EPISODE 1521

[INTRODUCTION]

[00:00:00] ANNOUNCER: This episode is hosted by Mike Bifulco. Mike works as a Developer Advocate at Stripe, and is also the co-host of the APIs You Won't Hate podcast. Follow Mike at mikebifulco.com.

[INTERVIEW]

[00:00:14] MB: Hello and welcome to Software Engineering Daily. My name is Mike Bifulco. I am your host. I am here today talking with someone who I'm very excited to chat with. As you may or may not know, I am someone who lives in frontend software development quite a bit. Sort of full stack development. It might be something that you've heard it referred to as.

And because of that, I am very, very excited to chat with my new friend here, Alex Kondov of Tao of React. Alex, how are you today?

[00:00:39] AK: Hey, Mike. Thank you for having me.

[00:00:41] MB: Yeah, of course, of course. Happy to? Why don't we start here? Obviously, I've sort of pitched a little bit that you work on React things. You've done some React stuff. You've written a few books about Node and React and the software world. Tell me a bit about your career. Tell me how you got into writing. And tell me some of the things you're interested in software as well.

[00:00:59] AK: Where do I start? I guess we can go back 10 years ago. I started working with PHP back in the day. I was writing mostly WordPress and Laravel applications. And then slowly around the time where when React and Vue started getting popular, I started dabbling a little more into the visual side of things. I've always been a visual person. I like both logic, but I like seeing the result of my work.

And when I started getting more frontend-oriented tasks, being able to see something painted on the screen and moving and getting animated, it was very rewarding. I started slowly moving more and more and more towards frontend development until at one point I decided, "Hey, I just want to go all-in on that. I want to work with React. This is what I want to do." I started working with React around 2016 or 2017, somewhere around that. It's been a while.

[00:01:56] MB: Yeah, early days for React.

[00:01:58] AK: Yeah, yeah. I had a lot of fun. It was very exciting to see the whole ecosystem evolve and grow before my eyes. And as time passed, it was somewhere around 2020, yeah, when COVID hit. It was the first winter of the pandemic. I was getting pretty burned out at that point because I was pretty much doing the same things for years now. And I desperately wanted some change in my life.

And to add on top of this, the fact that like you couldn't go out, you couldn't go to the cinema, you couldn't go to a restaurant. And the only place where I could get some variety was at my work. And I decided to switch languages, actually, back then. And I decided that I'm going to go work with Go for a little while. Go back to backend development. See what's going on there. I really wanted to work more with like Kubernetes, and Kafka, and tools like that.

But before I switch to backend, I wanted to write everything I've learned about React somewhere. Basically, just get all my knowledge. Put it somewhere. Leave it for other people to find it. Or if I decide to ever write another React application in the future, to have like a starting point.

Because I've been working with this technology for a few years now. I didn't want to leave all this knowledge behind. I put this really long form article called Style of React over Christmas. And I think somewhere in January, the first days of January 2021, I put it on Reddit and Hacker News. And I went to bed.

Next morning, I wake up, and there's like a lot of people that, I don't know, messaging me about this and like a whole bunch of comments. And people talking about what I've written. And this is the first time that something like this has ever happened. And I think it was one newsletter actually that they shared my article and told me, "Alex, you should just turn this into a book." And I was like, "Okay. Might as well do that."

It took like six more months to expand it, to write – to expand the examples, expand the descriptions, add more things in it. And, yeah, I published it. And after that, I published another one called Tao of Nodes, which is focused on – it's in a similar format like Tao of React, but it's more focuses on backend development or REST APIs.

[00:04:15] MB: Yeah. Wow! Yeah. That's pretty incredible. There are a few things I want to digest here. It sounds like you, overnight, woke up to a big interested audience who suddenly found all of your writing really interesting. Can you tell me a little bit what that was like? What was your feeling, I guess, the day before? Did you feel like you had much of a sort of people listening to you online for your expertise?

[00:04:34] AK: Well, not really. At that point, I'm not really famous in the software engineering community. I have a few people that follow me. But that's not that many really. At the point, when I published this article, I had like 140 followers on Twitter, I think. Somewhere around that. And I've been trying to be active on LinkedIn before that and on Twitter, but mostly just to connect to other engineers. Learn new things that are happening at the software engineering industry. Share knowledge and talk about fun stuff.

To me, posting this article was just another thing that I'm doing. I didn't expect it to gain any traction. Because I had been writing a lot before that. Maybe for the last two years before I posted this, I've been writing articles on my blog frequently showing them on social media for other friends and people who could know me to read. But it's been mostly my audience was very limited before that. I didn't expect any traction to come out of this really.

The next morning, I woke up, and there were like a lot of people. When I say a lot, they weren't that many. But they were a lot for my standards. And it was a bit daunting at first because I expected to open the comments and to see a lot of unpleasant stuff written about why does this guy think this way? Or why does he say this thing? This is not how it should be done. And there were some negative ones. But I think that I learned some things from them as well. But there are a lot of positive and supportive comments. And many beginners actually reached out and told me, "Hey, this is a very good resource. It really helped me to put some of the more chaotic thoughts and ideas that I had about React into order and be more confident when I'm building things." Yeah, ever since then, every now and then, I get a negative comment or more hateful comment. And this is still not something that I'm used to.

[00:06:30] MB: Yeah, I can relate to that.

[00:06:32] AK: Yeah. Every time someone tells me that one of my opinions is not that good, and I'm trying to avoid saying an explicit word here, but I still take it hard.

[00:06:46] MB: Yeah. You know, it's funny you say that. I publish quite a bit online as well. In addition to hosting the show, I'm a Developer Advocate at Stripe. And I publish a bunch of things under my own site on my own website site. And one of my favorite things about publishing is when I say something, like, I know what I'm saying. Like, "Hey, this is the right way to do this." And immediately someone comes in and says, "Actually, you got that wrong. And here's why." I mean, it is hard to hear that feedback. But I'm a big proponent that like the best way to figure out if you know something is to tell people you know how to do it. Because the smarter people show up. They always find you and they'll always say, "Actually, did you know this?"

It's a really good way to learn things. But I think it's also a good way to establish yourself as someone who knows how to listen, and take feedback, and can deal with criticism. Although, definitely, the core bottom of my heart every time I get that feedback, my first response is to sort of seize up and like feel very defensive too. I think that's very human.

[00:07:37] AK: Yeah, I know what you mean so well. Actually, when I had a very long thread on Reddit with one person who was disagreeing with me on a very important point actually in Intl React. And I think I didn't mention what the article actually was. It was a collection of my principles about building React applications. It was wasn't about how you work with React and like what state is, or how to write JSX or things like that, but mostly about how you structure your application. How you organize your component folders? How you organize the components itself? How to name things? And just maybe the meta, the secondary principles that kind of help you build better software. The things that you don't learn in the tutorials.

And one of the points that I had was about exporting the component at the bottom of the file. This is the thing that many people argue about.

[00:08:29] MB: It is. Yeah.

[00:08:31] AK: And before I wrote the article, I was a big advocate of having the export default statement at the bottom of the file. Because to me, this was like the signature. You go through the whole file and you see the most important thing that is getting exported is this. And there was one person, I don't know him, him or her, but they expressed in a really long form format their thoughts about why default exporting should be done at the top of the file. And we got into this very serious conversation about exporting. That was like six or seven, I think, really giant messages long. If you find the original Tao of React post, you're probably going to see me and this person just going at it about exports.

And at the end, I was kind of convinced that exporting at the top was better, to be honest. And this is something that – This is an example of how a person on the internet changed my mind about something that I had strong opinions about.

[00:09:32] MB: Yeah, I think that's the mark of an experienced person on the internet, is to be able to have the good discussion and decide when it's time to change your mind. And honestly, the truth may not be that either of you are exactly right. But it's worth weighing the pros and the cons of things.

[00:09:47] AK: Yeah, yeah.

[00:09:48] MB: I saw a really interesting tweet, I think last night actually, from someone talking about TC39, which is like the JavaScript working group that defines the specification for JavaScript. And how if you got hopped into a time machine and looked at what TC39 was talking about at any point in the past, you would see discussions that sound a lot like whatever it is you're arguing about now, but with something that you have since like assumed this is the only way to do it. Because these discussions happen so often. And people are talking about these things over time. And eventually, we land on something that the industry is generally comfortable with, maybe with some exceptions here or there. But I think that's also part of the process of moving software forward, is like we get into these debates and discussions and decide the meta things about how to build our apps. What we like and what we don't like too.

Yeah. You've suddenly found yourself in the spotlight about this. And you're now – I don't know. Maybe whether or not you want to be the de facto expert on the meta discussion around setting up React, as well as Node, right? You've written two books now. Is that right?

[00:10:43] AK: Yeah, yeah. Earlier this year, I published my second book, Tao of Node. After the success of the first one, I decided that I have more things to share. Because I've done plenty of backend development throughout the years as well. And I've learned a thing or two about building REST APIs. The book is mostly focused on REST APIs. And some of the ideas are very similar to the ideas that I've spoken about in Tao of React. But they are translated to the node environment.

And the book is – Maybe Tao of Node is, if I could summarize it, it's mostly focused on telling you how to extract your business logic from the rest of your application code. And in frontend development, this is a bit harder to do, in my mind. Because like the logic that visualizes the components on the screen is often tightly coupled to the logic that decides exactly what data is getting visualized.

In backend development, you can more easily put the boundaries between these two. You have a clear HTTP layer, something that the controller, for example, that accepts the response. You have the data layer, the database logic that persists or retrieves data from somewhere. And then you have that middle part, this place where your domain or business logic resides.

And one of the points that I'm making there is that the most important thing that you can do for your software design is to create such a layered structure in which those three pieces, the transport, the /HTTP, the domain, and the data, they are not intertwined. They're not connected to each other.

And the funny thing is this is not really something that I have come up with. It's not an original idea. It's something that people have been pushing for a long time with things like hexagonal architecture, or onion architecture and things like that. But I feel that some of the architectural ideas we have in software engineering can be very dogmatic about exactly how things have to be named. How things have to be structured? Exactly what has to be passed to this and that?

And I am trying to tell people that the abstract principles are more important than the specific implementation. And if you understand that, what's important is that you isolate your business logic. Because your business logic changes a lot and you don't want these changes to affect everything else in your application. This is an easy way to put it. This is a principle and idea that everyone can follow regardless of what the environment or framework they work with.

Yeah, this is something that I wanted to do with Tao of Node. I published it, I think, sometime this summer. Maybe in June. And it's got good feedback as well. And, yeah, I'm still undecided what my third book is going to be about. I do want to continue writing. But I think I'm at a point where I've said most of the things that I know about software development really.

[00:13:40] MB: Yeah. Well, the commonality between your two books that I think is really interesting is you've done a good job of zooming out a little bit on the problem. And you're not arguing about whether we should use tabs or spaces to indent. You're saying, as long as you indent, you're getting the job done. And the important thing is sort of that part of the discussion.

I don't know if I could plant a seed with you. Maybe your next book should be on how to see the bigger picture, which I think is probably a little harder to write. But it's an interesting framework for thought that seems to have gotten you a long way.

[00:14:05] AK: Yeah. Yeah, that's a very cool idea. I don't know. I've been thinking about a lot of things. Many people have reached out asking about more specific examples of some of the architectural ideas that I have written about. I'm considering writing a book that takes you from an empty folder to a complete application following these principles. I still haven't decided if this is going to be it. But I do want to continue, do want to continue writing. And I think I found a good intersection between the two things that I really enjoy between writing and programming. And being able to do that is like a dream come true. And I'm still fascinated by the fact that people read what I have to say. It's still hard for me to accept that.

[00:14:52] MB: Yeah. Well, it's a good writing, rest assured. I've done some digging in and dabbling on your Twitter feed and your own website and things like that in preparation for our discussion here. And you have a very nice way of framing your discussions that is sort of not antagonistic, but based – And the substance of the discussion comes first, which I really like. It causes you to – I don't know. It's almost disarming in a way. I've found your articles to be really interesting for that reason.

One of the things I'm interested in hearing your perspective on in both Node and React, one of the things that strikes me about those worlds is that they change so rapidly, because they're both ripe for animation. There's lots of people building with them. Lots of things going on at all times. Even as we're talking the thing that Twitter seems to have been a buzz with lately, is the React proposal for a used keyword and a cache keyword and things like that.

[00:15:39] AK: Yeah, yeah.

[00:15:39] MB: What's your strategy for keeping up with the news or keeping up with what you need to keep up with to be productive in React, and in Node, and those other things?

[00:15:48] AK: Yeah. something that I've learned with time is that productivity doesn't necessarily come with the latest features of a framework. This is something that I do want to highlight. You don't need to be running the absolute latest version of React at the day in which it's released. There are many positives to running like the latest version, bug fixes, security, patches, better performance, probably better APIs. But what you have running in production can and will run for a long time regardless if it's using the latest version. And you can maintain it and you can support it even if it's not updated. It's my job right now.

There are a couple of – You could consider them legacy applications, which still use the React class syntax. And this is something that you rarely see nowadays. But they still use it. They're written extraordinarily well. They're well documented. There are good abstractions built into them. And because of that, there is no real reason for us to go in and rewrite them from the bottom just because there are better ways to express these ideas.

Now with this said, you most likely want to keep up your software up to date because of the the reasons I outlined earlier. And also, when new people come to your project, it's easier when you tell them that they can open the latest React docs and learn from there instead of pointing them to a specific release or a specific version or something like that.

And another benefit of using like the latest version of something is that the whole community is creating abstractions and libraries and other tools that are kind of related to that.

How I have managed to solve this? Now, this is a tough one. I don't think I have. I have the benefit of working on a team that is building developer tools for other engineers right now in my company. I work for News UK. This is one of the big media corporation.

The team that I'm in, we build React libraries. We build a design system and large modules that are then used by other React engineers in the organization. It's part of our job to maintain everything and to keep it updated to the latest version, because we want people to be able to – when they scaffold the new project, for example, to use the latest React, latest library versions and everything. And we have to maintain support for everything new that is coming out. It would be hard for me to give an advice for people who are not in this situation. Because every time there's a new version, I can find the time to update whatever we are running.

My practical advice would be just to put a ticket in the backlog if you're using some sort of – You're most likely losing some sort of a ticketing system. Don't wait for permission. Put it in. And if possible, split your updating work into like small iterative steps. If your application is – Maybe if you're running microphone ends or your application is modular and you can kind of isolate the changes and do the upgrades module by module, that's a lot easier, because you don't have to block everything for two or three weeks on end to do it. But, yeah, at some point, you will have to. The longer you wait, the harder it would be.

Just sneak a ticket in the backlog. Don't really wait for permission. Tell your tech lead. They're probably sold on it as well. But it's going to be easier for them to convince the business people that you have to do this. But yeah, there are many technical benefits to keeping everything updated. It gets harder to sell those benefits the larger your software grows. Because the larger it grows, chances are the more successful it is if it's getting bigger and bigger. And the more maybe requests for features or changes you're going to get.

And maybe on your first question, something else that I can say is how to make your work future proof. How to make sure that whatever your writes won't have to be rewritten with the next version. One way to do it is with abstractions. Changes are hard when they have to be done all over the place. And if you can somehow isolate these changes in a specific file or in a specific module, this makes it – These limits the blast radius of any large refraction that you have to do.

For example, one way to abstract business logic is through custom hooks. And in your custom hook, you can write the logic to fetch. Some data to parse it in any way that you need to and then return it to the component in a parsed state. The component doesn't have to know exactly how this data came to it.

And the example is that there's an example that you gave with the use keywords, the new proposal in React that is going to allow us to unwrap promises easier. If you have an abstraction that deals with data fetching, you can potentially only replace the underlying functionality inside of that abstraction. Maybe you could be using an HTTP client, like Axios, directly. And you can replace that or wrap it in a use call, or wrap it in a React query call, or something like this. But if you have this in place, it will only be – This is the only place that is going to be affected. Then your component is still going to receive the logic the same way. It's going to follow that contract. The data structure that it receives is not going to change.

The harder question to answer is how do we know what abstractions to put into place? Now, I don't think we're going to have enough time to –

[00:21:26] MB: Yeah, that could be lifetimes of discussion.

[00:21:29] AK: Yeah, I don't think we can figure this out here. This goes back to the principle that I said earlier, because we don't know exactly what will change. But we're sure that business logic will change. Make sure the business logic is cut out from everything else.

[00:21:42] MB: Yeah, yeah, that's a very good lesson, I think. And it's one of those things that, as an engineer, maybe you're early on in your career or starting in your career, it may seem like it slows things down to add like why am I writing this extra file to go do this thing every time? Or why is this happening somewhere else where I can't see it? Why can't I just write the thing to go drag my users out of the database and put them on the page here? Or whatever the case may be.

I think when you've done it sufficiently many times and you've felt the pain of having to go make that change in 40 places, 200 places, a thousand places, whatever it is, you start to get the sense of what that meta pattern feels like. This is the thing that I don't want to have to redo again. The business logic is its own middle layer there. That kind of makes sense.

I think those are wise words, and I think it's something that's hard to earn and a little hard to explain in a sense. But I think you did a really good job of explaining the value and definitely one of those things that pausing and taking a breath and like planning a little bit before you go and build something is really helpful for that too.

As far as building software that's maintainable, especially like working within a team, do you have any – Gosh! Do you have any preferences or maybe lessons, best practices, whatever, that you've seen from going and taking like, "Hey, I want to put together a proposal for my team to go move from Axios to React Query?" How do you convey that information to your team? How do you express the value in that change in a way that is meaningful and won't just be falling on deaf ears?

[00:23:04] AK: Yeah, yeah. One thing that really helps when it comes to this is if your proposal truly solves a problem that the team or the business currently has. Because there are many things that can be improved in your project at any point in time. Maybe you have some performance optimizations that you can do, like images, or bundle sizes, or something in terms of your structure. Maybe you're doing too much prop drilling or something like this.

But the reality is that we have finite resources, finite time. We've got a growing backlog of things that we have to do. And every time you want to do something that has questionable rewards that you can get from it, you're going to experience pushback. And this is the sad reality of when software and the craft meets the business. That's just how things are. It really helps if your kind of proposal is kind of backed up by specific data and something that just proves that this is going to resolve a problem.

In the React query example that you gave with Axios, you can say something like – You can give an example with a previous bug that you have encountered related to data not being re-fetched. Or maybe you're firing too many queries, and this was kind of blasting your backends too much. Or maybe you had a problem with invalidating when exactly you should be getting new data and this was causing something on the screen to show old states, and then like show all state for a second, which is not consistent with what you have in the backend. And you can try to quantify the impact on users and say, "Okay, this is something that we can alleviate and prevent from happening in the future by switching to this library that has these things built in. It's going to save us time because we're not going to have to implement these features on our own."

And we step on the shoulders of an important tool in the industry. It has support for everything out of the box. We don't have to write this logic. We don't have to maintain it. Chances are that we're going to mess it up. But these people that have built this, they know what they're doing. And it improves our chances of riding a more stable solution. And you can keep going. You can say that a lot of people know this library. They know the APIs. You get a new team member. They can resolve books and write features faster with it. To me, it has to be backed by some either business or tech value that you are improving with this.

And if you say, "Okay. What if I have a proposal in mind that is not backed by anything? It's just something that's nice to have?" And like as someone who's been leading a team for a couple years now, I would say, "Well, is this the right thing to do?" There are a lot of things we can improve. But if there is no – If we can't get any value out of this long term, or if it's not a strategic thing that is going to unlock new possibilities in the long term, maybe this is not something that is worth doing.

And it's very hard for me to say this, to be honest. Because for many years, I was on the side of we should focus on the craft and do things in an excellent way and everything. But the more I write code in a professional environment, the more I realize that there is some balance between the craft and building a product.

[00:26:33] MB: Yeah. I can feel the experience sort of coursing through your words here. It's, for sure, one of those things where sometimes writing the most up-to-date, beautiful, syntactically perfect code still doesn't pay the bills. And sometimes you just need to achieve the business objective and move on to the bigger problems or whatever is in your team's way. But providing context and value and like shifting bigger functionalities. Like, moving an entire fetching line library over is useful if it solves a significantly sufficiently large problem or is easy enough to replace, whatever the case may be there. Yeah, fascinating. Fascinating. Cool.

Okay, I want to shift gears a little bit too. We've talked a little bit about you've written two books. One on Node. One on React. We've talked a little bit about maintaining software and how you keep up to date with news on React and your sort of strategies for not jumping into the newest thing too quickly. I'm a little curious about, if you can describe to me, how do you learn? What's your process for digging in and finding new things? Do you have a process for note-taking for how you consume information? Are you a visual learner? That sort of a thing? I'd love to hear a little bit about that.

[00:27:39] AK: Yeah, to be honest, no one's ever asked me this. So, it's going to take me a while to formulate an answer. But I take a lot of notes. I have this thing called the reMarkable tablets. You've heard of these, right?

[00:27:51] MB: I have, yeah. It's like an e-ink-looking iPad sort of thing.

[00:27:54] AK: Yeah. I was very skeptic about them, because my problem is that I really take a lot of notes. It helps me think. I don't know what it is with writing. But when you have to put structure into your thoughts and put them on paper, you really have to understand what you're thinking about. Because sometimes you have these abstract back ideas that run in your mind. But if you try to really formulate them in a – Compress them to words, it turns out that maybe it wasn't exactly what you're thinking about. The idea is something else.

I write down a lot of notes. And when I'm not clear about how I want to achieve something, I usually take a piece of paper and I start writing on it. And looking at them here, I think there are about six fully written notebooks that I have on my desk. And the problem with them is that, every now and then, I want to find something in them. And this is just a brute force search. Linear time. Just go page by page. I don't know when I wrote it. I don't know where it is. And this is, yeah, not ideal.

My girlfriend bought me this reMarkable thing that, as I said, I was very skeptic about them, because I thought the feel wouldn't be that good. I'm not sponsored by them. But I just want to say that it's a great product and it's the closest thing that I've got to reel than on paper.

I write a lot of things down. And usually when I'm reading about a new technology or reading a book, I do take a lot of notes. Often, I don't go back to them. There are only a few books that have inspired me so much that I go back through my notes and read what I've written about them. But I don't really go back to them. But just the fact that I have to formulate an idea into a sentence and put it there helps. I even do that for when I'm studying a new technology, I write down a lot of stuff. When I'm studying something else, I do it as well.

But the thing that, really, I need practice to remember something. After I've read the docs, I do have to install whatever framework library tool I'm using just on my machine and play with it for a weekend. Because when you go outside of the limited scope of the tutorials, you find out that there's a lot of gardening equipment waiting for you to step on it and bash yourself in the head. And I think that these are the valuable lessons that you can learn. Because the tutorial is kind of the golden path. But the moment you decide to do something else, that's when the interesting things happen.

And when I'm learning a new technology, I just install it. I take a look at the basic syntax. And then I just decide, "Okay, I'm going to write this thing with these constraints. And I just do that." And I look for information case by case. I don't read the docs top to bottom. I decide, "Okay, this is enough knowledge for me to start." And then I start building a page. And then I reach a point where I don't know how to do this. So, I go and find information online. Well, much like I would do in a regular working day, regular working environment.

And I think that practice helps me retain the most knowledge. And not necessarily practice, but problems. When you see error messages, when you have to spend time thinking deeply about a problem, I think that's what helps me learn the most. But yeah, yeah, I'm just a practical person. I have to touch it with my hands to write some code. And that's how I remember the best.

[00:31:14] MB: Yeah, I like that a lot. I think it's sage advice. I will say, I've been in engineering for a very long time at this point. And there was a moment in my career where someone asked me what I used to take notes. And I kind of like did a double take and I was like, "I don't take notes. Why would you take notes?" It felt very foreign to me, that it was just like, "No, I'll remember the syntax. I'll remember how to do that."

And when I started to take notes sort of formally for myself, not for meetings, not for handing off to other people, but just as a means of offloading things from my brain, they really expanded my ability to not only recall things that I had seen in the past, but also just to keep thoughts organized and to like get through abstract tasks more quickly.

If you and I were sitting down to write a new book on, it doesn't matter, building a picnic table, I think the process of formulating what that book would look like would go a lot more quickly because I'm used to writing notes and making a structure for the way I think about things.

And for new engineers, for people who are at any point in their career, if you haven't started the process of figuring out how you take notes, whether it's graphically, or with text, or audio, or whatever it may be, it's worth exploring. It's something that really is quite empowering when you get into it.

[00:32:19] AK: Yeah, yeah. And something else that I really think helped my career is keeping a blog even in the years when no one was reading. I would suggest to every – Especially to beginners. Whenever you face like a problem at work, whenever you overcome some sort of adversity, no matter how simple this seems to you right now, do document it on a page. Just write down what happened. What you did. Put it on your blog, and you will thank yourself in some time. Because this forces you to go back through the problem step by step to really think deeply about what happened. Because at some point, you realize that it's not – having a working solution, having working code and working software is not enough. You need to understand why things happen. How they happen? And whenever you face an issue, you don't want to waste time fighting that again.

And just writing down everything that happens helps you to go through it, understand it. And the fact that, to me, it's not enough to just write it somewhere. You have to publish it. Because when you know that other people – that there's a chance that someone else reads it, you go that extra mile. And you make sure that it really has meaning and it really is representative of your best work.

[00:33:35] MB: It forces you to validate your thought process.

[00:33:37] AK: Yeah, yeah, yeah. Yeah, if I can give one piece of advice to a beginner, just have a blog even if you're the only person who reads it.

[00:33:45] MB: Yeah, yeah. That's great it's like you're reading a book to me about myself. I think that's really good advice. A reflex that I think is important to have is anytime you find yourself looking up the same thing more than once, it probably means it's something you should go right down. And even better if you can write it down and publish it for someone else to see. That is the theme of a lot of the articles I've written it's just like, "Here's a little thing that I had to look up." And it was annoying that I had to look it up. So I spent an hour writing an article about it. And now I don't have to look it up again because I know it's on my site. But someone else will benefit from it too, you know?

[00:34:15] AK: Yeah.

[00:34:15] MB: Alex, let's talk a little bit about the future then in a couple of ways. I'm curious in maybe what you think you're interested in working about next. I also kind of want to hear like if you could pick anything, if you woke up tomorrow and you had a big amount of money on your desk to build a team and go build something, or write your next book, or whatever it was, what would that be?

[00:34:32] AK: Okay. This is the second time today where I'm asked the question that no one's asked me before.

[00:34:39] MB: Keeping you on your toes over here.

[00:34:41] AK: Yeah. What's next? I told you that kind of I foreshadowed a little bit that I think that I've said most of the things that I have to say about programming. I'm wondering what to write about text. And in this process, I think I have to build a product or two, an app or two, that are more challenging to make sure that I really have something to talk about. And this is something that I have started doing now.

I am working on a set project of mine that I want to release a better version of earlier at some point later this year. It's going to be a new kind of a writing community. This is something that I'm really passionate about. I'm really passionate about writing about journaling. And I'm even writing a fictional book right now. It's probably not going to be as good as Tao of React. My ego has accepted its potentially low literature value. But that's a personal ambition that I want to fulfill.

In the near future, I'm going to be building things, because I don't want to become the type of engineer who is very detached from software. Because I don't write this much code at my day job as I used to. I still write, but I spend most of my time talking to people or going over architectures and things like that. And I don't want to lose that feeling of creating something from a blank text file.

I'm first going to build this writing community. And then I'm going to build something else. I don't know what it's going to be. It depends on how the first project comes along. But, yeah.

And if I have like a bunch of money tomorrow, and I have to build something with them, I would – I'm not sure if I have a good answer. But do you remember those build your own adventure books?

[00:36:32] MB: Yeah, I do.

[00:36:33] AK: My dream is to build something like this for the web, but a really kind of complex application that will allow you to make really nuanced and detailed decisions and contribute to an evolving story with everything that you're doing. It's very abstract in my mind still. But it's something that I love these books when I was a kid. Every now and then, I remember how I – I always cheated, to be honest. Because when you have two options, I would kind of go see, "Okay, I don't like this result. So, I'm going to go together.

[00:37:10] MB: I did that too, yeah.

[00:37:13] AK: But I want to translate this into a virtual environment. And if I wake up tomorrow and I have all the funding in the world to do something like this, I will probably focus on that idea. But as you can see, it's still too abstract in my mind.

[00:37:26] MB: Well, hey, you never know if you're going to wake up with that funding tomorrow. If it shows up, be sure to give me a call, you know?

[00:37:31] AK: Everything is possible.

[00:37:33] MB: Yeah. Alex, tell me where our listeners can find you online.

[00:37:37] AK: I am @alexanderkondov on Twitter. And you can read my things on alexkondov.com. And you can also find me on LinkedIn. I am Alexander Kondov there as well, going by my full name.

[00:37:53] MB: Great. And we will make sure to put links to your profiles in the show notes for this episode as well. What about the books? Where's the best place to find your books?

[00:38:02] AK: They are linked on my website. But you can also go to taoofreact.com or taoofnode.com.

[00:38:09] MB: Brilliant. That's great. Well, Alex, thanks so much for chatting. It was really great having you on the show today. I appreciate hanging out. Thanks for being a guest.

[00:38:16] AK: Likewise. Thank you for having me, Mike.