EPISODE 1132

[INTRODUCTION]

[00:00:00] JM: The most popular email client is Gmail, the web-based email client from Google. Gmail is dominant, but that dominance has come at a price, namely speed. Gmail caters to the lowest common denominator serving a large ecosystem of use cases and plugins. This makes for a slow overall performance.

Superhuman is an email client built for power users. Rahul Vohra is the founder of Superhuman and joins the show to talk about the design and engineering of an email client that is made to be fast.

I want to mention that we're looking for writers and podcasters for Software Engineering Daily. If you're interested in either of those things, email erica@softwareengineeringdaily.com. We'd love to have your applications for writing and podcasting.

[INTERVIEW]

[00:00:51] JM: All right. Rahul, welcome to the show.

[00:00:52] RV: Great. Well, thank you for having me.

[00:00:55] JM: First question, very simple. Why are email clients slow?

[00:01:00] RV: Email clients are slow, because no one [inaudible 00:01:03], I believe has tried to make them superfast. It is in fact possible, but it requires a tremendous amount of work.

[00:01:11] JM: Elaborate.

[00:01:14] RV: Well, the main reason that no one has tried to make them fast is because those organizations who had email clients were busy doing other things. Take Gmail, or take Outlook for example, Google or Microsoft. Incumbents like these gigantic companies really struggle with speed for a number of key reasons. Number one, they have massive scale. These organizations are huge, tens of hundreds of thousands of people. Number two, they have this insane entrenched architecture. The number of service, for example, that powers Gmail would boggle the mind. And number three, they have to serve massive scale. Gmail alone has 1.5 billion users who run all kinds of different hardware all the way across the world.

So that optimization problem is really, really tough, as well as the engineering problem, as well as the organizational agility problem. How do you solve those three things? I don't claim to have the answer. I think it might be impossibly hard, which is why Starbucks are even a thing.

Now, why is it actually hard to pull off even for a startup? Well, it’s only just becoming impossible for a startup to build a high-quality email clients, high enough quality and amazing enough that people would genuinely choose to use it. Not just use it, but pay $30 per month for it over Gmail. This has become possible due to things like decent descent APIs, the Gmail and the Outlook APIs significantly better than IMAP or SMTP for anyone's worked with those, or MAPI or ActiveSync before them. Better distribution models, and it’s just faster to grow a technology company these days. So it's just become possible for a scrappy little startup like Superhuman to build the fastest email experiencing in the world and actually have people prefer that over the incumbents.

[00:03:01] JM: How does slow email inhibit the quality of interaction?

[00:03:06] RV: It's a death by a thousand cuts. Well, let’s actually do it by example. So, when Paul Buchheit, who was the employee at Google who created Gmail sat down to do so, he had a rule, the 100 millisecond rule. And this basically said that every interaction should take place in 100 milliseconds or less. Now, if anyone who’s used Gmail in the last 10 years will know that it no longer really abides by that rule. Opening an email can take a second. Searches can take many seconds.

And so when you have this slowdown, it's not really about the physical cost of the time. If you add that up, it will come to perhaps half an hour, to an hour a day maybe depending on how much email you do. It's the effect it has on your mind. So there’s this notion in human-computer interaction that’s actually being very heavily researched, that every time you change task or get interrupted, it takes your brain about 20 to – On average, 23 minutes to recover.

So, if you alt-tab or if you go to Twitter or if somebody taps you on the shoulder and you stop doing what you're doing and you start doing something else, even if it's for just 30 seconds. When you get back to your original task, your brain is still computing the old thing. And the most insidious thing is you don't even feel it.

So when our email go slow or when our software in general goes slow, you’ll realize. And I’m an aficionado and an advocate for speed. You risk losing flow. You risk this cognitive overload, this distraction effect. And that's why it's so important to never have that. If you’re a software engineering or you’re a founder trying to build high-quality software, you want to squeeze everything down to the last few milliseconds. And we’ve now got to the point in Superhuman where every interaction takes place in not just 100 milliseconds, but less than 50 milliseconds, and it's to keep people focused. It’s to keep people in flow. And I think it's one of the main reasons why people report saving many hours per week.

[00:05:08] JM: Is it anyway good for a client to be slow? I mean, we’re not talking back chat client here. Like maybe email is good if it's slow, because it's a pensive operation.

[00:05:19] RV: I think whilst you're writing your emails – Well, I was about to agree with you. But even there, I disagree. Imagine you were doing a pensive operation, but you tapped a key and there was a delay between what you tapped and where it appeared on the screen. And Gmail and Chrome, especially if laden with browser extensions can easily get to that state.

We probably all have the phenomenon of hearing ourselves talk but delayed by perhaps 200 or 300 milliseconds. And we know how cognitively overloading that is. So, I can't imagine any aspects [inaudible 00:05:49], which I’ll get to shortly, where slowness is actually a positive thing. Because I think it's very anti-user.

Now, the one case where slowness is a positive thing is potentially delaying in receiving and sending. I am very much a believer in humanistic technology. I don't like, for example, how Slack has designed to engineer fear of missing out. The whole threads mechanism, the whole unread channels mechanism is designed to keep you coming back minute after minute, second after second. And I think an email, we can actually start to solve this.

So this idea of perhaps pausing your inbox so that your emails are delivered at 3 known times per day. And the symmetrical idea of pausing your outgoing email so that maybe your email gets delivered to other people in three batches a day. I think that's a really good example of slowness. But if what we're talking about was how fast does the software feel, then speaking actually as a prior game designer and as a student of psychology in the areas around us, I would say absolutely. It has to feel fast.

[00:06:53] JM: Okay. Let's talk about the elephant in the room. Why is Gmail slow?

[00:06:59] RV: I think it's the three reasons I articulated earlier. You've got a very large organization with hundreds if not thousands of people working on Gmail. I doubt that if you were to get all of them into one room or even ask them individually and say, "What is the number one priority for Gmail?" Any one of them would say make it fast. Or maybe a handful would, but that's now point something percent out of the thousand people we spoke to. So, there's this organizational agility issue.

Secondly, you've got the architecture issue. Gmail is fundamentally a client/server web-based architecture. Superhuman, to my knowledge, is the first time anyone has ever built a JavaScript entirely email client. It looks like a web page if you go to our webpage. We also have a native app. But it isn't actually. You actually end up hissing in Amazon S3 buckets. You download a JavaScript payload, and it's the first time anyone has built an email client entirely in JavaScript.

And so, we don't even get slowed down by the network. When you search in Gmail – And Google is excellent, by the way, putting servers next to everyone. You're only ever one or two miles away from a Google server. It still doesn't matter. You're still limited by the speed of light and network traffic and hubs and so forth. When you search in Superhuman, you're only limited by the speed of your SSD, which is frighteningly fast.

And so there's this architectural issue that prevents Gmail from reaching the speed of what native clients can achieve. And we did a whole, whole bunch of work on figuring out how to download, store, and index you're your email within the browser, within Chrome itself so that it's as fast, if not faster actually than any native email app, as well as significantly faster than Gmail.

And then the third thing is this heterogeneity of audience. Imagine if you were serving 1.5 billion people on everything from an underpowered machine from 20 years ago to a $5,000 iMac. How on earth do you make it fast for all of those people? I mean, it's next to impossible. The heterogeneity is through the roof.

Whereas we've focused on a very specific subsets of users, those people for whom email is work, work is email. They do three hours or more of email a day. For them, $30 per month to say hours per week is a no-brainer trade-off. They would take that deal every single time. And they tend all to be unreasonably high-end hardware. So we can actually optimize for that. We can take advantage of the fact that we know the user has potent hardware at their disposal.

[00:09:26] JM: And I think it's worth noting that there are all kinds of things you can do with that extra hardware. You can aggressively preload all kinds of emails. You can do lots of aggressive caching. You could do lots of aggressive prefetching. Can you tell me a little bit about that kind of aggression that you can put into an email client?

[00:09:43] RV: Absolutely. All of the above and more. So we do aggressively pre-cache, pre-downloads of the labels or the lists, if you will, that we think you're going to attend to. This is obviously going to include the inbox. It's going to include sent mail. It will include other labels that you seem to be using. And we'll sync those lists quite far back into the distance. But having said that, we did want to avoid what we found to be one of the key sins committed by other native third-party email clients.

So in the first hero superhuman, I actually interviewed north of 1,000 potential customers on average, two or three customers day on what their email client pet peeves were. And for the Gmail users, I think this will come as no surprise. It was, "I don't know. Gmail, it slows me down. It's clunky. I have to click continuously. It doesn't sync properly against mobile. It doesn't work properly offline. It seems to consume tremendous amounts of battery."

And then for the people who use things like mail app Spark or Airmail, any kind of third-party native email app, it was actually – This thing doesn't seem to sync reliably. This thing seems to be unstable. This thing seems to be buggy. This thing seems to take my CPU and peg it at 100% plus constantly. And so this is one of the reasons why I'd always recommend any founder, like talk to a thousand people in the first year. It's going to be so valuable in terms of getting into product market fit. What we learned is that, yes, you can aggressively pre-cache so many different things. But you should only do it insofar as the machine is capable of doing so.

So we actually have two speeds of pre-caching. We have one speed if you're not connected to the power, and we have another speed that's about 10 times as fast if you are. And so we're very respectful of the current status of your hardware so as not to drain it unnecessarily, which I think basically every other email client doesn't care. It's just like, "Cool! We're going to download everything." And then for the next 48 hours, your machine is overloaded and running hard. If it wasn't connected to power, it would die pretty quickly.

[00:11:50] JM: Let's come back to Gmail. Gmail has this whole plugin ecosystem, which I mean I wonder if the plugin ecosystem actually slows down Gmail. Obviously, it does, if you have a lot of plugins installed. By I wonder if you don't have plugins installed. Is there some kind of overhead that is just material that is inherent to this plugin ecosystem? Maybe you could tell me about some of the other causes of slowness in Gmail, perhaps the plugin ecosystem?

[00:12:18] RV: Indeed. The plugin ecosystem is a huge contributor to this, and it is in fact one for which I feel rather guilty. You see, my last company, Reportive, was the first Gmail plugin to scale to millions of users. And I'm guessing about half our audience will remember this. But for the half that doesn't, we showed you everything about your contacts right inside your inbox. When emailed you, we showed you what they looked like, where they worked, their recent tweets, links to their social profiles. We grew rapidly, and two years later we were acquired by LinkedIn.

And at LinkedIn, I ran all of our email integrations. Now, the thing is, we were the first Gmail browser extension to do that. And I remember in that first year when I started, I told people what I was going to do. "Hey, I intend to build this app. It's going to be a venture-backed startup. It's going to be a fairly complicated app actually, and it's going to live inside Gmail."

I grew up as a programmer and as a coder, and coding buddies would say to me, “Hey. So, you’re going to build this really complex multi-megabyte app and stick it inside somebody else's really complex multi-megabyte app that is constantly being AB tested, changes every day. It doesn't have a CSS or DOM laid out sensibly to help you, because it’s all compiled.” And there's no API. There is no plugin architecture, and that's going to be your business? You’re insane.”

Everyone tried to talk me out of doing it. I think what we showed is that you could actually build a really compelling product that way. And since then, you have things like Boomerang, Mixmax, Clearbit, Yesware, you name it. People have it. And each one of these is built without an API. There is no plugin architecture. It’s only recently that Gmail actually has a sidebar architecture, but that has all of its own pros and cons and most people actually still use browser extensions, because it could do so much more power. The problem though is that because there is no architecture, what do you do? Well, you and doing window.settimer. You end up tracking every mouse move events. You end up doing things that, yes, they let you create a product that’s powerful, but let’s be real. This isn’t how engineering should work. It's not going to be the most friendly thing to the user’s hardware.

Yes, absolutely. This whole plug-in ecosystem, this whole browser extension ecosystem for Gmail that we helped kick-start at Reportive slows down your Gmail experience. In fact, Gmail plus plugins inside of Chrome is the single biggest reason why your laptop runs out of battery. And I don't like that. I kind of feel a little bit guilty for that. It's not what I had intended when we built Reportive. And in many ways, Superhuman, it's kind of like a love letter to the industry, if you. Well, what would happen if we just rebuilt this entire thing from scratch? What if instead of this disparate collection of apps and experiences, it was beautifully designed and as if it were meant to be together.”

And so that's when we imagined this email experiences that’s blazingly fast, where search is instantaneous. Where every interaction is hundred milliseconds or less, where you never have to touch the mouse, where could do everything from the keyboard. Just worked offline, and it had the best functionality from the best Gmail plugins built-in natively and was still somehow subtle, minimal and visually gorgeous. And so there was this direct transition from that to Superhuman.

[00:15:44] JM: Sorry. Are you saying that the plugin ecosystem from Gmail works inside of Superhuman?

[00:15:50] RV: No. We're taking a very different philosophy. Let's do this by analogy. You can take an ecosystem like Android. Android has obviously Google who work on the operating system, parts of the operating system itself minus their apps open source. So anyone can take it to make their own. There's any number of handset manufacturers, from Samsung, to HTC, to Huawei. You have the Playstore, but you other Android stores as well. A disparate way of [inaudible 00:16:17] developers. It's a very heterogeneous ecosystem.

And as a result, the Android experience is somewhat lower quality and somewhat a little bit more out there and a bit more Wild West than the iOS ecosystem. Now, the iOS ecosystem and Apple in general is much more vertically integrated. If think about why your iPhone or why your MacBook is so – It feels so design, is because the same company made the hardware as made software. Indeed, there is a phenomenon that directly addresses this.

Most software engineers will have read or come across Conway's Law. Sorry. Not Conway's Law, because most people don't know that. Most people know *The Mythical Man-Month*, obviously written by Fred Brooks. In that book he mentioned Conway's Law. Conway's Law states that every app of service grows to represent the structure of the organization that made it. And if you look at the Android experience, if you look at the Apple iPhone or Macbook experience, you can literally see the structures of the companies and the ecosystem behind then.

And if we just glommed-in Boomerang or if we just glommed-in Clearbit, we'd be no better than Gmail on those things. So what we instead decided to do is we took a page out of Apple's strategy and said, "Well, what if we vertically integrated this as far as we possibly could? What if we designed the whole thing? Well, then we could unlock a quality of user experience that nobody else could compete with."

And so for the functionality, that really matters. We'll never claim to replicate all the Boomerang, because that's just too much. But for the stuff that really matters, we've built this ourselves natively inside of Superhuman.

[00:18:06] JM: Understood. Before we get into Superhuman in more detail, while we're kicking around Gmail, are there any other design issues that you see within Gmail?

[00:18:15] RV: I mean, I could talk about that for hours. I think the biggest one, and I say this as having been a product manager at LinkedIn. I know what it's like to be in a large tech company. This is less of the design issue and more of a political issue. You see, the way that product managers tend to be rewarded in companies like these are by pushing the needle. Now, you don't push the needle usually by building new things. You push the needle especially when a product has scaled by making something a percent better, or 2% better.

And yet not everyone can do that. So the growth organizations in these – The growth organizations and these companies tend to be rather small. So what do the other PMs do? Well, they go out on their own and they sort of make new features. But those features tend to be starved resources by comparison eve to a startup. This is one of the most eye-opening things about moving from Reportive to LinkedIn. It felt like before LinkedIn, I had access to way more resource as a venture-backed company, even one that had only raised a million dollars compared to being at a $30 billion company. Because in the $30 billion company, you're competing politically not really based on merit a lot of the time against everyone else who needs engineering resource or marketing resource or whatever it is.

Whereas a venture-backed company, it's literally just how much buzz can you make and how much money can you raise and how much revenue can you create from your users. So there is this notion in a large company of just how difficult it is to get stuff done. And I remember estimating at LinkedIn, we moved approximately six times more slowly than when we did as an independent company, as Reportive being independent. And I don't think that's unique to LinkedIn. I think LinkedIn is a phenomenal company. I don't want to sound like I'm dunking on them. I just think that's the tax you pay being a large organization. And I can only imagine Google is slow.

[00:20:07] JM: Okay. So now that we're talking about design, tell me some of the high-level design decisions you've made in building Superhuman.

[00:20:14] RV: So, we are very aggressive about keyboard shortcuts. We have a strong belief that the keyboard is faster than the mouse for about 90% of the things that you need to do in your email. There are a few things where the mouse is faster than the keyboard. Famously, just selecting random points on the screen. Let's say you're playing a first-person shooter or you're doing Photoshop type work, or you're just trying to create a selection. That is an example of something that's faster with the mouse. But by and large, the keyboard is significantly faster.

And so in Superhuman, on any given screen, there were only three buttons. And that has been the case since the start of Superhuman since five years ago. And I challenge folks listening to think of a product where that's been true. Think of a product where for five years the designers have resisted adding a button, but have still nevertheless continued to add feature after feature to product. We have.

And the way that we have is that we have this paradigm that we've borrowed from the developing ecosystem called superhuman command, a.k.a. our command palette. If there's anything you ever want to do, just hit command K and up will appear Superhuman command. And simply just type in the thing that you're trying to do.

Not only will it match it for you and you can even make a spelling mistake. We have a really cool fuzzy matcher. It will show you the direct shortcuts to do it next time. And this is one of the key innovations that allows us to add arbitrary amounts of functionality, and Superhuman now is such a powerful tool without ever encroaching on the beautiful minimal premium design that the user-interface has. So that's one example.

Another example would be single tasking by default. You've got me on a [inaudible 00:22:03]. So, bear with me. Let's say you're in Gmail and you hit C, because obviously you're using keyboard shortcuts and you get that little Window to start typing. Well, guess what? You're still looking at your inbox. And email is still going to arrive. The next shiny thing will still distract you, and this goes back to the notion of disruption of flow that 23 minute recovery time. That happens in Gmail to Gmail users hundreds of times a day, leaving people in a permanent state of distraction.

In Superhuman, you single task. When you hit C, you are looking at a blank canvas that occupies the entire screen. We’ve rebuilt Reportive over on the right hand side. That’s purely contextual information. There is no way that a new email could come in and distract you. There is no way that you can break focus or lose flow. Now, if you want to multitask, you can. Of course, there is a keyboard shortcut for that. But the important thing is single tasking by default. So those are two examples. I’ll pause there. But these are examples of how we create flow and help users move through their inbox twice as fast.

[00:23:09] JM: What are some other ways the you've built in performance advantages?

[00:23:14] RV: Performance advantages. So, I think the biggest one is we figured out how to download, store and index all of your email in the browser itself. So if I do a search in Superhuman, it's actually a hybrid search. It searches locally negatively on your device. At the same time as via the CDN on Gmail. And then it merges those things in real-time, which by the way is non-easy engineering problem. It turns out to be quite tough.

But we sold it. It took us about a year or two to figure out how to do that. And now it works really beautifully. So, our searches feel really, really snappy, really instantaneous. That was one thing. Another thing is making every single interaction optimistic. Let me give you an example. In Gmail, you hit send and the UI actually blocks on send-in. Now, about half the time you don’t notice it, because about half the time you're going to be on a fast Internet connection. But for anyone who’s used Gmail whilst tethered or perhaps in a plane and back when we were all flying around or in just some of the low bandwidth connection, you’ll know that when you hit send, it can block up your UI in the region of 5 to 10 seconds or longer depending on the quality of your connection. That to me is in insane. Every single interaction in Superhuman, whether it’s archiving, storing, sending, it really doesn't matter. Every single one is optimistic. It doesn't block on the network.

The only thing it can possibly block on is disk, and we're riding the trends that everyone basically has an SSD at this point or at least a hybrid SSD spiny disk. And so, the interface can feel incredibly fast.

Another example would be our choice of browser. So, we work inside of Chrome. We work inside of a negative app that's based on Electron. Electron, of course, is based on Chromium. These platforms can use newer HTTP protocols like quick and speedy. We're talking directly through Gmail. Google have spent billions of dollars of putting hardware a mile or two from basically anybody. And so the very nature of the transport protocol and of the hardware, and because we don't do – This is something I've sort of glossed over. But we don't take a copy of your emails. We don't actually run servers. That would be slower than riding the coattails of Google. So, this choice of architecture and the choice of using Chromium as the fundamental way to communicate with the network also ends up speeding things up.

[00:25:42] JM: Tell me more about the frontend. You mentioned Electron, or Chrome. But I assume you're using React?

[00:25:49] RV: We are, except for the places where React would be too slow. And one of the classic examples is an infinite list. So one of the fun things I like to do just to see, just to see how fast we are, is go to my inbox and scroll down. And this is a very complex piece of UI. You imagine you have 20 to 40 items on the screen. Each item has recipients. They have – Which each individual React objects. They have a subject line. They have buttons over on the right-hand side. They have checkmarks that could be clicked or hit with keyboard to select he thread and so on.

Now, imagine you potentially have hundreds of thousands of those in your inbox. Well, clearly, that would bog down the page to an unreasonable degree. The DOM cannot get that large. Your JavaScript stack shouldn't get that large. So what do you do? Well, most people who've done this know that to render an infinite list, you actually just keep a subset of the list around. And as you scroll down, as things scroll up off the top of the page, you un-mount them and you remount them down at the bottom. It turns out that even doing that in React was too slow. That is the only part of the app where we've eventually unrolled it and hand-rolled it.

[00:26:58] JM: Very interesting. Well, we talked a little bit about the frontend. Tell me about the backend.

[00:27:03] RV: The backend uses Go, and it takes care of things like making sure that an email gets sent even in the most adverse of network conditions. So, we all had this experience. And I've heard this a number of times back when I was asking people what their pet peeves about using Gmail were or any other third-party email clients. The problem when you hit send and then you're in a rush, so you close your laptop, and then you travel somewhere. And the worst case scenario, this could be like days, because you're going to the airport or something. And you open up your email again and you connect to the internet and you realize that email didn't go, because you closed your laptop too soon.

So, the way that we solve this is as you're writing the email, we actually stage it, just the draft on our servers on the backend such that the instant you hit send, it's actually gone. Of course, we wait a little bit for undo send. But you're safe to close your laptop the instant after you hit send. You don't even have to wait for attachments to upload or anything like that. And so, that's possible. What the backend does, it goes a lot deeper than that as well.

Another example of what the backend does, and again, this is all written entirely in Go, is this concept of a mass archive. So again, some of our listeners would have had this experience where they want to decline email bankruptcy. And so what you do? Well, you go to Gmail, because you're not using Superhuman yet, and you select all, and then you click select all, really, and then you hit archive. Well, if you've got more than a few hundred messages selected, God help you. Because if you have a thousand or more selected, you're actually going to rate limit yourself out of Gmail. And we've had customers who've done this prior to coming to us said, "I got myself rate limited out of my own Gmail account for two days at stretch." There was no customer support I could talk to. I just had to wait for the machine to let me back in again.

Now, they use case is very legitimate. It is totally reasonable to say, "I am never going to reply to any emails that are older than seven days. I really just want to archive them more so I can search them. but I don't want them in my inbox, because I actually want a chance of achieving inbox zero."

So we do this on the backend. And what we do is during our onboardings, as you might know, we onboard every single user one-to-one. We actually ask that user and we say, “Hey, we’re super keen on getting you towards inbox zero. How about we archive everything that’s older than, let's say, few days. And obviously, we can leave the starred items around, or the unread items around, or whatever permutation filter that you want. But we’d really recommend getting rid of those old emails and getting them archived.”

This is something that the backend does. And we’ve built a rather complex piece of machinery that can take an instruction like archive everything older than three days that is not starred and which is read. And it will run that process gently in the background over time. Always keeping an eye on the rates and never getting you rate limited. It can be stopped. It can be paused. It can be reverted. It can be resumed. It's controllable by a user from the UI. And this is one of the many responsibilities of the backend.

[00:30:10] JM: Email is kind of interesting, because there's a lot of functionality you can push to the client. You can have some functionality in the server. You can have as much as you want kind of in the client though. How do you balance that functionality? Would do you try to push into the client to make it more eager?

[00:30:27] RV: In as much as possible, we put almost everything in the clients. This few exceptions would be where we want to share technology or capability between our mobile clients and our desktop clients. Because of that point, there is potentially some speed of development benefits to be gained. And also, lower maintenance of overhead. You got less code to look after and having it on the server.

But for the most part, it's very, very heavy clients. I mean, this is by design. It’s part of our strategy. We know that our customers tend to have really beefy clients. And so, clearly, we should take advantage of that in delivering the fastest email experience in the world.

[00:31:09] JM: How eagerly can you grab emails from the server?

[00:31:14] RV: Pretty eagerly. I forget the precise rate limits. If I recall, it's something in the region of 50 requests per second to Gmail. If you go more than that, then obviously you’re going to get rate limited. And the degree to which you get rate limited depends on how much you abuse the system. But that's fairly fast. So you could download a corpus of email potentially over the course of a few hours. It really does depend on the size of the email account, of course.

[00:31:41] JM: Can you tell anymore of the interesting client/server interactions?

[00:31:46] RV: Let's see. I mean, this is just a tiny little long and it's an example of how we really try and go above and beyond. So when you send an email in Superhuman, there's obviously a notification on the bottom left that says message sent. In that notification, it says undo. Now, you can hover over that notification. When you hover over that notification, a really cool thing we've done is the client tells the server, "Hey, by the way, user is in the zone. They might decide to undo sending this email at any point. Hold on to it. Don't send the email. Don't send the email." And this is because I've personally had the experience. I've had users had the experience of, in Gmail, they send an email. They're moving their mouse all the way up to that undo button and the mouse is hovering over it. Guess what? As the mouse is there, it disappears from beneath you. It's the most utterly awful feeling, especially if you said something terrible.

And so, it's a fun little client/server interaction. But if you're hovering the mouse over our undo notification, the client will forever tell the backend, "Don't send this thing." Until you move away for a certain period of time or, of course, you click undo.

[00:32:53] JM: Let's talk about some more engineering subjects. So, what about reliability? It's email. This is really important. Your email messages should be sacred. They should not be tampered with. How does reliability factor into your engineering?

[00:33:08] RV: Absolutely, hugely, to the point where if something is affecting reliability, we treat it critically. It bounces the queue to everything else. Internally, we have to tags that we use to label a potential liability issue. Unfortunately, we haven't had to use either of these for quite some time. One is a lose data, and this is an issue that has potentially caused the user to lose some piece of data. Perhaps they lost a draft, or something else went missing for some reason, or look foolish, which depending on your perspective, could be even worse. This is where the user has done something, which has caused them to not represent themselves in an ideal fashion to somebody else.

Maybe they sent a half-finished version of their email or something like that. Now we haven't had to use these for a while, but in the early days when – Because you’re right. Building an email client is super-hard. The surface area is immense, and reliability is mission-critical. We use these labels to triage what we did.

[00:34:07] JM: What about testing principles?

[00:34:10] RV: Testing principles. I mean, we obviously write a ton of tests, and that’s everything from unit tests to integration tests. I wouldn't say that we have a testing principal. We’re not as extreme as saying, “Oh, we do test-driven development,” or anything like that. For us, what's really important is making sure the product works well in practice.

And the way that we’ve accomplished that over time in addition to using the loose data and look foolish labels is through this notion of onboarding. So, folks might not know, but we onboard every single user one-to-one in a 30-minute video call for Superhuman. Initially, we did so deliberately slowly.

Back in the day, about two years ago, when it was just me doing the onboardings. I would do only about five or six onboardings per week. Why? Because that would give me the opportunity to sit down with each user and watch them do their email for about 30 minutes. And actually back then, the whole onboarding process was two hours. So I got a lot of FaceTime with users.

Inevitably, I would find 5 to 10 bugs. Some serious, most not so serious. But in any case, I would take those back to the team and insist that we fix them before we onboard next week's cohort of users. And why did we do that? Well, if you don’t do that, what you end up with is a piece of productivity software that's like basically every other productivity software that is two years old, that’s immature. It will only be usable by early adapters, because most people, most mass-market consumers have a really high quality and reliability bar. And if you don't fix the obvious bugs, people aren't going to run into or reports the nonobvious bugs. This is what I call bug reporting fatigue. Any given user, let’s say, might only ever have the enthusiasm, or the energy, or the initiative to reports four or five bugs over the course of their lifetime with your product.

Now, if your product is buggier than that, then given a certain number of users, you won't actually find out about the rest of the bugs. That's why it's so important when you're building complex software to on board users slowly, to observe them using the product and to critically fix their issues so that the next cohort of users can uncover their next cohort of issues.

[00:36:28] JM: Tell me a little bit more about how you manage the engineering across your different teams.

[00:36:34] RV: Well, these days, it's very different. Back in the days, I was just describing we have a head of engineering, Emuye Reynolds. She’s amazing. You should totally talk to her at some point. There are only 14 people in the world who’ve been making iOS apps for longer than she has. She was back at Apple in the early days of Apple. She runs our overall engineering organization. She reports directly to me. She also serves on the leadership team. So she takes a hand in running the company. And she also runs customer delight, which is what we call customer support. There’s probably an interesting side conversation there. It kind of relates to your reliability question. But I deliberately decided to put support adjacent to engineering to make the distance between users and support, and therefore engineering as small as possible. And this, it turns out, increases the velocity at which you can fix bugs and maintain quality and reliability.

She then works alongside Conrad Irwin. Conrad is our CTO. He’s a cofounder of the company. I've been working with Conrad for over 10 years, because he was the first employee I ever hired at my last company, Reportive. And he is also phenomenal. Probably one of the best engineers you'll ever meet. He was a co-contributor to Pry. It’s used by about 25% of all Ruby developers. And when we sold our last company, Reportive – Here’s a fun little story for you. We obviously interview a bunch of different potential acquirers. There was LinkedIn, of course, who ended up buying us. But also Twitter, Facebook, Salesforce and a few others.

When Conrad interviewed at Twitter, he scored the highest engineering interview scores of any engineers to have interviewed at Twitter in the entire history of that organization. So that sort of gives you a sense of the caliber of the leadership team and both on the head of engineering side, on the CTO side. And we've continued to maintain that bar.

So we split the team up into mobile, desktop and backhand. And every single engineer on the team has come to superhuman having done something incredible in the past. Whether it's a work achievement or a piece of open source that they've worked on, or something that they've contributed to the community or something that they've hacked on. Everyone's done something really, really cool, and is continuing to do something amazing at Superhuman.

[00:38:50] JM: Let's talk a little bit about the future. Do you see any opportunity with WebAassembly?

[00:38:55] RV: Actually, yes. I was talking about this just yesterday with our dear Conrad. So one of the things I'd like to get to is writing assistance. This is a set of technologies broadly related to helping you write better and faster. I think people are familiar with the Mac OS style auto correction, also available on iOS. Folks will be familiar with Grammarly that gives you style advice. And there's a bunch of things in-between as well.

Now, having done 500 onboardings myself, if not more, I've witnessed firsthand that one of the next frontiers of speed that we can unlock is helping you write faster. Of course, this is something that Gmail, and Google more generally, is working on with things like smart reply and predictive auto complete. I actually think we can do better than that. I'm not going to reveal exactly how. But part of how we plan to do it is by leveraging web assembly, because this stuff isn't easy. You've got run machine learning, or the very least, inference models on hardware. You're going to do it fast and you got to do it in a way that doesn't impede typing.

So one of the things that we might actually do is not write this in JavaScript, how we've written most of the rest of our apps, but actually perhaps work with existing Java libraries or write it in, really, any language that we want. But then compile it down so that it will work in WebAssembly.

[00:40:18] JM: Fascinating. As far as the competition with Gmail, I wonder, can you compete with Google's kind of economies of scale and what they can do with, for example, the smart auto complete? The smart auto complete in Google is A killer feature. I wonder if you can replicate that.

[00:40:37] RV: I think we can. I don't think we could individually as an organization, but it's so obviously beneficial. And I would actually argue, it's more beneficial – This is what I personally find in Google Docs compared to an email. But it's such an obviously good idea that I can think off the top my head, five startups working on it right now. I'm also an active angel investor. So it's something that I pay very close attention to and stay close to.

I think that's – And especially given the advances that GPT3 has made, and we're working closely with OpenAI as well. I think that in a fairly short period of time, it might be a year or two, there will be libraries widely available to do that kind of thing. Probably, actually, I'm going to rewind that. Not libraries. There'll be services. I think that this stuff is hard enough and requires venture capital. You'll have to pay for it, but it is totally feasible that this will become integratable in the order of magnitude of years.

[00:41:33] JM: So you're thinking, this will be available as a service and you can just import it as a library or something.

[00:41:38] RV: Like I said. I know, yeah, at least five teams working on precisely that.

[00:41:42] JM: Fantastic. What else? What's the biggest engineering problem you've had to solve in building Superhuman?

[00:41:49] RV: Biggest engineering problem. There have been so, so many. I'll tell you a fun story of when it was particularly hard to debug something. And debug stories are always fun. So, for example, in order to power our blazingly fast search, we indexed emails using WebSQL, which some folks might know is actually a SQLite light database built into Chrome. It turns out we are basically the only production user of this, I think in the world according to the Chrome team at scale.

Now, there was an extremely esoteric bug in the full-text search of SQLite that crashed if certain specific requirements were met. And this bug was very hard to track down, because the database had to be in a very specific state. In particular, dive into the example just for color. You have to have unbalanced search trees. Unbalanced in such a way where the adjacent node was, in fact, null. So you'd have to have an index with, let's say, a lot of words starting with the letter A. None starting with a letter B, but some starting with a letter C such that when you insert a new word with starting with A, it tries to split the A node, because the A node has got to big. But because B doesn't exist, guess what? It's segfault. And this was happening to hundreds of our customers every single day because Chrome had updated the version of SQLite that they use.

And then our CTO, Conrad, completed a feat that is truly heroic, working feverishly morning all the way through the night. I'm not even sure if he slept properly through one weekend. He jumped into Superhuman. The codebase found out that the issue still exist there. Jumped into Chrome, the codebase found out that the issue didn't exist there. Jumped into SQLite, the codebase gets bisected himself all the way through the recent commits. By the way, this is software that is extremely widely used. It's super battle tested. And found the single [inaudible 00:43:37] line that was causing this crash. He then reproduced it locally, wrote some tests, submitted a patch. SQLite accepted the patch, updated their code. We pinged the Chrome team. They update their SQLite. They released that new code. And now that code with Conrad's bug fix, is running in every instance of Chrome around the world.

Sometimes, the bug is not in your code. It's in somebody else's. In this case, the bug was not in Chrome and not in Superhuman, and it wasn't even in Chrome, but we fixed it nevertheless. And that's just the kind of thing you have to do if you're serious about quality.

[00:44:10] JM: Interesting. Well, any other parts of superhuman that we haven't really explored in much detail engineering-wise?

[00:44:18] RV: Sure. I mean, there's going to be quite a few. What kinds of things would you like to go into?

[00:44:25] JM: Technical things, distributed systems. Yeah, you got any distributed system stories?

[00:44:31] RV: Not really. I'm sure we have. It’s probably a little bit further away from what I personally get involved with. I’ll tell you another WebSQL story, because it's sort of top of mind having talked through all of that. Like I mentioned, we are, as far as we know, according to the Chrome team, the only production user of WebSQL at scale. And there was a point in time where they did not know that. And so they were actually planning to remove WebSQL from Chrome to the extent that they had removed it from upstream versions of Chrome before it hit production, but in canary and in beta.

And this caused a considerable degree of panic inside of Superhuman. Imagine if you’ve worked for years on an app, and you know like clockwork there is this thing coming down the line. You’ve got about four weeks before it hits production, and it will literally break your app that you've built for everyone. An app that asked people are paying you for. And we didn't have long to deal with this tool.

So this really isn't an engineering challenge. It’s more of a sort of an existential crisis for a startup. I mean, how do you deal with this? And this is really going to be more sort of a founder tale. But it's a good example of why you want a diverse cap table. It’s why you want lots of different investors in your company. So, I went to, First Round Capital, who led our seed round, and I said, “Look, this is the issue. I need the highest level introduction to the Chrome team possible, because the folks that we’re talking to right now, they don't seem to be able to help us.” And they figured out the people that they knew who was running Chrome. Tried to get connections, but it just didn't work.

I tried it with our other major investors. It just didn't work. And then sort of as a last-ditch attempt, I downloaded our cap table with every single angel investor who's ever invested in Superhuman, and it’s sort of 200 point at this point. And I basically mail merged everyone, sort of panic email saying, “Hey, this thing is going to die shortly. I'm pretty sure they’re doing this deliberately. This seems like a complete oversight. Please, can someone help us?”

And I got a bunch of emails back saying, “I'm really sorry. I’ve put some feelers out. I’ll let me know if something can be fixed relatively shortly.” And I got one email back actually from Sam Altman who most of our listeners will know as having run YC for a period of time. And he said, “I happened to be a party where Sundar is standing, and he's standing right here next to me. This is Sundar Pichai, the CEO of Google. He said, "Give me an hour, and I'll get back to you."

So, I can only imagine what happened, because at that point, of course, it's no longer via email. But I assume that Sam goes up to Sundar. They have this conversation. The next thing I know, this whole train of events is set in motion, and Google reaches out preemptively like, "We're so sorry. Like this was totally not deliberate. Our goal is obviously to push the web forward. And we didn't know that this was being used in production. Now we do. What can we do? Let's work together." And since then, we've had a really, really strong relationship with the Chrome team, and they've been fantastic to work with.

[00:47:26] JM: That's a great anecdote. Do you got anymore cool anecdotes?

[00:47:30] RV: Not off the top of my head. But maybe you could jog some memories if we ended up talking about something else.

[00:47:36] JM: Well, one thing I wonder is like how much scalability is there in your infrastructure? Does it really matter? This is kind just like one client and it kind of pings the server, and that's it. You don't really need to scale it up, right?

[00:47:51] RV: Exactly. This was one of the core reasons why we decided to architect the way that we did. Okay. So you wanted a fun anecdote. Here's another little fun anecdote. Our last company, Reportive, that Conrad and I also worked together, was the opposite architecture. Everything ran through our servers. And we had a very complex database schema where for each user we stored a representation of the entire world. What they can see, what they can't see. It's sort of the N x M problem for anyone who's familiar with that.

And as a result, we were running the largest database on Heroku. For two years in a row, we have the largest database of Heroku. So large in fact that they had to provision special machines and special services just for us. We couldn't achieve the database sizes that we needed to via their dashboard and their user interface. And it was really, really expensive to run that.

We've all heard horror stories of how expensive Heroku and AWS can get when you're at scale. And for this company, Superhuman, we knew that unless we were careful, it would be significantly worse. Because back then, it was only just contact data. It was only just social profiles. Per person, you're talking about perhaps 15 to 50 GB email data. Clearly, we're not going to store that, less learn for the performance and the security and the privacy issues. So we decided not to.

We decided to instead index everything on the clients themselves to not store it on the servers. And that gives us an inherently significantly more scalable system. Now, that said, that's still a backend. That's still a user's table. They're still are sending emails to deal. And all of the things that we've talked about, that's still the gentle archive process that I ran you through.

And so there are still bottlenecks there. It's just that with the architecture that we've chosen, we've pushed out when you would have to scale to significantly later. And this became readily apparent with me when I spoke to the old mailbox team. I moved on to building something new, and it's supercool. But they shared with me how when Mailbox launched, they didn't make these choices. They centralized everything. And it was one of the things that critically slowed them down, because they have to spend so much time and energy just trying to scale our service to keep up with demand.

[00:50:06] JM: Interesting. Well, I guess we can begin to wrap up. I want to thank you for coming on the show. It's been real pleasure talking to you. I'm trying to think if there's one more question I can ask, is Superhuman is something that a lot of people wanted to hear about. Here's random one. Any engineering difficulties in maintaining consistency across iOS, android and web? Do you use React Native?

[00:50:28] RV: We don't use React Native. No. So, almost all of the iOS app is built using Swift. There are a few areas of the app like quoting logic, dealing with the internals of HTML data itself. We are doing very complex string manipulation that is in fact written directly in C++. For our android app, we're using Kotlin. So, I think the most interesting answers to your question is – And I'm going to give your product management slant, because we started the iOS app a little bit later compared to desktop app. The desktop app had a two-year head start. So it's quite a lot more featurefull. It's only recently that they've sort of come into feature parity.

And we’d get asked this question a lot by customers who rise in and they say, “Hey, why haven't you built my pet feature?” which incidentally would take us a year or two to build, of course. And so I think of our team capacity now like a glass jar. Launch projects are big rocks, medium and small projects are pebbles of small sizes. And the most efficient way to fill that job is a combination of big rocks and small pebbles. First, you add in the big rocks and then pebbles can fit in and around them to fill the job. But if you were to add the pebbles first, fewer big rocks would fit in. But if you only added big rocks, then there would be unused space.

So what we do is each quarter we identify one massive rock and a few medium-sized stones and a large number of small pebbles. We know roughly the order we’re going to do them. But what the small pebbles give us is the opportunity to be flexible with the plan in the face of unknown [inaudible 00:52:03]. Let's say we run into an API issue, which I’m sure everyone listening has. Or there's a scaling problem which we've all dealt with. Or something else, maybe a key engineer leaves the team. So something happens that throws a wrench in our otherwise beautifully designed plan. Well, then what we can do is we can move those small pebbles around and still ship with high-cadence, high-quality, high-reliability despite the headwinds and despite the changes.

[00:52:29] JM: Well, that sounds like a good place to close. Thanks you for coming on the show, Rahul.

[00:52:32] RV: Well, thank you so much for having me.

[END]