EPISODE 1373

[00:00:00] KP: Angular is a free and open source web application framework. It's maintained by the Angular team at Google. It's used by millions of web applications and has a strong ecosystem of core contributors and library builders.

In this episode, I interview Minko Gechev, Developer Relations Lead at Google. We explore several aspects of open source software development, Tensorflow.js, Angular, and a few other things worth sticking around for.

[INTERVIEW]

[00:00:30] KP: Minko, Welcome to Software Engineering Daily.

[00:00:33] MG: Thank you for having me.

[00:00:34] KP: Can you tell me a little bit about your journey as a software developer?

[00:00:38] MG: Sure. Should I start from the very beginning when I wrote my first program? Or are you mostly interested in what I'm up to right now?

[00:00:45] KP: Oh, I love the full journey. If it starts in QBasic, or wherever, let's do it.

[00:00:49] MG: Okay, yeah. It started in high school. Back then, I didn't think that a normal person can write a program and send instructions to the computer. I thought that this was incomprehensible for me, literally. And I had classes in software developments. In Bulgaria we call it informatics. I'm from a small town in Bulgaria, Eastern Europe. And we use Pascal to just implement a very simple Hello World application. So this was fascinating. This was kind of mind blowing.

After that, we move to a little bit more advanced programs, for example, just some equation solvers. And from there, I got really excited and passionate about software development. So

with my high school teacher, he knew he knew PHP, JavaScript, and SQL. So we created the entire web system, which use MySQL's database.

It took me a while to understand what's actually going on. But I was really excited to know that I can create things. So this turned into an obsession. I remember how I used to dream that I'm creating software and I was solving some of the issues in my head or in my dreams, literally.

After that, I went to college, where it was a pretty constructive environment, a lot of people with similar interests like mine. I was really into Linux back then. So I was spending thousands of hours configuring my vimrc and my .files, which are still somewhere online. Went into **[inaudible 00:02:25]**, into C#. So, yeah, I had to have a virtual machine on my laptop back then. C# had more specific requirements. And eventually, I got introduced to open source from some local communities where I found the whole idea of open source to be very interesting.

Originally, I was probably more excited about the visibility and the validation that you're getting out of it. But over time, I saw more and more the value that it brings. So I continued contributing to open source around my open source contributions. I started a company here in Silicon Valley, and ended up moving to work on open source full time at Google about three years ago.

[00:03:11] KP: What's specifically about open source is attractive to you?

[00:03:16] MG: A couple of things. It gives everyone the opportunity to collaborate and learn from some of the best software engineers out there. As a college student, as a university student in Bulgaria, I was really interested in growing my skill sets. And there were a lot of great software developers around me. But still, there were software projects on another level, such as, let's say, the Linux Kernel, or back then, AngularJS, which was developed by companies like Google or large organizations like Linux Foundation. And I was really able to collaborate with some people who are so much invested into software, and I was able to get a lot of their ideas, learn from them, gets cold reviews. Get pretty much mentorship for free just by helping out with the tasks that they don't find too exciting, that we're waiting for contributors in their issue tracker. This was one of the things. So it gives equal chance to everyone. It doesn't matter where you are in the world. If you're passionate enough and, I'll say, motivated, you can work with some of the best software engineers out there. And you can get free mentorship.

And the second thing is from inspirational perspective you're getting. You can get inspired by open source projects. Everyone is passionate about something particular. And there is a chance that you would find something about this particular area, this specific domain on GitHub. So you can look up topics that you find interesting. And I'm sure there will be something on GitHub that is going to align with your interests and is going to bring you some extra inspiration.

[00:04:49] KP: Well, to be working on open source full time sounds, in many ways, like a dream job. But from a naive perspective, it seems like it doesn't make sense for Google. Why would they pay you to write the code and give it away?

[00:05:01] MG: Yeah, that's a really good point that you're raising here. There are quite a few benefits for Google itself. None of them are really directly related to the revenue that an open source project would produce for Google. All the benefits are pretty much transitive. For example, it increases the reputation of Google in the software development community. If we're developing popular platforms or popular products that are used by millions of developers, this increases the trust in the brand. Also, it helps us hire people. We're using the project I'm working on at Google Angular. We're using it heavily within the company. And if we make this project and we maintain this and it has higher adoption externally, then these developers could move to Google eventually and become productive from day one.

And making the project open source has a very critical part of the adoption itself. These are just two of the benefits. And there are many others. Developers in the open source community are constantly creating new libraries and resources in, let's say, for example, blog posts or even answers on Stack Overflow that are benefiting Googlers and external developers as well.

And at the same time, it's also a good opportunity to give back to the world in certain aspects, for example, software engineers. Inspire them by sharing the whole implementation for free. This inspires other projects in the ecosystem, which are using similar ideas or solving similar challenges.

[00:06:41] KP: And what are some of the contributions you've made to Angular?

[00:06:45] MG: Originally, the very first contribution I made was adding just a colon, and missing colon in a comment. That was back in 2012 I think. Since then, I remember how nervous I was when I was opening this pull request, because I didn't think that people from Google, they could miss a semicolon. So I spent a couple of hours opening this pull request. Closing it, and after that, reopening it, because I messed up something around like the git history. I don't know what I could mess up with just a single line of change. But I think I did.

And after a couple of years later, I started doing absolute AngularJS back then. And I found that there are some gaps in best practices. So I implemented – Rolled style guide with best practices, which turnout to you've got quite some traction in the community. And later on, when the team moved to Angular, started developing the new framework, Angular with TypeScript, I worked on a book and there were literally no resources on learning Angular back then. So I had to get familiar with the code base with implementation in order to get better understanding of how the whole thing works.

And to get better understanding of the implementation, I started contributing different, let's say, small feature requests or issues. I was getting this mentorship from Googlers, who were helping me with reviewing my pull request, and I was helping them back with fixing some of the issues that they were considering, let's say boring, or that they were not as excited in fixing.

And from there found other gaps in the ecosystem. For example, build a tool for static code analysis for Angular, which later ended up being adopted by the Angular CLI and by Google itself. And that's how the Angular team got interested in me and invited me to go through the interview process.

[00:08:39] KP: Very cool. Well, on paper, it seems like if you can make a pull request, and it can be accepted, then you've made a change. But I imagined for a project as big as Angular, there must be a little bit more process to it. Can you speak to, I don't know, any of the internal ways that things are looked at when pull requests come in? Or maybe how you encourage people to submit good pull requests?

[00:09:02] MG: Yes, yeah, that's an excellent question. That definitely could be really challenging, especially when someone wants to implement a massive feature that is not

necessarily very much related to the vision that the team has for Angular. This could be pretty frustrating. I can imagine how someone can spend months in implementing something that they're really passionate about, really complicated. And at the end, this pull request can be well-structured but may not align with how Angular is envisioned to grow in the future.

So the couple of guidelines here are, first, if there is a large feature proposal from someone, they should, for sure, first, go with opening a feature request that would go through discussion from the Angular team and from people from the community. We recently introduced a feature request process that includes voting. So if the feature request gets enough votes, then we're going to consider it. We're going to see whether it aligns with the future version of Angular. And we're going to decide how to move forward with it. So that's very important part of the process, for sure.

If folks don't necessarily want to contribute with a large feature, they can just open a pull request for a small issue, let's say. And the important part there is to first file the issue or find an existing issue, because this might be unexpected behavior. Who knows? And from there, open a pull request or for states that they would want to be working on a specific thing so that there no two people in parallel who are developing it. Because this way, the effort could be wasted as well. And make sure there are no already existing pull requests with the same topic.

And from there, we go through a regular triage process. So we're going to look at the pull request. We're going to probably have quite a few back and forth because of coding style or because of backward incompatible behavior. And eventually, we're going to merge the pull request and it is going to go in and it's going to be used by literally millions of developers.

[00:11:09] KP: So developers have lots of choices today, especially on a greenfield project. Why did they pick Angular?

[00:11:16] MG: A couple of the reasons for that are that's Angular accounts opinionated. So if you'd want to build an application with Angular, you must use, for example, TypeScript. We believe that static typing is a very important part of the development process, especially if you're planning to build a production app that is going to scale. Angular also comes in pretty opinionated. I'll say that there are still a lot of decisions that could be made to make it even

more opinionated. But we're trying to find out the right tradeoffs, the right line from where we are opinionated enough and where this opinionation is causing inconveniences for developers.

But Angular is definitely opinionated. We have a platform that integrates well together. And we have the integration of this platform with the Angular style guide, which discusses best practices such as, let's say, file, directory structure, naming things, and so on, and so forth. These are two of the benefits. Others are that we have a pretty massive ecosystem right now. There are literally tens of thousands of developers who are building libraries for Angular.

And Angular is pretty stable. As I mentioned, it is used by thousands of projects within Google. And every single change we make in Angular, we're testing against all these thousands of projects. If we break any one of them, this means that the change was backwards incompatible. And we either rollback or we work on making it compatible, if that's what we want.

So stability, opinionation and ecosystem, these are some of the main benefits. And from there, there are others as well clearly. At Google – So we are working with a lot of web platform teams. So we're making sure that Angular is always on top of the latest web standards. For example, we adopted Trusted Types. And we were pretty much the first framework that adopted Trusted Types, which is a spec that allows to take advantage of a web platform feature that prevents cross-site scripting attacks. And the investment in backward compatibility also allows us to implement – Requires us to implement the ng update experience. So it's a single command where once we release a new major version of Angular, we are encouraging everyone to run the ng update command so that they can get up to date with the latest features, bug fixes, and security fixes in Angular.

[00:13:45] KP: Well, when it comes to static typing the, I guess, number one reason I hear people give for why you want to choose something statically typed is because a lot of errors will then be found at compile time rather than at runtime. And of course, you want to get those errors sooner. Is there anything else that I'm missing? Are there reasons why static typing makes Angular a better choice for a lot of groups?

[00:14:07] MG: Yeah, static typing definitely allows us to catch a lot of issues. Something that we often don't appreciate enough is the development experience improvements that static

typing is bringing. For example, auto completion is great with TypeScript. We know all the different fields and their types, and all the different objects that we're accessing. So this is definitely a great benefit. And many of the issues we were catching, even as part of our development process, we don't have to run computation so that we can catch potential issues.

Static typing also allows us to build more advanced tooling based on the type information that we have. For example, in Angular, the ng update experience is allowing us to make very advanced refactorings only because we understand the whole type information within the project. And type systems are also very, very convenient for automatic generation of documentation or reasoning about the source code. And clearly, also refactoring. We can rename a field. And based on the type information, this change can propagate across the entire project. So there are quite a few benefits of static typing.

And as you as you said, catching issues ahead of time is one of the most tempting benefits. There was a research by Microsoft, I believe, and a couple of folks in the academia where they were comparing how would open source projects that were implemented in JavaScript would have benefited if they had a type system, if they were implemented in, let's say, Flow, or TypeScript? And I think they discovered that at least 10% to 15% of the issues in their issue tracker would have been caught by the type system ahead of time. So that's a pretty significant benefit.

[00:16:03] KP: Maybe the motivation to create TypeScript even. I don't know.

[00:16:08] MG: Yeah.

[00:16:09] KP: Well, I think most listeners will certainly be familiar with Angular, even if they haven't had the opportunity to go use it themselves yet. I'm not quite as confident everyone will know what guest.js is. So I wanted to ask you, what is guest.js?

[00:16:24] MG: Yeah, guest.js is a project that I – It's pretty much a conference-driven development that caused its implementation. Back in, I think, maybe 2017, I wanted to speak on a conference at Oxford Render. And I was thinking, "What cool think can I build for this conference? Can I start prefetching crowds in the application by using machine learning? It's

like, “Maybe I can. Let's figure out how to do that.” And I applied for the conference. My talk ended up being accepted. So I had to figure it out.

I remember I was here, in the Bay Area, when like hanging out on a dinner with friends and with a colleague. Back then, actually, he was not a colleague of mine. He was just a person I really admired in the open source community, **[inaudible 00:17:17]**. I chatted with him and I mentioned that that's something that I'm planning to build. He said, “Well, yeah, I was thinking about this too. And I have been doing some high-level exploration.”

So a couple of weeks after that, after spending like hours and just writing things in my notebook, I figured out how we can use Google Analytics in order to build some kind of a data analytics model that we can later on use in order to prefetch routes in an application and make it in a way that is pretty much developer ergonomic so that people can add one Webpack plugin to their Webpack config, specify their Google Analytics View ID. And from there, everything will happen automatically for them.

And yeah, I wrote a blog post explaining code of theory behind that. It was a lot of fun to figure this problem out and make it work with popular frameworks such as Angular. Back then, the original framework we support with guest.js was actually Gatsby, which is a React-based static site generator. And we also introduced support for Next.js and Nuxt, which are respectively React frameworks.

Later on, Addy ended up sharing this project as part of his web talk on Google IO 2018, which was I felt pretty excited. That was before I joined Google. So listening his talk online, hearing him talk about guest.js was like a dream come true back then for me.

[00:18:53] KP: Very cool. Well, it sounds very easy to set up for a developer. I'm wondering if we can explore what's going on under the hood. What is being guessed?

[00:19:02] MG: Yeah. So depends on the implementation right now. Since I joined Google, we explored this topic even further with the TensorFlow.js team. And we're doing more accurate predictions. I can talk about both implementations. I can start with the original implementation of guest.js. And after that, we can move on to the more advanced one.

So what happens internally is when someone runs engine build, let's say, in the context of Angular or npm run build, we're, first, with OAuth, requesting access to their Google Analytics data based on the View ID that they have specified. Once we fetch this data, we're going to have topples. In these topples, we're going to capture how many people went from page A to page B. And based on this information, we can build a very basic prediction model, very basic predictive model which calculates the probability for one person to go from page A to page B, rather than to page C.

Once we have this information, we can embed it into the main application bundle and later on introduce a small runtime that either takes that the user is navigating from one page to another, or they have navigated from one page to another already. We can look at which are the most likely to be visited next neighbors and we can prefetch the corresponding JavaScript bundles.

That's pretty much how it works on a high level. There are quite a few challenges in the meantime how we can make sure that the model that we're building is good enough so it doesn't add a high-performance penalty to the user's application. And the second challenge is how we can map routes that we're getting from Google Analytics to actual JavaScript bundles, because this is not a trivial problem to solve.

[00:20:57] KP: Yeah, both are interesting. If we could unpack the routes and bundles challenge first.

[00:21:03] MG: Sure, yeah. So it's frameworks. We usually have routing. And the declarations of the routes within the application, they usually map pretty much one to one with the routes that we have in Google Analytics. For example, if you have, let's say, page/a, we're going to have a route called /a that points out to a particular JavaScript file. If we have a route called, let's say, :id, this is probably going to be a placeholder. So there is going to be a route parameter here that could map to a variety of different pages in Google Analytics. But this is still something deterministic. We can still figure out to which routes this route declaration eventually maps to.

Now, once we have the route and the corresponding JavaScript entry points that needs to be loaded for this particular route, we can plug into the build process and figure out what is the

name of the bundle, JavaScript bundle that is associated with this entry point. And we can transform the graph that we're getting from Google Analytics or just probability probabilistic model, let's say, from which corresponds to which pages are likely to be needed from a given page, to which bundles are likely to be needed after a given bundle is already loaded. That's pretty much how it works on a high level. It pretty much involves some static code analysis. And since JavaScript routers, they could be pretty dynamic. We can have the declaration of the path. It could be a concatenation of strings, let's say. The path could be the string/ plus the string concatenated with the string A. This also involves some partial evaluation. So as part of the build process, we're going to look at the source code and to try to evaluate it to something, to a static value, to a literal. I guess that's on a high level. I might be getting into a little bit like too many details. The implementation is also available on GitHub.

[00:23:15] KP: Gotcha, gotcha. Yeah. Well, machine learning can take a variety of different forms. But I think the stereotypical one people think of is you get some raw data. Maybe you do some feature engineering. Now you have a training set, and hopefully some objective function you want to predict. Is that what's going on here? Or is it a different flavor of machine learning?

[00:23:36] MG: In guest.js we're using a very simple Markov chain model. So not necessarily – Nothing really too advanced. It is a very simple matrix, which predicts pretty much the values in the matrix is what is the probability for the user to go from route A to route B? That's it. With TensorFlow, we explored deep neural network that we built based on the same information. How many users went from – Actually, there, we consider those who user identifiers so that we can personalize the predictions for a particular user. There, we use the more advanced pipeline where we're taking the data from Google Analytics, piping it to MapReduce where we're bouncing is to TensorFlow extended, building a TensorFlow model where the different features or different variables, let's say, or like features in the model or the user identifier and the navigations that they have performed. And this TensorFlow.js model, TensorFlow model, we are wrapping it into TensorFlow.js S and running it into the browser.

[00:24:48] KP: Very cool. Do you see any major performance differences between the Markov approach and the TensorFlow approach?

[00:24:55] MG: Yeah. With the Markov chain, the payload of the model is way smaller. And also, at runtime, we don't really have to invoke a complicated neural network, which needs to perform some GPU calculations. We can just perform a very simple constant lookup. That's it. So I'll say that in the majority of use cases, when the very high level, the very high accuracy is not critical. People can just go with Markov chain.

Another great advantage of the Markov chain is that, for every bundle, we can just have the corresponding row of the matrix, of the matrix embedded inside of it. So we don't have to ship the whole model as part of the application. We can ship parts of the model in different parts of the application. And this makes it even more efficient. That's how guest.js functions currently.

[00:25:46] KP: I guess what is that model? I know there's like the Onyx format that we might use if we're going to do some deep learning. TensorFlow has its own formats. How are you managing the Markov model in JavaScript?

[00:25:58] MG: This is just one vector currently. It is a matrix. But since we can – Actually, it's a map, when I think about it. It is a matrix. But since we're taking different rows of this matrix and embedding them into separate bundles, we're just keeping all the neighbors of the particular bundle. And from there, we are keeping also the weights or the probabilities for the user to go to another bundle. We are removing irrelevant values. And we're also doing some processing of the probabilities themselves to keep them short, let's say. Reducing the precision a little bit, because we don't want to have decimals with like a couple of thousands of digits for performance reasons.

[00:26:42] KP: Interesting. Well, am I correct in saying the goal is really to empower the prefetching decision?

[00:26:50] MG: Mm-hmm.

[00:26:51] KP: And then what gets the – The next page gets prefetched, I guess. Is that all managed through the Webpack integration? Or do I need to do anything as a developer to ensure that that goes smooth?

[00:27:01] MG: It is all done through the Webpack integration, yeah. And it also follows best practices. We worked with the Chrome team in order to figure out what is the most optimal way to prefetch without causing any frame drops. So we're writing the predictions, even though the predictions are literally just a for loop over all the neighbors of the current bundle and removing all neighbors that are below certain probability thresholds, which probability thresholds depends on the user's network speed. Even though the computation is pretty simple, we're still running it in a request idle callback, which is the callback that the browser executes when it doesn't really have anything important to do so that we don't drop frames. And we're using also a low priority prefetching with link rel prefetch that the browser can prioritize whenever there are no critical resources that it needs to download.

[00:28:03] KP: And are there any metrics of success that you look at? Like obviously, if you prefetch something, and then I click on that, that's a win. But if you prefetch something, I click on something else, is that a loss?

[00:28:15] MG: Yeah, we're not capturing this information so that we can keep the JavaScripts and the model as minimal as possible. We have done the couple of – It very much depends on how often people redeploy the model and whether they're making significant changes in the structure of their applications. We have done a test with a Pakistani job website. It has hundreds of thousands of users who are visiting the website every day. And with the pre-aggressive prefetching, we reached about 90% accuracy. So that was a decent win. And also it was way more efficient compared to alternative prefetching strategies where we're prefetching all the JavaScript bundles associated with visible links on the page, let's say. But still, we can over-fetch, for sure.

[00:29:02] KP: Makes sense. And is that something a developer might want to fine tune? Or do they just trust that Guest is doing the best job it can?

[00:29:10] MG: Yeah, developers can override the weight. So guest.js tries to be adaptive, to adapt based on the network speed of the user. So if the user is on an LTE network, then guest.js need to be a little bit more aggressive. And there is one probability threshold. For a slow 3G network, let's say, there is another probability threshold. And for 2G network, there is a third one. So there are some default values, but developers can override them if they need to.

[00:29:43] KP: If I do some major changes to the routing structure of my site, or maybe just introduce some new routes that are especially popular, what happens then? Obviously, there needs to be a time of adjustment. If the user's behavior changes, how does guest.js respond?

[00:29:59] MG: This pretty much depends on the information from Google Analytics. At first, when there are not enough users who have visited the new routes, the information might not be extremely accurate. Over time, once you have more data for the user navigation for a pattern, so the user is navigating across the application, then the accuracy will improve. So, yeah, it all depends on for how long these routes have been up and how many people have interacted with them.

[00:30:28] KP: And in deciding between if they want to take the Markov model approach or use the TensorFlow model, which as you point out, has the added benefit. It could be coming up with predictions that are specific to a particular user, if that's appropriate. How should a team decide which way to go?

[00:30:47] MG: The Markov-based approach, currently, it is very magical, I'd say, and his works with Webpack version four. So haven't updated to version five just yet. That's one of the differences. And also the Markov model approach, it only prefetches JavaScript. What we implemented with TensorFlow.js was a service worker prefetching approach. Because the TensorFlow model, it could be – Although it is really well optimized, it could be heavier running it in the main thread. So we're running it in a service worker. And the service worker is prefetching predefined set of resources in its service worker cache. And also the machine learning, TensorFlow pipeline is way more sophisticated with way less automation. So I'll say that I'll recommend this approach for developers who were more advanced. They're sure what they're doing and they have high-level of confidence that accurate predictive prefetching in their website is going to be beneficial for their business.

[00:31:56] KP: So I have always thought that Google Analytics was a somewhat underutilized resource for projects exactly like this one, that it's this wealth of data. It has an API. You can query against it. Maybe do some interesting real time stuff. Are you aware of other projects? Maybe something that inspired guests? Or alternatively, do you have any ideas for how

companies could be better making use of this what I'm asserting to be an underutilized resource?

[00:32:24] MG: I think there was some prior work before guest.js. Addy, I just mentioned, he captured some of the prior investigation in this direction in the readme of the project. That's everything that we found. I think some folks were using a little bit more manual approach in order to predict the prefetching order and which resources are more critical for prefetching.

Right now, what I'm thinking is maybe for some recommender systems, this is pretty much a recommender system. We're recommending which bundles might be needed in the future. So we're prefetching them. For a little bit more product features, I think the same approach could be pretty valuable as well.

Yeah, generally, the direction that I'm thinking about as in terms of recommender systems. I'm sure there is probably way more. But probably that's just a space I'd been focused on over the past couple of years actually.

[00:33:21] KP: Yeah, promising route for sure. Do you consider guest.js to be complete? Or is there ongoing development?

[00:33:31] MG: Yeah, it's pretty complete, I'd say. It works better with some technologies than others. And currently, it does not support Webpack version five. So I have opened an issue to gouge the interest. If people really want us to support the latest version of Webpack, then this is something that we can do. Aside from that, definitely, there is always an opportunity to add new features. But I think it is in a pretty good state when everything more than this is going to be potentially a scope creep. So it is in a good place right now. Works particularly well with frameworks where the routing structure is statically analyzable. In particular, Angular, and Next.js right now.

[00:34:15] KP: And is there any reason why I might not want to install guest.js. Could my site be structured in a weird way where it would have little or no benefit?

[00:34:27] MG: Yeah. Actually, guest.js is not going to be the first approach without recommend to people exactly because even though it has developer friendly experience, the lack of Webpack version five support right now and also the fact that people would have to necessarily have static analyzable routing structures sometimes, it is just a constraint that people can't afford. So a couple of alternative approaches for prefetching that I'll recommend are prefetching **[inaudible 00:34:59]** over a particular resource, or quick link prefetching, which is prefetching of the resources associated with links that are visible on the page. These are very easy to set up. They work pretty well. Well, at least with Angular, this has been my focus. But I think alternative JavaScript frameworks can have their implementations as well. So yeah, that's what I'll go with first.

[00:35:27] KP: And do you have any opinions on whether or not features like that the, hover prefetch? For example, is there a reason someone would say that should just be baked into Angular? Or are there fine lines here between framework and library that you can highlight?

[00:35:42] MG: We've been having similar discussions with a team about whether this should be baked into Angular. Yeah, maybe it should be. It is, for sure, baked into some frameworks. When I was collaborating with Kyle Matthews from Gatsby, I think this approach was baked into Gatsby. In Angular, we had an alternative pre-loading strategy that there we call them pre-loading, which was Preload All that we should eventually deprecate because it is pretty on the CPU sometimes with a lot of fraps. I will say that we are not opinionated about this just yet. We can consider being opinionated about it. But this also adds extra bite to the production bundle. And we're not sure absolutely everyone will be benefiting from it.

[00:36:27] KP: That's a good argument there, yeah. Well, are there any things on the Angular side, maybe recent releases or stuff coming up on the roadmap that you're excited about and able to speak to?

[00:36:37] MG: Yeah, there are quite a few things that are happening right now. So the biggest release over the past year that we did was Angular Dev Tools. It's currently has over 105,000 people using it, I believe. We are working towards optional engine modules, or standalone components. This is a significant change or a simplification of the mental model. People will not

have to declare Angular modules. They can just use the very basic component model and declare the dependencies of their components within their metadata.

And this, by its own, unblocks a couple of other proposals that we are exploring. One of them is template composition API. So people can dynamically assemble their templates at runtime if they need to. Another one is out of band type checking. This is actually something that we have explored in the past. Actually, this is not necessarily related to standalone components. What I meant to say is more localized component compilation.

Now, when developers are specifying the dependencies of their components directly in the metadata, then we have more explicit dependency graph, which would allow us to make the build process faster. So let's say around standalone components, I'm pretty excited about how it will reduce the learning curve. And from there, what benefit is going to bring to the developer ergonomics and to the build speed?

[00:38:11] KP: Makes sense. Minko, where can people keep up with you online?

[00:38:17] MG: I am most active on Twitter and LinkedIn. So you can find me on both places. My username there is mgechev, the first letter of my first name and my last name. And I recently started a newsletter, where I'm pretty much – So I've been sharing tips and tricks about JavaScript, Angular and development to encode the past couple of years now. And if you're following these channels on social media, definitely you're going to get exposed to all these tips and tricks. And the in the newsletter, some people just prefer to consume this content in an email format. So that's an alternative place you can find it on my Twitter profile as well.

[00:38:58] KP: Well, I recommend people take a look at that and sign up. Minko, thank you so much for coming on Software Engineering Daily.

[00:39:04] MG: Yeah, thank you very much for having me. It was a lot of fun chatting about Angular and guest.js.

[END]