EPISODE 1092

[INTRODUCTION]

[00:00:00] JM: A software company manages and interacts with hundreds of APIs. These APIs require testing, performance analysis, authorization management and release management. In a word, APIs require collaboration.

Postman is a system for API collaboration. It allows users to test APIs with collections of requests and monitor the API responses and visualize the query results. Users of Postman can collaborate with their team through team workspaces, sharing collections, environments, history and more. Abhinav Asthana is the founder of Postman, and he joins the show to talk about API collaboration. Postman was started as a side project, as a hobby project. It's grown into a large and successful business far beyond the original product of API testing.

If you would like to advertise on Software Engineering Daily, you can send me an email, jeff@softwareengineeringdaily.com. You can reach more than 30,000 engineers every day, and we'd love to have you as a sponsor. If you are a listener who wants to become a paid subscriber to this show, you can go to softwaredaily.com and click subscribe. On Software Daily, you can also find information about different topics. You can find episodes that relate to one another. And we'd love to have you as s subscriber. Thanks for listening.

[SPONSOR MESSAGE]

[00:01:29] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team.

These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW]

[00:03:18] JM: Abhinav Asthana, welcome to the show.

[00:03:20] AA: Thank you for having me, Jeff. Excited to be here.

[00:03:23] JM: You work on Postman, which is a platform for API collaboration. What is API collaboration?

[00:03:30] AA: Postman started off as an API REST client. Today, we service the whole spectrum of development, testing, operations and monitoring for APIs. What we kind of learned was that all of these things require close collaboration between different teams when they're using all these tools that are there on the Postman platform.

What effectively Postman's collaboration platform let's all these teams do is come together on a single platform, work together, share and get feedback on things quickly and effectively and just help you build high quality APIs faster.

[00:04:11] JM: I remember using Postman in college. You started with that original just API client use case. At what point did it become something that was bigger that actually resembled a business?

[00:04:25] AA: That's a good question. Postman for me was kind of like the side project for a while and I built it to solve my own problems working on APIs. What I learned was that the problems I had was basically the entire developer community shared, and those were the things that people were dealing with. We actually had the first version on the Chrome Web Store back in 2012. It got featured on the new version of the Chrome Web Store in 2013. We had about half a million users back then. There was just so much incoming feedback and usage on the product and so much more that people wanted us to do that we decided to start a company. At that time, we were fortunate to get our investors to write a check for us. That was where we kind of starting thinking that this is a very big company that we want to make.

[00:05:15] JM: Can you describe the usage of Postman today in more detail? What is the average developer use it for?

[00:05:22] AA: There are a bunch of use cases. The first one that typically most developers would start with is what I call like this debugging loop for APIs. You are a developer, you are writing some code. You want to make sure that the API works as expected, and you would file that API call in Postman, ensure it's working. If it's not working, you go and fix something and then go back to using Postman again. That's the very core use case that sticks to developers on their day-to-day workflow.

Beyond that, what developers then start doing with Postman is documenting these API calls, creating richer documentation to be shared with others, writing test suites. They make sure that they don't have to manually look for bugs anymore. They can write automated test suites in Postman. And carrying on from there, they can integrate Postman into their continuous integration workflows and in their build pipelines. Eventually, all the way to when they ship an API to production, they can use our monitoring service to make sure that those APIs are working as intended. For a developer, we take them through this whole cycle of thinking about an API, testing their first API call, all the way to using it on a regular basis to make sure all those APIs are working.

[00:06:39] JM: If I'm doing API testing, what are the different aspects that need to be tested? I mean, there's so much that goes into an API. You've got performance and consistency and you want to test for the right inputs. Tell me about the workflow for doing API testing.

[00:06:57] AA: Some of the first things that people like to do is make sure that the API data model is correct, like am I sending the right parameters? Am I getting the right response? The way we look at this whole process is kind of like defining the right boundary for your software. You're writing all of these code and the API is effectively like a boundary through which data is coming in and out.

Where developers start with is specking out that what would happen if I send a [inaudible 00:07:28] parameter? What if I have a use for which I need to get a certain response? Is that response correct? You're going to start with the very basics there. You want to look for what are the right status codes for the API. Sometimes developers would start, if they're building a REST API, just with a default that their framework gives them. But you want to make sure that, let's say, if you don't find the right response on your server, you want to return a 404. Or if something was not working, you kind of return something in the 3xx category. There's kind of like this evolutionary loop for your API that kind of happens. With APIs, generally you're not testing just one API endpoint. You're testing – Sometimes your API in conjunction with other APIs. You want to see the flow of those APIs. Is that working or not? Or you might want to run a regression test to make sure if you make a change in one API, then is that change not breaking all these other endpoints that you have inside your API?

From that point onwards, if you want to ensure that your API is working as per defined behavior, you can go towards checking for security parameters. Making sure that if you, let's say, inject something or you try to kind of break the API into spitting out some data that you didn't intend to, you would want to check whether the API is secure. That way, Postman also returns request file data. How fast is your API at a per API call level. Is it taking too long? Where is exactly the flaws at in terms of your infrastructure when you're looking at your [inaudible 00:09:01] performance perspective. All of these things kind of going there. Some of the more complicated are really testing for all the different ways in which an API can go wrong, and you can embed those test cases in Postman.

The testing landscape, like as we have started is like there are lots of different terms associated with it, like integration testing, contract testing, functional testing. And I'm kind of giving like a high-level summary of kind of all of those. But you can pick one of these as your preferred approaches towards testing and then kind of go for that.

[00:09:37] JM: So if I'm a developer and I'm messing around with APIs, I could use curl, I could use wget. Why would I use a platform for doing API testing?

[00:09:52] AA: That's what where like I'd say the worst state was when I started building Postman, and I was one of those developers who would have to rely on curl.

One basic problem that you end up with is as your APIs are getting more complicated, working on a command line with curl or wget becomes very hard. Something as simple as sending a simple parameter, if you make a change in one character, your API will not return the right response. All of those things kind of become very hard. It's okay sometimes for simple APIs if you're just doing just the basic get call to fetch some data. That's fine. But as you go towards building a more complicated API request with form data parameters, files, and you also have large JSON responses to process. You end up typically writing a homegrown API client yourself.

When I actually was building Postman, I used to go to developers and ask them, "Hey, how do you solve this problem? Everybody would have their own API client that they've built on the side to help then with API projects. What we learned was that we could just solve all of these problems in one go. We help you construct those requests better. You see the responses in a pretty way. You don't see them as a single line JSON test. Once you see them in a legible way, you can just do that faster, the debugging process and the testing process faster.

Now, the interesting part about APIs is that – And the way we defined it is that an API is an intent to collaborate with somebody else. If you're writing code, it's something that you might use it for yourself. But if you're building an API, somebody else has to use it. Otherwise, it's going to be a pretty sad API, like I built an API just for myself.

Sometimes those relationships extend to backend and frontend developers. It extends between backend and backend developers when you look at microservices architectures within a company. It extends to development teams across different business units, different development teams working across company boundaries, like partners, or if you look at public APIs where you or an individual developer, relying on public APIs from the likes of like Stripe, or AWS, or Google.

When we kind of looked at this problem of API testing, we saw that it's not just about making sure whether the API is working. It's whether these people and these teams who are working with each other through this connecting pipes of APIs, are they working better or not? What we started building towards was helping developers and teams and manage the complexity that comes with it.

The next step that happens is, "Okay, you tested the API is working." But what if an API changes because somebody else changed it? Now, they might not know which parameter that you were relying on or what was the behavior that you relied on. There's an interesting law called the Hyrum's Law here that all observable behaviors of a system will be relied on by consumers of an API. Even API producers typically don't know how exactly consumers are using. This becomes a collaborative problem. Somebody changes an API, somebody else relies on it, somebody builds other APIs on top of it. The reason you use a collaboration platform is just all like this meta and a much more major problem of making sure that you're working fast enough. You can do that. You can send curl calls over email or Slack. But as the problem scales up across hundreds of thousands of APIs, you would need a collaborative system to help you do that.

[00:13:21] JM: You've built a variety of other tools around this problem, because API collaboration wasn't even a space when you started Postman. You basically identified that this is actually a really big opportunity. There's one product you built called a Postman workspace. Can you explain what a workspace is?

[00:13:43] AA: Yeah, absolutely. Workspace is an interesting concept as we developed it as Postman evolved. One of the things we saw was to solve this collaboration problem, people are essentially sharing files all over the place. Even Postman collection files, they were downloaded

and shared to others. They would sometimes put it inside a repository and other people would take it from the repository. There was a lot of like this back and forth that is happening through tooling.

What we envisioned was a shared environment in which different people can come in. They see the same API as everybody else. Postman workspace in at a real-time collaborative construct. So let's say you and I are working together on Postman and you changed something. Then I'm going to get that update with an automatic DIF of what was there.

If you imagine what Google Docs is to documents. A Postman workspace is the same thing for your APIs. And you can invite as many people as you can. You can do your testing. You can do your monitoring. You can do your debugging and documentation. All of that together. It just [inaudible 00:14:49] on a lot on the amount of time it just takes to share things along. That's what Postman workspaces help you do.

[00:14:56] JM: Workflow, by the way. So let's say we got a company with two developers in it and I'm developing a service that's downstream of your service. You've got, say, service A and your service A is often calling service B, which I maintain. We've got this interaction pattern between each other. Whenever I deploy a new version of my API, you need to be able to hit that API still. What is the workflow like?

[00:15:26] AA: That's a good question. We are a little bit non-open ended about which workflow you would like to adapt, because there are certain social constructs between people. But in this case, one thing that we would recommend is – Before I go there. The notion of a workspace in Postman is that they are contexts. They are not containers. The differences that you can have a workspace for yourself and somebody else can have a workspace with themselves, and the same thing can be shared between the two. Because I'm seeing something in one context, and maybe you're seeing something in the development context and I'm seeing something in the consumption context.

What here we would do is you would clear the Postman collection and we can probably talk about Postman's design tools if it comes too. But not we are flexible with respect to what we support, an open API file or whatever. I would, let's say, clear the collection on some

documentation. I would share it in your workspace. If I make changes to my API, update something, you will get the same change.

Alternative is we can clear the shared workspace and I can invite those developers and they can build something together. If I change something, I will update that collection. That collection is going to be synced automatically to somebody else's Postman instance, and now they get an updated version of the API. There are two models available. And as the development team kind of scales up and they have different contexts of going from development, staging, production, you have all that flexibility there.

[SPONSOR MESSAGE]

[00:17:04] JM: Today's sponsor is Datadog, a monitoring and analytics platform for cloud scale Infrastructure and applications. Datadog integrates with more than 400 technologies so you can track every layer of your complex microservices architecture all in one place. Distributed tracing and APM provide end-to-end visibility into requests wherever they go; across hosts, containers and service boundaries. With rich dashboards, algorithmic alerts and collaboration tools, Datadog provides your team with the tools that they need to quickly troubleshoot and optimize modern application. You can see it for yourself and start a 14-day free trial, and Datadog will send you a free t-shirt. Just go to softwareengineeringdaily.com/datadog and learn more. Get that free cozy t-shirt and start your 14-day trial of Datadog. Thank you to Datadog for being a sponsor of the show.

[INTERVIEW CONTINUED]

[00:18:03] JM: Can you tell me more about how companies actually use Postman? Because I know how I've used it as just a rogue random developer in College, but I'd like to know more about how like actual enterprises use it.

[00:18:17] AA: Postman is being used at large enterprises across hundreds of developers working together for all the cases from design to production. Typically, for larger organizations, they are structured into individual teams. Those teams will be part of the same Postman

account. These teams will have the workspaces that they work in along with other teams having their respective workspaces.

Once the entire organization is on Postman, what they end up doing is kind of having a system that becomes like a source of truth for their APIs in Postman. Okay. Everybody in those teams, is our source of truth up-to-date? Then they share things across workspaces for all the aspects of your API cycle. This would be designing your APIs through open API files, GraphQL files, RAML files. Of course, the debugging use case is something that's a day-to-day part of every developer just like yours, the use it for API consumption and API debugging, building and sharing documentation for those APIs.

A key part is integration testing and test automation. A lot of those teams have developers working alongside ops, alongside QA, and the QA engineering team would build test automation suites in Postman using the APIs that the development team has published for them. Then they would be integrated as part of the CI pipeline that the ops team is running. In this construct, the same things that you were using to share an API hacked as quality gets in your pipeline. Then the ops team takes those things, builds the pipeline. Once the APIs go to production, they use the monitoring service to make sure they are able to hit those APIs or they also use it for just checking once in a while that things in production are working well. That's the production workflow.

We also see use cases across non-developers who are involved in what we call the business of APIs. This involves let's say driving adaption of public APIs through our documentation tools that you can publish, building recipes or onboarding journeys for your API buyers in a way. All those things happen for large enterprises within a Postman account.

[00:20:37] JM: Tell me about the stack of technologies around Postman. I'm imagining myself issuing an API request in Postman and I'm wondering what's going on on your stack.

[00:20:52] AA: Postman uses NodeJS for almost all of the client and the backend services that we have. Postman is written in JavaScript and NodeJS. Today, we published the application on Electron. We actually recently announced that we'll be having a web version of the app. It's

called Project Artemis for us. It's going to be launching soon. But a JavaScript background kind of helps us to that.

Now, when you send a request in Postman, the interface calls our Postman runtime. Postman runtime is an open source project under Apache that basically executes the request. It executes those test scripts and renders the response back for the Postman interface to process. Those are things that happen on the client side of things. We of course have other tools that complement around time. All of them, again, are on JavaScript. This is the client side version of things.

I forget, like the Postman runtime also has an open source version called Newman that you can run in your command line that's published through npm. That's the stack of tools that you use to run requests. Every time you send a request, the runtime run those things. Tells you whether things are working or not. Process the response back and puts it in the Postman interface for you to parse.

Now, what you also have is the cloud aspects of the collaborative platform. That's what – We are on AWS, and we have about I'd say 50 microservices that power that experience and the tools that we offer on the cloud, monitoring tools, documentation tools, mock servers and things like that. Again, we are very heavily based on NodeJS, and there is a whole stack of like tools that we use within AWS to power that.

[00:22:33] JM: Tell me more about that. Like on your backend, what's the AWS Infrastructure look like and what have been some interesting architectural decisions you've had to make?

[00:22:42] AA: That was actually a very interesting journey for us. One of the things we learned – And we thought we knew about APIs pretty well because we have kind of breathing that space for a while. So the first architectural model looked like actually have centered around the three founders of Postman. The code that I wrote was basically like Abhinav's have the code [inaudible 00:23:05]. Ankit's have – Kind of like the way we divided it was basically how it evolved. My third cofounder, [inaudible 00:23:13], who managed the sharing and the collaboration aspects in the early days along with like [inaudible 00:23:18]. Those were like the three hubs, and they kind of worked like the largest pieces of code that we wrote.

As we started evolving more, we understood that we had to of course add more engineers, kind of segment that work. Our first split was around backend and frontend. There was like a backend team and a frontend. A client team and a backend team. What we started realizing was that that doesn't work very well. For software delivery, we'd have like a lot of different issues between teams when it came to shipping.

Today, we are actually organized around this notion of squads. Each squad owns a particular domain of the Postman product, and we follow like these domain-driven design practices. A particular domain in Postman would be documentation. Anything to do with documentation including the client apps, the backend apps, the data stack. All of that would be owned by that particular squad, and that squad then essentially communicates through APIs with other squads. That's where we have been like a little – Like we are evolving and we're not at that AWS level of strict separation yet. But most inter-squad communication happens through APIs, internal APIs, and squads publish this API. They're going to use Postman for that. And then other squads can use that functionality easily.

A related part of our stack is client application. So we have more deals like rendering an editor which are available for the entire company to use when they're building interfaces. We have a design system that helps us kind of manage that. Related to these squads which are intended for product delivery, we have a platform team that looks at infrastructure security and operations. And these teams interface with squads at the platform layer. They are helping them manage like AWS better. Helping them manage security and quality independently. We have like the centralized teams and then these distributed teams around different squads, which communicate to each through APIs.

[00:25:20] JM: Talking more about how Postman fits into the workflow, can you just describe, like as I am working on my API, where are the different places where I am using Postman? Am I using it to test the API as I'm developing it? Am I using it during the – Through some more of the use cases for actually using Postman.

[00:25:51] AA: I'll talk about like the ideal workflow that you should have if you are using Postman for that. And we have this whole notion of being API first if you were to start from the

very beginning. What it means is let's start from the fact that you just had an idea about an API. You would open up Postman. You can write your specification file in Postman and you have kind of just sculpted like the first few [inaudible 00:26:16]. What you'd be able to do in Postman at this point in time is create a mock server or a prototype of that API before you write any code. At this point in time, you have a specification file. You have some documentation and you have a prototype. Those things, you would then translate into code, which would be outside of Postman. You would go to your editor, you would have your repository. You would translate that prototype on to a framework. And that's where you get a living, breathing API. The cycle of writing code intersects at different points in time with the API cycle.

You've written your code there and now you want to test it whether it's working as intended. You come back to Postman. Use the Postman client. Fire up the request response and see if it's working as intended. Now, within Postman, you can update the specification. You can update its prototype. You can update its documentation and then go back to kind of code again. Or you might stick to code and flesh-out a few things there and then come back to Postman and update these things.

That's like the first part of the development cycle. The second part of the cycle is when your API is a little bit more finished. Typically, we see two people or two kinds of people who might go from a single developer to a different developer. It might be a frontend developer who has to build a client application to integrate the API into that application. Let's say if you're building a smartphone app, that's what – You might take the real API along with the prototype and you start using Postman to test out like what you would be using the API for. Does that work as intended? Then the frontend developer would [inaudible 00:27:57] that API in their application. Again, they would have like their coding tools to do that. Postman kinds of acts as the glue here to connect the backend developer and the frontend developer.

Another persona that comes in is the quality engineer. Now, they would start testing out the API very early in the cycle. They know what's coming up. They will start thinking about ways in which they would want to probably [inaudible 00:28:18] the API. They would write test suites in Postman. All of these would happen in Postman where it's a JavaScript code that you can write. In the early phases, we still see a lot of manual testing work, because the API is not really finished.

Now, we just go with these three units, backend developer writing the code, frontend developer consuming the API for eventual integration and a quality engineer testing out the API for an eventual test automation speed. Once you go past the testing phase, the backend developer would check in into a master branch that, “Hey, this API is done.” The tester will check in the test suite and the frontend developer would check in their application. Then that whole API testing part of it would be run on Postman’s tools.

Kind of like the way at least we see it as that this is a unidirectional flow, but the fundamental truth about APIs is that every time you make a change, you have to go through this cycle again. That change in the cycle can come from any place. The tester might discover that, “Hey, this API is not working for X use case.” So they might communicate it back to their developer that is going to update the API. Then frontend developer has to go and kind of make changes there. Every time there’s a change in the cycle, this dev test and deploy cycle is what you use Postman at different points. Hopefully that gives the picture of the workflow.

[00:29:36] JM: Definitely. Now, Postman is built in Electron, and at least the user-facing client, the desktop client. And I’m wondering if there are any interesting challenges around building in Electron, which for those who don’t know is the platform that’s kind of like a browser or a wrapper, uses Chromium to build desktop applications.

[00:30:02] AA: Electron has been actually – The reason we actually chose Electron was that to kind of go back a little bit, Postman was a Chrome App, and Chrome Apps were basically attached to the Chrome browser with elevated permissions to help you send API calls – I mean, from our perspective, behind a firewall.

Postman’s core technology was Chrome Apps, and [inaudible 00:30:24] Chrome Apps was that you didn’t have to ship the entire Chromium runtime separately. They would just be attached to the Chrome browser.

What happened for us was actually Google deprecated the Chrome App platform actually twice, and then we had to figure out a way to service all of our users across all three desktop

platforms, and we looked at what was our out there, and Electron basically was the best choice, which gave us like the widest reach among developers.

Now the challenge with Electron is that, of course, you have to build a performant app through that. The other challenge with anything, like a technology like this is that you have to do a lot of work at the API level when you're integrating the desktop application with a native OS experience. The APIs need to work well and there are certain limitations that you typically run into. That's one kind of challenge. The second challenge typically is around how often you can update these applications and give like a newer experience to people. Typically, electron applications can take. If you're running a lot of them together, they can be taking up a lot of memory at times. So we kind of worked a lot on just optimizing for performance, and that's been like a continuous journey for us.

Kind of looking at VS Code, which is probably like the most popular editor in the world today, and they've kind of shown a lot of ways in which how they can be very performant. Those are the typical challenges that we see. Yeah, we've been just trying to meet the expectations for people across all [inaudible 00:31:58]. The advantage is that development teams and companies that are using Postman don't have to be bound by one operating system. But yeah, there are things that we have to do to make sure people get a great experience.

[00:32:10] JM: Tell me more about the testing and how Postman fits into an automated testing or unit testing procedure.

[00:32:22] AA: Typically, the thing that maybe some listeners would be familiar with is like the testing pyramid. So, UI testing, integration testing and end-to-end testing, right? Unit testing is something that you do at a code level to make sure the function that you wrote is working well. Integration testing is typically where you kind of go towards APIs and that's where you look at whether the combination of APIs and the individual APIs that you're working is working well.

There is some confusion in the community always, like whether an API end point testing is unit testing or not. So we treat them the same way. Then there is end-to-end testing, which is basically running your application through Selenium or Cypress or any other end-to-end automation testing tool.

Postman fits in right in the middle with a little bit of that overlap on the unit testing side if you choose to kind of go that way. So you would test all your APIs together in Postman, or you would test them individually, and there are like the contract testing and functional testing, a lot of techniques around this. But yeah, that's where our sweet spot it.

[00:33:24] JM: Postman has an abstraction called collections. Can you explain what a collection is and how it's useful?

[00:33:31] AA: According to our definition, a Postman collection is an executable API description. What that means is that unlike static descriptions, the request that are there in a collection file are ready to be sent and executed. A postman collection can contain templates of API requests. It can contain documentation about those requests. It can also contain scripts that are supposed to be run before and after the request. Those are powered by the Postman runtime.

A Postman collection at its core is this. Actually, one more important thing it helps you with is storing authentication information securely. So you can say to Postman that I want my API to be authenticated in this way, like what auth one, or auth two or you want to use basic calls. All of those things, all those pieces of metadata are inside the collection.

A collection can be used for anything in your API development cycle. So you can use it for testing. You can use it for prototyping. You can use it for combining like different APIs together. Again, unlike static API descriptions, which are meant for one API, a collection can contain multiple APIs together. There are lots of these use cases of APIs that spring from its core definition of often executable API description.

[00:34:48] JM: Earlier you mentioned something called Newman. What is New man?

[00:34:53] AA: Newman is a command line companion of Postman. That is a Seinfeld reference there. That was actually an interesting line in Seinfeld which said that – Newman in the show is the last name of the character there. Says that, "Jerry, the postman always has to kind of keep running. It has to continue delivering mail all the time. It just never stops, Jerry." That's what

Newman does. Newman runs in the background for you, whether it's part of your CI environment or if you want to script it using batch scripts or whatever you prefer. It takes those collections that you create in the Postman authoring environment and you plug it in and it runs exactly the same way as it runs in Postman, because they share the same runtime. You can use it for a lot of different ways. Yeah, Newman is what helps you keep running.

[00:35:43] JM: Right. I forgot that Newman actually is a postman in the show.

[00:35:47] AA: Yes. That was the reference there.

[00:35:49] JM: That's pretty funny. Users can [inaudible 00:35:51] APIs within Postman. Why would I want to do that? That almost sounds like a tool of an IDE.

[00:36:02] AA: Yeah. As we're talking about Postman as a collaboration platform, I think – The second thing we talked about, like this philosophy of API first. Typically, what we saw developers starting with was their IDE, writing some code in and then kind of going on from there. What we understood was that this was such a common pattern, and APIs have become kind of like this separate abstraction altogether when you kind of put code aside. That needs more attention, more tooling, more specific kind of guidance around the process of building APIs. What we did with this feature here was, of course, let you use the format or specifications that you're coming from. You can connect Postman to a repository. So we can fetch a schema that you probably have written before and then kind of refine it in Postman's API feature. But it then goes back to like just helping you tame through being API first.

Before you write a line of code, before you jump to your editor, you start specking out and prototyping your APIs there. Once you're ready, then you kind of hand it off for development and actually writing out a business logic. The idea is that you should not comment too much until you know exactly – Or let's say not exactly, or generally with a fair degree of margin of error that this is the API [inaudible 00:37:18].

Now, what that feature lets you do is not just write the specification, but also help teams guide through this whole process of taking the API all the way to production. One thing we understood was that every team rediscovers this process that I should have probably prototyped my API at

least once between frontend and backend developers. At least I should have written some documentation before.

What happens with this kind of API last approach that we call it is that you're kind of always catching up to what is already there. So you kind of have to understand what's going on. When you're catching up, there are always kind of new demands. All of your engineering processes around building APIs fall behind. The feature helps you structure those [inaudible 00:38:02] and you can, in the API feature, actually link together your collections with specific versions of your API.

For example, if you have like 10 collections which run your API test suite, you can link them all to the API feature in Postman and say, "These are all my collections of version one. These are all my collections of version two. These are all my collections of version three." Then you can run them all together in one go. This is more intended for software architecture. If you want to structure the engineering process of building your API, you use that while also making sure that your developers and your testers are also on the same page with the tools that they are using.

[SPONSOR MESSAGE]

[00:38:50] JM: If you haven't tried Airtable, one way to think about it is like AWS with a visual component. And I say that because it's a very new way of creating software. Just like when AWS came out, it was quite a new way of creating software. Airtable is a low code platform with the ability to become code-heavy if you want it to.

Airtable has gone through years of development as a modern approach to a spreadsheet. And beyond being a spreadsheet, it's a modern approach to a database backend. With that effort gone through, Airtable introduced blocks, which is a system for building rich application components that sit on top of your Airtable backend providing a user interface or a map or whatever you want out of your Airtable block.

Airtable is now releasing custom blocks so you can build your own Airtable components. And if this doesn't sound exciting, that's probably Because you haven't tried Airtable. You can make Airtable custom blocks out of JavaScript and React components and they work through the

Airtable SDK to enable developers to make their own functionality on top of Airtable. You can make a modern authenticated real-time CRUD app for users, their admins or for yourself. And if you're looking for the right time to get started, that time is now. You can enter the Airtable custom blocks hackathon at airtable.devpost.com and get started. It's a great hackathon. It's $100,000 in cash prizes, and you can start by building a database, then move up the stack to build a fully-fledged custom block using the Airtable SDK.

You can really build anything with Airtable, and you can compete to win $100,000 in cash prizes. So it's quite a good time. So just go to airtable.devpost.com, try out a new way to build software with Airtable. Thank you to Airtable for being a sponsor of the show.

[INTERVIEW CONTINUED]

[00:40:56] JM: Can you say more about that workflow? How does it benefit developers to spend all that time specking out the API before the actual creation publication of the API?

[00:41:12] AA: I'd say for a developer working on API, that's probably one of the most important things that they should be doing. Typically, that exercise would be done on whiteboards or on structured documents. What it helps you do is catch issues and bugs and I'd say even mentally [inaudible 00:41:34] about your API very, very early on without overly commenting to a process of developing code. If you look at a regular sprint cycle two weeks or four weeks, once you are in that sprint cycle and you are kind of writing code upfront, it can always be very taxing to go back and repeat your work, sort of test those to rewrite that code to retest those APIs. Just because you found out, "Hey, that thing was not needed."

Some of these practices for us actually come from the product design world, right? Product design doesn't start with, "Oh, let's go and build like the best thing like right off the bat." You want to understand who your consumers are. You want to understand what their needs are.

If you bought it from that analogy, like writing code is just shipping out the first thing that you thought of right away. But when you spend that time and energy earlier in the process, you just prevent so much pain down the road. The more people you involve in the process, it's just so much better. If you can involve your frontend developer early and they can say, "Hey, I don't

actually need like 100 reports every time. I'm fine with just 10 reports. My API might be performant that way."

The quality engineer asks you to check for, "Hey, have you looked at –" I don't know, a time zone setting early on into the process. You can prevent all these things very early on, Because once an API goes out in production and an API consumer starts using it, we talked about item slot before, then you can take the API out. It's just out there. I mean, we have seen companies supporting hundreds of like dated versions of the API with small increments over time. You just can't take the API out because you have established a contract with a consumer to deliver that API. That early effort helps developers have an easier life. It helps consumers. It helps engineers who are deploying the API. It also helps architects as they scale up from one API to hundreds of API within an organization. They can check for compliance tools, governance tools, all those things that are a critical part of delivering APIs at scale.

[00:43:40] JM: I'd like to understand better what is on your servers. If I think about my Postman client and I'm hitting an API that is hosted on my infrastructure, what is going through Postman and what is getting saved on your infrastructure?

[00:44:01] AA: The API runs on your Infrastructure if you sign up for a Postman account. Then we can help you back up your requests and those requests would be stored on our platform. That is what then eventually powers collaboration and sharing. You can be as flexible like as you like. Sometimes if you want to sync responses, if you're in a development environment, you can do that or you can [inaudible 00:44:27] and those things would not hit our servers. You can keep your keys and passwords locally on your client and parameterize them using what we have as environment variables. So you can sync only the metadata parts on the collaboration platform that we have on the cloud. But everything else runs on your Infrastructure and only the things that you need for sharing and kind of backing up is what gets to our cloud.

We are not in the pathway of sending those requests. Postman runs locally on your machine. Newman runs locally on your machine. When you decide to opt-in for those things with an account, that's when these things become available to you. Also, you can still use Postman in a totally locked down client state if you want. You don't have to create an account. The full feature set of all the client-side tooling is available to you and the same goes [inaudible 00:45:18].

[00:45:19] JM: Can you help me understand, what's the bigger vision for Postman? Where can you go in terms of API collaboration beyond the things you've already built?

[00:45:33] AA: We're getting hundreds of requests like every week on where we could go. Almost all of these stuff that we have built are actually coming from developers themselves. I actually thought when we started the company that we'll do like these five things and we'll be done. We drew this beautiful ideal cycle of the API testing, mocking, monitoring and we thought we'd be done. Then what we've seen is that people just kind of want more every single week.

Some interesting challenges that we're seeing is in the performance testing and security testing of the API. Running API integrations for people. I'd say some of our shorter time goals are basically bringing this whole community of people building and consuming APIs on the same platform. Recently, we introduced to what we call the Postman API network through which public API publishers can come on Postman and publish the most up-to-date versions of their APIs on Postman, which become available for other developers to consume. Where we envisioned this going on is all of the world's APIs are available for all the developers in the world war on Postman. We use those APIs in a very effective way.

[00:46:44] JM: It's a big vision. It's like an index of different APIs that are available to you.

[00:46:51] AA: Coming directly from the publishers.

[00:46:55] JM: Tell me more about the business. What's the current business structure and what do people pay for?

[00:47:05] AA: Our customers pay us for collaboration and for the tools that they use on the cloud when they want to use them at higher quantities. We give away the Postman client some amount of collaboration for free as well as most of our tools for developers to work with. When you start paying for Postman is when you have adapted Postman for your team's workflow, and that could be a small team of 3 people or probably a large company to scaling up to hundreds of thousands of people in Postman.

We have three plans. We have a team plan, which for smaller teams. We have a business tier, which works slightly larger teams. Then we have an enterprise layer which consult all the things that typical enterprises need in terms of custom roles and what we have in the roadmap as reporting and user groups and all of that.

When you start deploying Postman at scale in the organization, you would typically go for enterprise. But most our customers actually just buy the product online on team or the business tier themselves, and they [inaudible 00:48:10] pay for collaboration and some of the cloud-hosted tools.

[00:48:13] JM: What are the biggest engineering problems that you're faced with right now?

[00:48:18] AA: I'd say one of our biggest focus has been just performance. Postman, we are basically helping developers work with all the APIs that are in existence in the world. It's scaling up Postman as a product and as an engineering system to accommodate for all the ways in which devices are connecting with each other.

We first started with HTTP APIs. We are actually going to be supporting asynchronous APIs in the future, like web sockets or other asynchronous technologies. That's a very big challenge for us. How do you kind of keep the simplicity and the performance of the product so that we will then kind of do all of these? A big thing for us is also like the pace of software delivery, like how quickly can we iterate on the product and kind of get things out to production. We made a lot of progress on that. We ship like new version every two weeks and on the cloud, but we ship something every day. On the backend side, we just have to work with tons and tons of data that we have to delivery to developers in a real-time way. We built this massive kind of infrastructure on AWS, but the experience for developers is still what they've learned to love.

To kind of give them their experience with hundreds of people working together in their time with hundreds of thousands of APIs across all of those different kind of APIs is a huge challenge. That's how we kind of get like something every week. Once Postman starts getting used, and you're like, "I want to send every API in Postman and I want to send every API in Postman with everybody else." It's just crazy to just [inaudible 00:49:57]. But what we love. It's been amazing for us to handle that.

[00:50:02] JM: Are there any management lessons that you've learned as you have incidentally gone from building what started off as a side project and eventually holding a large company?

[00:50:16] AA: I learned a lot from my mistakes. That's how you kind of like eventually end up understanding the difference between what you read and what you implement. I'd say one of the biggest things for us was – Or at least like that's the one that's kind of coming top of my mind, right? When you start with a small team or a sort of developer and you're doing everything, right? You're doing the marketing. You're doing the selling bid to whoever is required. You're doing the recruiting. Scaling up that system requires like a mix of different talents. In the initial phases, you're not ready to – You don't probably even understand all the bits and pieces that kind of go in.

At what time you kind of give control of it while also trusting that this other person who's coming, this team that is coming in will operate the right way. We have kind of gone on both ends of the spectrum. We were like, "Hey, this is big challenge. Go and build it." What we learned was that people were excited but they would freak out. If you had to push an app to like a million people next week, you would really thought people would be excited about that challenge because you're doing it, but they just weren't ready yet.

We had to build these systems so people could come in. They would learn a little bit. Every new developer who comes to Postman actually [inaudible 00:51:33] Postman. Their first job is to build a Postman collection. Until you have built a Postman collection and demoed it, we don't expect that you'll learn the product yet.

We kind of have to get people to execute at the maximum scale possible. We had to build those guardrails that they can walk through before we let them free. I guess my biggest lesson was that I implicitly assume that everybody else would know as much about Postman as me would be as excited with Postman as me and they would just go for it. What else is there to not like? I learned that I had to do some work there to get people up to speed, and they were as excited as I needed to be, but we need to give them the guardrails before they can execute.

[00:52:15] JM: Okay, last question. Are there any technologies or SaaS or platforms products that have surprised you in how useful they were in building postman?

[00:52:31] AA: I think one of the things that we saw a lot of value in eventually was using Looker as a platform. One of the fundamental problems that we were running into as we were scaling the organization was different teams at Postman would have different dashboards. Engineering would have something in Grafana. Sales would have something in Salesforce. Product would have something somewhere else.

What we decided to do was basically have a dedicated team to help us manage all of these APIs and integrations and effectively map to us for public APIs. But building and managing them all through Looker. I think Google acquired the company recently was very, very useful and it helped the company a lot into understanding all these different aspects of Postman both as a product, as an engineering challenge, and as a business challenge.

[00:53:24] JM: Okay, Abhinav. Well, it's been really great talking to you. Thanks for coming on the show.

[00:53:26] AA: Thank you. Thank you so much, Jeff.

[END OF INTERVIEW]

[00:53:38] JM: Vettery makes it easier to find a job. If you are listening to this podcast, you are probably serious about software. You are continually learning and updating your skills, which means you are staying competitive in the job market. Vettery is for people like you. Vettery is an online hiring marketplace that connects highly qualified workers with top companies. Workers and companies on the platform are vetted, and this vetting process keeps the whole market high-quality.

Access is exclusive and you can apply to find a job through Vettery by going to vettery.com/sedaily. That's V-E-T-T-E-R-Y.com/sedaily. Once you are accepted to Vettery, you have access to a modern hiring process. You can set preferences for location, experience level, salary requirements and other parameters so that you only get job opportunities that appeal to you. If

you have the right skills, you have access to a better hiring process. You have access to Vettery. So check out vetter.com/sedaily and get $300 signup bonus if you accept a job through Vettery. Vettery is changing the way that people hire and the way that people get hired. Check out vetter.com/sedaily an get a $300 signup bonus if you accept a job through Vettery.

Thanks to Vettery for being a sponsor of Software Engineering Daily.

[END]