EPISODE 678

[INTRODUCTION]

[0:00:00.3] JM: Front is a shared inbox application that has seen rapid adoption within companies. Front allows multiple members of a company to collaborate together on a conversation, whether that conversation is an e-mail, it's a Twitter message, or a Facebook messenger message. This is useful when a customer e-mail, or other message needs to be shared between the sales and the engineering teams, or when a single e-mail address is shared between different members of the same team, such as contact at softwareengineeringdaily.com.

The contact at company domain inbox is often completely flooded, but that doesn't mean it's lacking interesting, or sensitive e-mails. This might sound like a niche problem solving shared e-mail inboxes, but it's actually a problem faced somewhere within every company, because the problem of shared inbox is really prevalent.

Front has grown its user base really quickly, scaling its team as well as its infrastructure. The sensitivity of the data, e-mails, that front is handling means that security is paramount. As users of front rely on it more and more as a central point of communication, uptime and consistency at Front needs to be maintained.

Laurent Perrin is the CTO at Front and he joins the show to describe the software architecture and the product strategy for Front. It was a fascinating show and we covered the full stack on the backend Front pulls e-mails into S3 buckets and maintains the schema of that inbox in a SQL database. The desktop front client is written in electron, which is a way to write desktop applications in HTML5, JavaScript and CSS.

We also talked about the system for keeping the communications feeling real-time. It's important that users are aware of what each other is doing, because you don't want to be preparing a response to an e-mail to a customer at the same time that I'm preparing a response, and then maybe we both send the e-mail at the same time and it looks really unprofessional and strange. It needs to maintain that real-time nature and we talked about that as well.

Before we get to the show, I want to mention that we are looking for a few different kinds of roles and you can find those roles at softwareengineeringdaily.com/jobs. We're looking for writers. We are looking for people to help us with operations and we're also looking for potentially podcasters. The bar for podcasters will be very high, but I do want to open up some applications and see who out there is interested in podcasting about software engineering. Please apply to those jobs if you're interested. I'd love to hear from you.

[SPONSOR MESSAGE]

[0:02:56.6] JM Leap.ai matches you with high-quality career opportunities. You are more than just your skills and a job description and a resume; these things can't fully capture who you are. Leap.ai looks beyond these details to attempt to match you with just the right opportunities. You can see it for yourself at leap.ai/sedaily.

Searching for a job is frustrating and Leap tries to reduce the job search from an endless amount of hours, days, weeks, to as little as 30 seconds trying to get you matched to a job instantly, by signing up based on your interests, your skills and your passions. Leap works with top companies in the Bay Area, Unicorns and Baby Unicorns; just to name a few; Zoom, Uber, pony.ai, Cloudera, Malwarebytes and Evernote.

With Leap, you are guaranteed high response and interview rates. You can save time by getting direct referrals and guaranteed interviews through Leap.ai. Get matched to jobs based on your interests, your skills and your passions instantly when you sign up. Try it today by going to leap.ai/sedaily. You would also support Software Engineering Daily while looking for a job. Go to leap.ai/sedaily. Thank you to Leap.ai.

[INTERVIEW]

[0:04:35.8] JM: Laurent Perrin, you are the CTO and Co-founder of Front. Welcome to Software Engineering Daily.

[0:04:41.1] LP: Hey, thanks for having me.

[0:04:42.7] JM: Front is a shared inbox app, so it integrates with e-mail and other various communication tools. What problems does Front solve for the companies and the people that use it?

[0:04:54.8] LP: Basically, we try to solve external communication. When we started it, was really the moment when everyone was starting to get fed up with e-mail and we realized that most of our friends were miserable at work, because of e-mail. At that time, lots of new products were released, the new e-mail clients, but either they were built for individual users so they looked nicer, but that didn't really solve a new problem for teams. Or they were so different that people could not adopt them. They did not integrate with legacy and systems. We decided to give it a shot and build a modern communication software for teams that happens to integrate with e-mail really well.

What we solve basically is that we allow you to effectively work on asynchronous communication as a team, so we're going to sync with legacy systems, but threads are going to live in a global repo for your team, where information can be private by default, but is always easily shareable with your team without creating additional copies.

For example, a scenario where you have a customer, you're having a sales discussion with them and suddenly becomes a support discussion, it's something that you can solve very easily in Front. Typically, you have conversations that will involve a growing number of people within the company. Normally you'd have to juggle between tools, or you'd have to use a complex workflow with e-mails. In Front, it's something that can happen very easily. You have one conversation, you can mention other people, you can reassign the conversation, you can share it with more members of the team.

[0:06:31.9] JM: An example that you gave was the process of a conversation around sales becoming a conversation around support. You might talking to somebody – you're talking to a potential customer and that customer says, “Well, do you support this integration with our electrical system?” Can you say to yourself, “Well, I'm a salesperson. I don't really know that,” and you want to loop in a support person who might have a more technical answer, but you don't necessarily want to start a new e-mail thread with them so you can have a commenting

functionality within Front in order to have an internal discussion around the answer to the customer.

[0:07:06.6] LP: Yes. That's right. Yeah. You have lots of tools, so you have super tools that already exist today. The problem is that these tools are built with the assumption that companies live in silo, so you have the super team that's going to do their own thing and the sales team that's going to do their own thing and maybe the partnership team. The reality today is that a lot of work happen in gray areas that involves several teams and each work with completely isolated tools or with e-mail that's going to eventually break down your company culture.

[0:07:38.8] JM: What are some ways that workflows within companies change as a result of Front?

[0:07:44.2] LP: The biggest thing is that in Front, you can have a team inboxes that are automatically shared with a number of people in the team. Every time this inbox received an e-mail, maybe 10 people can see it and can act on it. You can have private inboxes and what we see is that people move the greater share of their communication to team inboxes that are shared by default.

They realize that the reason why e-mail is a private channel, it's not because people want to work that way, it's because e-mail works that way. For example, when you join a new company by default, you have access to just ready known of the history of the company, even though you're the person that actually needs that information the most.

[0:08:27.2] JM: Okay. Well let's get into some of the engineering. Can you talk about the engineering stack for the first version of Front?

[0:08:35.5] LP: Interestingly enough, it still with the version that's in production. We've never really rewritten the system completely. It works mostly in Nodejs and we have a fairly classical backend. We store all messages, metadata in [inaudible 0:08:50.6] and the content of each e-mail is stored in a unique S3 document on AWS. Then we have a distributed architecture with lots of message queues. For example, if you send an e-mail that's going to leave in a message queue, if you receive an e-mail and we need to apply rules on that e-mail, that's going to also

process on the message queue. In the end, we have hundreds of different message queues that perform each individual actions that are required to make the product work.

[0:09:19.9] JM: You got MySQL storing all the metadata for every e-mail and you've got S3 storing the contents of the e-mail and you've got Message Queuing that is building the backend for the interactivity and the real-timeness. What are you using for Message Queuing? Are using a service? You're using Kafka?

[0:09:40.0] LP: No. We're using SQS. As much as we can, we try not to build things ourselves, so we rely a lot on AWS for that. Then we integrate with service for WebSockets that all ads stays in sync, because Font is also a real-time app. For example, if you start replying to an e-mail, people that can see the conversation will see that you are replying to that e-mail, so that you don't have two persons replying at once.

[0:10:05.8] JM: Yeah, that's an interesting set of challenges there with the real-timeness, where you really don't want there to be a conflict. If I'm writing an e-mail to reply to a customer, you don't want somebody else in the organization to be also writing a reply at the same time. You need to have the quick syncing of different people, Front client apps. What are some of the challenges around getting that real-time nature engineered properly?

[0:10:35.9] LP: Some of our customers, there are on the single instance of comp, you have 600 people on. It's organizations that receive multiple messages per second and it's like a message arrives and it's going to be automatically assigned to a group of people. You have really complex workflows that happen in real-time.

I guess, the biggest challenge in terms of engineering, it's not any specific challenge, it's that we have to be really good at a lot of different things. We have to be really good at content engineering, because we need to make an app that's really fast, even though it's really complex. We store format, like at Facebook scale, but we store great – we store billions of messages, so we also have to be pretty good at that.

Also, Front it's the major work tool of our customers. If Front is not available, our customers cannot do anything. We have to build that in a really reliable way. To give you maybe a more

specific example about the problems we solve, maybe one example is our counting infrastructure. In a typical e-mail trends, you can think of context or something fairly simple. It's like you receive an e-mail, it increases by one, you read an e-mail, it decreases by one.

In front, it's vastly more complex, because you're processing counters for hundreds of people, you can see folders of other people, but every time you see slightly different versions of this counters because you might have different permissions. If you have a hundred users, you have – the order of magnitude is a hundred scale, so 10,000 counters that needs to be updated in real-time and that might all change multiple times per second. That has to be synced with hundreds of clients in real-time.

[0:12:16.3] JM: Is it counters?

[0:12:17.8] LP: Yes.

[0:12:19.1] JM: What do you mean by counter? What's an example of a counter?

[0:12:22.5] LP: In Gmail, next to your inbox, you have a number that tells you how many unread e-mails you have right now.

[0:12:28.2] JM: Okay. That's going to vary from different people depending on the privacy settings, the permission settings of different e-mails?

[0:12:37.0] LP: Yes. For example, you could say that I have access to these five inboxes and you have access to three of your inboxes, but I've manually given you access to a few conversations in inboxes that you normally contact us. The numbers you see have to accommodate that.

[0:12:53.4] JM: There was this issue with Facebook Messenger I remember a while ago. When I was talking to people about ReactJS, in the early days of ReactJS, there are some Facebook engineers that came on the show to talk about why ReactJS would – what it did in a novel way and one of the things that they would always bring up as an example was this issue with

Facebook Messenger, where it was hard to actually get your different Facebook Messenger clients to register the right number of messages that you had received.

You would log into Facebook Messenger and you would see a red indicator that said you had a message and you would click on it and you would go into it and look at it be like, “Well, I don't have a message.” There would be some problem with I guess, syncing all these different clients and making sure that something has gotten red, or it's gotten responded to. It sounds like what you're referring to there is a similar problem, where you have all these different clients that are interacting in different ways. All these different counters, like the number of messages that have been read, or need replying, or need – these can be addressed by different people. Keeping all these different counters in sync is not an easy problem.

[0:14:02.9] LP: Yes, exactly. I guess, we apply in the end the same types of patterns as React, but in this case we apply them on the backend, because the problem is that in the teaming box, someone else could mark this inbox as red, because the goal is not for everyone in an organization to read every single message, it's to make sure that the team as a whole is able to get everything done.

[0:14:24.5] JM: Front is a shared inbox. I have communications in my inbox that might be private, they might be public. Is it hard to train users to understand who has the rights to seeing a particular message and train them to understand that there's certain circumstances where you need to share a message, and other circumstances where you want to keep things more private? How do you change that behavior, because e-mail behavior is very hardwired for some people at this point?

[0:14:54.3] LP: Yes. I think it's the hardest problem we have to figure out in terms of product. The biggest problem as you said is that there are lots of habits that are sometimes bad habits. Even if you come up with a solution that you think is better, you have to reeducate people. The way we solve this problem is that first, we really position the product as a product to solve shared inboxes.

We worked on this problem of teams that are receiving messages that are not meant for individuals, like your support inbox, or your contact inbox. It was so painful to solve it with

traditional e-mail that people who are willing to spend time and understand how a product might work differently.

Then as the company and the product expanded, we eventually like, we built the ability to have your private inboxes. For example, my Laurent at contact inbox is something that that leaves and come today. The reason we did it is that even though it starts as something private, it often evolves into something that involves the rest of the team. One thing that happens very often is that someone I've met would reach out to me directly and say, “Oh, we've met at this place. I'd like to apply for a job at Front, and then immediately I want to loop in the recruiting team. What I'm going to do is that I can simply move that conversation from my private inbox to the recruiting team inbox and so the need to share the conversation.

[0:16:17.6] JM: You said bad behavior. What are some of the other anti-patterns that people have developed, because e-mail – they don't even realize that e-mail is not a sufficient tool for the kinds of communications that we'd have?

[0:16:31.5] LP: It may solve everything by creating a copy. I guess, the [inaudible 0:16:35.8] is on police forwarding. In Front, you have the ability to loop in people. Instead of creating a copy that's never going to be in sync, you can just invite someone to this conversation. It doesn't mean that you're sharing your entire inbox, it just means that you're sharing that one thread with someone from your team.

There's also the fact that in e-mail, you have a single channel. If you want to – like you have an important message that you need to reply to, you have to – so you're going to forward it with some comments and eventually someone will reply to your external contact. They will forget to remove your private comments and that could be sent and you would feel miserable. In front, we have private comments and you know that private comments are always going to stay inside of Front. They're never going to be sent to someone outside of the organization.

[SPONSOR MESSAGE]

[0:17:33.2] JM: Data holds an incredible amount of value, but extracting value from data is difficult, especially for non-technical, non-analyst users. As software builders, you have a unique

opportunity to unlock the value of data to users through your product or service. Jaspersoft offers embeddable reports, dashboards and data visualizations that developers love. Give users intuitive access to data in the ideal place for them to take action within your application.

To check out Jaspersoft, go to softwareengineeringdaily.com/jaspersoft and find out how easy it is to embed reporting and analytics into your application. Jaspersoft is great for admin dashboards, or for helping your customers make data-driven decisions within your product, because it's not just your company that wants analytics, it's also your customers that want analytics.

Jaspersoft is made by TIBCO, the software company with two decades of experience in analytics and event processing. In a recent episode of Software Engineering Daily, we discussed the past, present and future of TIBCO, as well as the development of Jaspersoft. In the meantime, check out Jaspersoft for yourself at softwareengineeringdaily.com/jaspersoft. Thanks to Jaspersoft for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:19:06.2] JM: What are some other unique technical concerns? You're building an e-mail client, what are some particular challenges that are different from a more – because a lot of the companies that I interview are building a SaaS web app and they all have their unique challenges, but a shared inbox, an e-mail client, that's a little bit different than a lot of the SaaS companies that I interview. What are some other ways that your challenges are unique, with the particularly sensitive area of e-mail communications?

[0:19:36.4] LP: Another big issue is that as you said, we store a lot of data for our customers. For some companies, we have hundreds of gigabytes. We are constantly improving the product, and at the same time, you need to make sure that data is secure. You need to have a strong engineering processes to make sure that you don't introduce vulnerabilities. In the end, an e-mail client is a piece of software that runs the HTML that was generated by who knows who, that might contain [inaudible 0:20:06.6].

There are lots of ways in which your product could break down and you need to assemble a fairly good team to make sure that you have tests everywhere, so that your product is safe, secure and your customer stays happy.

[0:20:22.0] JM: To what degree can you leverage the security guarantees of Gmail, or Office 365? I mean, these systems have been building spam detection and virus detection and so on for a really long time. Can you leverage that at all, or do you have to rebuild a lot of that?

[0:20:41.4] LP: If Gmail gives us an indication that something is dangerous, we'll use that information, but we cannot rely on someone else to do that job for us.

[0:20:53.3] JM: What's an example of – do you have to write spam detection? You have to write virus scanners?

[0:21:00.2] LP: No. For example, one thing that happens is that you receive a message and the name of that message matches the name of someone from your organization, but the e-mail address actually doesn't come from your organization. In that case, since we know exactly who belongs to your organization, we can tell you this person is not from your organization.

The problem is that if you build something like heuristics to flag with spam, it's a system that's going to work, but there's always a risk. What we try to build is things when we can objectively say that there is a risk here, we're going to warn you. If it's something that has to be trained, today we're going to rely on partners, like Mimecast, or Gmail to make that detection.

[0:21:46.4] JM: Memcast. What's Memcast?

[0:21:47.5] LP: Mimecast. It's something that would capture every e-mail that is sent or received in your organization. Then for example, if you realize that you've received a phishing e-mail, you can actually say, "Okay, who clicked on that link in my organization?"

[0:22:01.3] JM: Okay. You have talked a little bit about the backend infrastructure, the fact that a user signs up, you've got MySQL database that can get the metadata of their e-mails, you've got S3 bucket that stores the contents of the e-mail and you've got some queueing

infrastructure. Can you give you a little bit more detail on that core backend infrastructure, what happens when a new user or a new company signs up?

[0:22:27.9] LP: Yes. An interesting things that we have several data centers in multiple regions. When you sign up, we are going to check, okay what is the closest location available and we're going to pin your data to that location. Usually, there are local regulations. Meaning that companies, for example European companies do not form their data to ever go outside of the European Union. We have cases where a company wants to relocate to another data center and we have systems, so that we can seamlessly move your data to another place without any downtime.

Then what do you mean exactly what happens after? We create an account and then, so there is a system so that you can connect your existing e-mail accounts, or for example, SMS phone number in Front and then you can start using Front as your menu maintenance.

[0:23:19.5] JM: Right. I guess, I was just referring more to the backend, what else needs to get spun up and I guess what the schema for those different – you mentioned you're storing this stuff in S3, so there's like a new bucket get created for this new company and this a new queueing channel gets set up for handling all the messaging. I'm just wondering how the scale up partitioning is set up.

[0:23:45.0] LP: Yes. Each region, we have MySQL chart. We're going to again ping you – so today, the way our architecture works is that we have multiple regions, multiple charts and today, a customer lives inside of only one chart. We will get eventually to a point where a customer has to leave in multiple charts at the same time, because they are too large. That hasn't happened yet, so it's not a problem we solve today.

What we do solve is that we had the ability after the fact to rebalance our chart and to move customers between charts. It's something that we do all the time and we are constantly moving data between data centers.

[0:24:23.9] JM: Cool. This is MySQL on managed AWS service?

[0:24:29.7] LP: Yes. It might be tempting to say like, “Oh, I'm going to manage everything myself,” but it's actually hard to be absolutely sure that you're going to do it better than AWS. It's also how to convince your customers. Doing it better than AWS is one thing and convincing your customers that you can do it better than AWS is another thing. I think that until we reach a pretty big scale, it's pretty important to architect your service that you can leverage the experience of actors like AWS.

[0:25:00.8] JM: Well certain, if I knew that Front was using their own bucket storage solution instead of S3, or Google's bucket storage, or Azure bucket story I would probably not feel comfortable putting my e-mails there. What are some of the particular thing – when you can use a managed database service like AWS. what is it Aurora? Is that the managed MySQL?

[0:25:26.2] LP: We tried with all, but it doesn't work well with our workloads. We stayed on Vanilla MySQL.

[0:25:33.2] JM: Okay. Oh, vanilla MySQL. Okay, so are there services that make it easier to do that sharding process when you can move these different shards between different data centers and so on, is that taken care of by AWS, or is it have to be manually programmed?

[0:25:49.0] LP: It's something that we built manually, but it ended up being easier than what we thought it be. NodeJS is actually very, very good for these types of tests. We have a system that's going to open up – it's a script, it's going to connect to the server where the customer’s data lives and it's going to pretend it's a new MySQL replica. It's going to get a live feed of all the changes that are applied to that server and we have a simple system that extracts all the updates that confirm that customer. It's going to replicate in real-time these updates to the new server.

Then the script opens a second connection and copies all the existing data. At the end, we reach a state where the customer is kept in sync between the two data centers. We have an automatic system where we can say at some point, “Okay, all of the new rights are going to land on the new server and once we detect that there are no new rights on the back server, on the old server, we flip a switch and the customer is relocated.” This system allows us to – we have customers with a lot of data and we can automatically relocate them between data centers.

[0:27:03.2] JM: Since we're on the subject of services and backend engineering, you're migrating the Kubernetes right now. How is that going?

[0:27:12.4] LP: It's going well. Actually this morning, we have trainings to make sure that all the on-call teams know how to operate the cluster. I guess for us, we want to eliminate risks, and so having tool systems is always worries, because you don't know what you're relying on. It's also a way to be more compliant and to make sure that you do not have to give SSH access to a lot of engineers without any accountability.

You can limit what people can do, like they can only access logs, but they cannot physically work on the servers. It's also like in terms of costs, it's easier to start and stop new services, so it will help us reduce cost.

[0:27:56.0] JM: Why does Kubernetes make that permissioning easier?

[0:27:58.7] LP: Today, we use Chef. Chef is going to always reinstall the same instances, but in order to troubleshoot some things we have a group of about 10 engineers that can SSH through a server. We have SSH logs, but it's how to edit them. It's how to say for sure that you know exactly what everyone has done on every server. Since we use always the same containers, you don't know that maybe someone made a manual operation on one server one day and that server is functioning today because of that one operation. With Kubernetes, you are always going to start with a clean slate, which means that you know that you're not relying on something that that someone changed outside of Chef control.

[0:28:43.8] JM: Are you using one of these managed Kubernetes services?

[0:28:47.5] JM: We wanted to use AWS managed ELK services, but there were features that were missing. In the end, we decided to roll out our own deployment.

[0:28:58.2] JM: It sounds like auditability and having logs that can ensure that the security of your data has not been compromised, it sounds like that's really, really important to you.

[0:29:11.3] LP: Yes. At the same time, you're inventing a new product. You start with e-mail and how it works individually. Then you try to think, “Okay, if I expand it to include teamwork workflows, how does it change?” It turns out that a lot of things change. It's a challenge on the product and it's also a challenge in terms of engineering, because you cannot rebuild your apps from scratch every week. You need to architect your product in a way that the new things you're going to learn about the product are things that you're going to be able to bake in your existing platform. That's one big area. The second area is that as you do that, you need to make sure that your system is secure.

[0:29:55.9] JM: In terms of auditability, do you back up all of your logs, like how often do you roll your logs? Do you have some append-only system where you could always be able to look up at what point in history somebody has access to system?

[0:30:10.7] LP: Yes. Yeah. We have that for – I think we can go back – at least, we have a [inaudible 0:30:17.0] cluster to access basically all the information that we need to access. We have a year of data that's accessible immediately and the rest is in its free buckets. If we need to pull up something that happened two years ago, we'd have to reread the data, but it's still available.

[0:30:34.5] JM: You've got people communicating over e-mail, over Facebook Messenger, over Twitter, I think. Do you have a schema that aligns these different messaging platforms, or do you just keep them totally disjoint?

[0:30:50.0] LP: What we've chosen to do is I think, 95% of our code base is message agnostic. It doesn't care if it's handling Twitter messages, or e-mails. We’ve tried to keep the e-mail path through a very tiny set of our code that’s really channel dependent. In the end, there are differences, but we try to keep them in very key areas of our code. For example, one big difference that's very annoying to us is that in terms of the life cycle of messages, the idea of messages are not always set at the same time. When you set an e-mail, the client actually sets the ID of the e-mail, but when it's a twitch, it's actually decided by Twitter themselves.

It's a bit different for every type of channel. The idea that your ID is going to set at different times in your life cycle is something that when you build your architecture, if you wanted to take an

existing system and then say, “Okay, then no I want to add chat messages, or I want to add SMS,” it can actually be a lot harder than it seems, because of simple problems like that.

[0:31:56.9] JM: Do you solve that by having an internal ID and then an external ID also?

[0:32:02.6] LP: Yes, but I guess the more important thing is that very early in the product development, we have added a second channel just to make sure that we could do it. At that point, adding a third type of channel was a lot easier. When we started the product about four and a half years ago, so once the main use case of e-mail was working, we added Twitter as soon as we could to prove ourselves that and to make sure that our architecture could accommodate different types of message types.

[0:32:33.4] JM: Are there any other challenges that come from that heterogeneity, from the fact that you have Twitter and Facebook and all these other messaging thingies in the same app?

[0:32:43.6] LP: Yes. For example, we see both like about a dozen of channel types today and there are no two systems that work the same. For example for Facebook, you can reply but you cannot compose, because that's not something that the Facebook API allows.

[0:33:01.2] JM: You can reply, but you can't compose. What does that mean?

[0:33:05.0] LP: If a user initiates a conversation with you, you can reply to them, but you cannot on your own initiate a conversation with the user, because Facebook doesn't want people to use their API to spam people. That's not something you can do with API access.

In your clients, you have to remember that when you compose, you have to exclude Facebook messages. You have channels where you can reply to one customer only. You have channels where you can compose to multiple people at people at once. Every channel is a bit different. In the end, you have to find a common ground so that you can describe all of these channels in a single product. If you can do it, it's very useful because we constantly have things where people will call us out on Twitter and then a tweet becomes a support discussion.

Then if that person says something really nice about us, we want to loop in the marketing team, because maybe that's something we'd want to post on our homepage. That's just for us. We constantly see our customers do amazing things, because they can have a complete picture of their external communication in one place.

[SPONSOR MESSAGE]

[0:34:22.5] JM: When a bug occurs on your website, LogRocket captures the user behavior and allows you to do an instant replay to see how the user responded to the bug. LogRocket lets you replay what users do on your site, helping to reproduce bugs and fix issues faster. See issues as if they happen in your own browser with a full video replay and get those issues fixed fast to maintain the health of your application and keep customers happy.

LogRocket works regardless of your applications language, or framework and it provides SDKs for specific technologies. You can easily integrate with the tools that you already use. You can check out a demo at logrocket.com/sedaily, which would also support Software Engineering Daily.

LogRocket also records console logs, JavaScript errors, stack traces, network requests and responses with headers and bodies, browser metadata and custom logs. If you're triaging back-end errors, it can be unclear why the front-end made an unexpected request. LogRocket integrates with back-end logging and error reporting tools to show you the corresponding frontend session logs for every back-end error and log entry. Quickly understand your bugs and fix them with LogRocket.

Go to logrocket.com/sedaily to find out more to see a demo and to support Software Engineering Daily. Thank you to LogRocket for being a sponsor.

[INTERVIEW CONTINUED]

[0:36:05.9] JM: I want to talk about the front-end a little bit more. The app is real-time and it's a desktop app that's HTML5, JavaScript, CSS, this is an electron app. What does the architecture on the front-end look like?

[0:36:22.4] LP: We started initially with angular one, which was ultimately the best framework at the time. This year, we've been rewriting everything in React. I think that, so the important thing was to find patterns that could be reused by the team so that new people could contribute to the user base, to the code base, sorry.

I don't know how I would describe the architecture of the app. I guess, what we have today is we've retried to build something that's state of the art. For example, in terms of CSS, we actually use a very, very small subset of the CSS features. We've rebuilt everything to use CSS grid. We have one way of positioning components and only one way. We use [inaudible 0:37:09.4] chain a system so that we build components that are completely reusable, so they do not depend on CSS that's outside of them.

Basically, if you include a component somewhere, this component is always going to look the same way. If you want it to look differently in some context, you need to add a system to configure that component, but that logic we always live inside of the component. Does it make sense?

[0:37:34.1] JM: It does. Yeah. The first thing you mentioned when you start talking about the front-end was the fact that you wanted to make the front-end more approachable to new developers that were coming in to work within Front. Did you have to do some refactoring to standardize the different API surfaces that – or add comments, or something, breakable monolith perhaps?

[0:37:57.1] LP: Yeah. What we found is that – we had a code base that we thought was pretty good and seasoned developers were able to iterate quite quickly on it, but we realized that as the engineering team grew, it became harder and harder for new people to understand what were they breaking. Even if you have a test Twitter, it's how to as a newcomer, to gain the confidence that you're not breaking something.

For example, for a very long time we prevented ourselves from rewriting the app, because you have to deliver new value to your customers. If you spend a few months rewriting your app, you're not doing that. At some point last year, we said, "Okay, it's been more than four years. It's

the right time to do it." One of the big changes that we made is that we rewrote everything in Typescript. Even though it's JavaScript, you have type safety. That means that when you're looking at a symbol somewhere, you can know exactly where is it used, how is it used. It also means that your build tool chain can do tree shaking, it can be more efficient at minification.

As your team which is a certain size, it's really important to think of you have components that were started by an engineer that grow in size and that are now going to be owned by another team, and it's important that that team has complete ownership of the code. If they feel it's been written by someone else and it's just their job to maintain it, you're not going to be successful.

[0:39:30.0] JM: It's like the same thing that Slack – when I talked to Slack about using Typescript, they say basically the same thing.

[0:39:36.1] LP: Yeah.

[0:39:37.2] JM: I mean, they have basically a Typescript electron app, very similar. Why is electron useful? Why do you see companies like Front and like Slack building these electron apps, instead of just having people open a new browser tab?

[0:39:52.8] LP: It's fun, because if you look at – I think that users likely love to complain about electron apps, because let's face it, it's a bit heavier. It means that you're going to have basically a new browser that's running with slightly different libraries, which means that it's going to – it cannot reuse the same in-memory library, so it's going to consume more memories. In the beginning, we tried really hard not to do it. There are two things that you realize. The first one is that if you are just living inside of a browser, you do not have the same engagement, because people I'm not going to go back to your app as easily. If it's their walk tool, it's a lot harder to use.

At Front, we can say that if you're using Front all day longer, you'll absolutely want to use the desktop app. If you're just using it one hour a day, then it's okay to live in a browser. We see the two usage patterns. Then in terms of why use electrons specifically, it's really, really hard to make an app that works in all sorts of environments. Electron is very good at insulating you against the OS and everything that might be different.

For a long time on Mac, we would not have an electron app. We would just have a regular app with a WebKit webview. We constantly had customers complaining about some niche features being broken, because they had a slightly different version of WebKit, with slightly different bugs. With electron, you are able to ship one version that you control. You're able to have QA and make sure that everything works and you're able to make sure that people are not going to have real bugs.

[0:41:30.7] JM: How much code reuse can you have if you want to take an electron app and make a mobile app out of it?

[0:41:35.9] LP: It depends. In our case, so since want disposition also against other e-mail clients, when we started, it made sense to have a complete native apps. Today, we have an iOS app that's fully native and we have an Android app that's also fully native. If we were to do it today, I think we'd use React Native and most of our code could be reused.

[0:41:56.4] JM: When you said that, I was just thinking about that Airbnb post about React Native. Airbnb decided to move off of React Native recently, because I guess of several different issues with it, but that's a different conversation.

[0:42:09.9] LP: Yeah. I guess in our case, so when you sell to our customers are not just, so you can value companies. Actually, most of our customers come from traditional industries and they don't always have the latest phones, or the latest desktop apps. If you want to really make something useful, that's not going to be marginally better than what was before. You have to target legacy industries and you have to admit that sometimes you're going to work on devices that are not the fastest. It's something to keep in mind. That's also why for mobile devices we chose to have a native apps.

[0:42:46.4] JM: I agree with that approach. What about the operational side of things? How do you do continuous integration, continuous delivery, staging environments, testing on all these different mobile client surfaces, what's your approach there?

[0:43:03.6] LP: We've automated everything we can. I guess, our approach is to make sure that we never release a change that we do not understand. That means if I go back a little bit, so we have continuous integration. If you match to the production branch, everything restarts, every change has to go through a code review and you have to answer a checklist. The most important thing in that checklist is if it breaks, how bad it can be? Second, if it breaks how do we roll back?

If we have a good answer to these two questions, we can start carrying your change to a growing number of users. As long as we know that if we have feedback that something is broken, we can quickly roll back, it's okay. Then as much as possible, we'll first released the changes to our own team who is using Front heavily. If they don't find a problem with it, probably it's fine.

[0:43:56.4] JM: What about product and engineering? Front is a really well-designed product, nice work there. How do you manage all of the different features and integrations while avoiding the interface getting too busy and overwhelming?

[0:44:15.5] LP: Yeah. Like I said, we use our own product. Every engineer uses Font all the time. Every person at Front uses Front all the time, so we're all way too opinionated about things we want to see and things we don't want to see in the product. Then another thing that helped us a lot is that Mathilde, my co-founder and I, for a very long time we had a complete view of the company just the two of us. She would understand, like she would talk to customers, she would have a full grasp of who we are selling to. On the other side, I would know in fairly minor details everything about the product.

The two of us, we've a few senior employees in the company. We could make a lot of decisions and we could sometimes decide, okay everyone is asking for this, so let's build it and recognize that we are actually getting this feedback from several customers. In some other times, going in the direction that no one was asking for, but we really believed was the right direction.

[0:45:13.5] JM: Between the two of you, you and Mathilde, she could hold all the product conversations in her head and all the customer conversations in her head, and then she could have a conversation with you and you could know how those things might translate to different

engineering decisions. Between the two of you, you could weigh the different trade-offs between decisions.

[0:45:31.9] LP: Yeah. It wouldn't be just the two of us, but it would be a fairly small group of people that had the ability to make a very impactful decisions about the product.

[0:45:40.6] JM: How have you scaled up that process now there's no longer the two of you? How this product interact with engineering today?

[0:45:46.6] LP: We have a product team today. I guess, one thing we've retained is that we realize that engineering requires a lot of creativity. If you have a product team that craft specifications that are extremely accurate, since you're working on paper, you cannot know exactly know all the edge cases. If it's too detailed, the engineer that's eventually going to build a future loses their creativity and you work with blinders. In the end, you can build something that doesn't make sense without realizing it.

We try to have a lightweight specs with the understanding that the people who is building the future will actually contribute a lot to hold that feature works. Then sometimes we mess up, so we realize that when the feature is built, we will discover something we hadn't planned. In that case, you have to admit that okay, we have to go back to the drawing board. This feature is going to be late, but it's better to have something that really works, than something that's not great.

At the same time, sometimes we know that we have the beginning of something great, but it's not complete. In that case, we'll still visit just to see how people react to it. If we know that we're going in the right direction, but it's not all that it could do it okay, we can release it, if we know that it's not the right direction, then we go back to the drawing board.

[0:47:08.8] JM: I started my podcast about three and a half years ago at this point and I built most of it on Google Drive, Gmail, Google sheets, the office suite of things from Google. Now there are all these next-generation tools; there's Front, there's Notion there's Air Table, there's Slack. I don't know if I should be migrating to these, or just if I should use them because they're

cool. Do you see these new productivity tools is replacing the classic office productivity tools, or do you see them as complementary? How do you see that collaboration world changing?

[0:47:47.4] LP: I think that eventually, it's not necessarily that these tools will replace the existing tools. It's like everyone will we move to a closer, to a SaaS model, because it's a lot easier for organizations to manage these tools. At the same time today, if you want to use all the cool tools that means using 40 different tools and constantly switching between them. I expect that there will be a consolidation that will happen around a small number of platforms.

[0:48:17.4] JM: This seems like they all have really good economics, so they all make good money and there's not much reason for them to merge with one another.

[0:48:24.1] LP: No, but it means that you will have systems of records. Front could be one of them, maybe Salesforce. You have a lot of good candidates. As long as the source of the data comes from these products, eventually all the other products I think will become plugins in a small number of platforms.

[0:48:44.1] JM: Yeah, okay. I know we're up against time. I want to ask you a little bit about Slack. How do you see people using Front versus Slack? Do you guys use Slack internally at Front, as well as Front?

[0:48:56.9] LP: Yes. We use both products. I guess for us, so there are two divides; one of them is that we use Front for external communication and Slack for internal communication, but that's a bit of an artificial divide. Slack could decide to build more external stuff and we could decide to build more internal stuff. The biggest divide is between synchronous and asynchronous. If someone pings you on Slack, you are expected to reply right away, or not reply at all. Front on the other hand is really meant to be asynchronous. Since you can move conversations around, you can move them, you can escalate them to another inbox, you can reassign them, you can snooze them, which means that it's okay to – if that person does not reply right away, there's a good chance that they will eventually reply within the day or tomorrow. That's not something you could do on Slack.

We've experimented with new asynchronous workflows in Front. Today, we've decided to stay away from them, because we think – actually, I'd rather do this in Slack than do it in a place where I mix synchronous and asynchronous stuff.

[0:50:03.6] JM: Yeah, synchronization versus asynchronous. Okay, last question. There's a recent update to Gmail where machine learning is getting into Gmail more and more and it seems like Gmail is getting more and more acquainted with who I am and how I want to interact. Do you have a data science team and how are you thinking about machine learning? What are the opportunities for machine learning in Front?

[0:50:24.5] LP: What we think today is that it's hard to see in which direction data science is going to do – what you can do with machine learning. We've invested on our platform. On providing API, so that whatever we are going to build on Front in the future could be built by someone else.

Even today, we already have teams that are building cool integrations. For example, there was a product called maya.ai, and they built a bot platform, so you can have over messenger I think a bot. When that bot doesn't know what to reply, they'll actually call in a human from Front that can take over the conversation to handle the complex question asked by a human, and then hand it back to you to a bot.

For example, we have customers that has built extensions on Front, where when you receive a message they'll try to guess what the answer should be. It's not something that is built inside of Front, but it's things that some customers built on top of Front.

[0:51:21.7] JM: Very cool. Well Laurent, I know you got a lot to do and I appreciate you making the time to come on Software Engineering Daily. It's been really fun talking to you.
[0:51:28.7] LP: Thanks as well. It was really fun as well. Have a good day.

[END OF INTERVIEW]

[0:51:34.7] JM: DigitalOcean is a reliable, easy-to-use cloud provider. I've used DigitalOcean for years, whenever I want to get an application off the ground quickly. I've always loved the focus

on user experience, the great documentation and the simple user interface. More and more, people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU-optimized droplets perfect for highly active frontend servers, or CICD workloads.

Running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance. The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily. As a bonus to our listeners, you will get a $100 in credit to use over 60 days. That's a lot of money to experiment with.

You can make a $100 go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure and that includes load balancers, object storage, DigitalOcean spaces is a great new product that provides object storage, and of course computation. Get your free $100 credit at do.co/sedaily.

Thanks to DigitalOcean for being a sponsor. The co-founder of DigitalOcean Moisey Uretsky was one of the first people I interviewed and his interview was really inspirational for me, so I've always thought of DigitalOcean as a pretty inspirational company. Thank you, DigitalOcean.

[END]