist of the Edge.

So the general trend I think will be edge and really fast code execution, globally. We've seen that it makes sense to push everything to CDNs because it's much faster and cheaper, but that used to be static. If that can be dynamic, it means that, “Okay, we can not only host static fast, but actually we can execute code.” So we don't really have to worry about the building problem, like big building times, because we can actually run on the edge. I think that will be the next big thing.

Disclaimers