EPISODE 1489

[00:00:00] ANNOUNCER: Creating great creative tools is extremely difficult. There are thousands of parts a user could take, and every aspect of the user experience has to be carefully considered and optimized for performance. And when you try to add real-time collaboration on top of that, you can be sure that you are left with almost no time to focus on the core user experience of your product.

Liveblocks is a set of tools and APIs that help you create amazing real-time collaborative experiences. With Liveblocks, you can make anything collaborative in minutes so that you can give your core product the attention it deserves.

Guillaume Salles is the co-founder and CTO of Liveblocks and joins the show to discuss what he and his team have built.

[INTERVIEW]

[00:00:47] JM: Guillaume, welcome to the show.

[00:00:49] GS: Hey, thanks for having me.

[00:00:51] JM: You are working on Liveblocks, which is a system for multi-user collaboration. And I think there are a few good examples of this that I can think of. One is Figma, where multiple people can collaborate on a design document at the same time. Another is obviously Google Docs, where people can collaborate on a document at the same time. So, if you generalize this to multi-user collaboration more broadly, I guess your thesis is that there are enough use cases in the span of potential products that you would want to make APIs and abstractions for multi-user collaboration. Is there something about modern software development that has opened the opportunity for more multi-user applications? Because if I think historically, there's not a ton of multi-user business applications.

[00:01:54] GS: Yes. So, that's a great point. So, like you said, Google Doc and Figma were the pioneers in like collaborative apps. Google came along. They took a lot of market share over

Microsoft Office at the time. Like, six or seven years ago, Figma came along and took a lot of market sharing in the design industry, if you compare it to Sketch, InVision and this kind of companies.

And now, we are seeing more and more app built around the same system. So, for example, Pitch is to create like slides, or Mirror. Basically, everything that you do in a non-remote world inside the room that you can have an app, collaborative app, to do that in a remote world.

So, I would say not a technical opportunity, but more the changing part [inaudible 00:02:43] where remote is becoming more and more like present in companies. And as soon as you need to basically edit data, it could be in finance. It could be make presentation tool. It could be in pretty much every industry. There is at least at some point in the creative process or even just the process where you need to collaborate.

And currently, there is a lot of app where it's a step-by-step process where you do your part and [inaudible 00:03:11] to someone. And after that, they edit the data. It's not necessarily a necessity. It's because the technology didn't really enable it. And we feel like it's going to be more and more like common in the future. It's not just about creating something like Figma. It's not just about collaborating. It's also being able to have the most fresh version of the data. And you can work offline and go back online and you will see the data without thinking about synchronization issue. And that makes sense?

[00:03:42] JM: Yeah, certainly. If I was to build – Like, I've made a game before. And my way of doing multi-user collaboration was just having two people writing to the same database and just pulling the database really aggressively. And that seems to work just fine. What's wrong with that approach?

[00:04:07] GS: There is nothing wrong with that approach. You can use like a database and you can create. But if you really want something to have it in real-time, like a game, for example, what kind of game did you build? I'm assuming it's not going to be something really intensive like an FPS –

[00:04:25] JM: It was super intensive. It was just like let's say a card game.

[00:04:28] GS: Yes. So, a card game, you can imagine that you can query and just use even Firebase. It's kind of real-time. It's not the best performance. But even if there is like one second latency, it's completely correct. But you could ask the same question to Figma. Why did it use like something already in real-time where you can see people drag a layer in real-time to see what they are doing?

If you have a latency that is close to what's happening in the physical world, like a few milliseconds, the UX is way better. You can understand the intent of what's happening. You can get the – It's like the non-verbal communication that you have in the real-world that start to happen when you have done a good performance.

So, it's not technical limitation. I think you can pretty much do pulling every five seconds for any apps. But more and more, it's becoming a commodity, and we are getting used to have that feel instant even if there is network involved.

[00:05:32] JM: Right. So, if I was to take that approach of just pulling and writing to a shared database, what are some synchronization issues that could emerge?

[00:05:46] GS: So, it would be – For example, if two people edit the same piece of data at the same time – And let's take Figma, for example. If Figma was built around one row in a database, it's a rectangle, a shape. And someone is moving it the position, the X and Y, and someone else is changing the background of the rectangle.

Usually, it's a single entity. You can have conflict. We are taking care of that. You can edit at the granular level a single property, and Liveblocks will solve the conflict to make sure that everyone will have the right rectangle with the right properties, colors, and it will be solved on every client. You can do pulling, but conflict would happen. It's always the last writer win in this context.

With Liveblocks, for example, a to-do list. If you want to insert data in a to-do list multiple items at the same time, the holdering has an impact. It's important. What's important to make sure that the order is the same on every client. So, even if you add until it go to the backend, you want to create instant locality that you add a new item to the list. But you will get a new version from the

backend like five seconds later if you do pulling. And you want all the tools that stay at the same place. It's the kind of conflict that we are solving with Liveblocks.

If you are familiar with [inaudible 00:07:10], it's the technology that is used by Google Doc and also Figma a little bit inspired by [inaudible 00:07:17] is the kind of technique that we are implementing. We are not the first one to do this kind of stuff. We are just reeling on the shoulders of giant people that did a lot of research and built similar apps.

[00:07:30] JM: So, if I think about the problems that come from real-time collaboration, if I try to generalize it, I think of network partitions as being a pretty big issue. Like, if I get separated from the network and somebody else makes a bunch of writes, and then I make a bunch of writes, and my writes conflict with their writes, when the network partition reconnects, then there's subjectivity as to how to resolve that network partition.

So, the way that you can resolve that – I mean, first of all like, you would have to have some kind of – We're thinking about how to deal with that. You'd have to have some kind of system for writing to local storage when there's a network partition. And then you have to have a system for resolving the merge conflicts when the network reconnects. So, are those the kinds of issues that you're handling with Liveblocks?

[00:08:44] GS: Yes, exactly. Currently, we don't support the full offline experience. We are supporting if you have like unstable connection and you disconnect for five minutes, we have a way to solve conflict optimistically. So, like, we will not let the user choose which version is the best right now. We basically store the operations in-memory. And when you go back online, we patch everything optimistically. Because we feel like if it's for a short period of time, it's okay to solve it like that. But we will pretty much use the same pattern when we will support the full offline experience. We will persist the operations to local storage. And when you go back online, if the data is important, you will have a choice as a developer to let the user merge the conflict manually. So, if we go back to a design tool like Figma, it would be, "Do you want to choose like this color of our rectangle or the color that is currently locally saved?"

[00:09:51] JM: Gotcha. So, at this point you defer to the user to resolve those kinds of conflicts.

[00:09:58] GS: Sorry. I was not – Maybe not explaining properly. Currently, let's say you are going offline. You edit a rectangle. You go back online and edit this rectangle. If it's the same property, you will win. So, it will be your local version that will override the backend. If it's different properties, so, there is not direct conflict, even if it's on the same entities, we will solve the conflict optimistically. In a future version of Liveblocks, we will let the user choose which version they want to keep.

[00:10:33] JM: Gotcha. And what about the local storage system? Are you just storing data to the browser's local storage in those kinds of situations? Or do you use a database?

[00:10:48] GS: Not right now, because we only support like flaky connections. So, when you are currently only offline for five minutes, for example, we don't persist anything. Everything is kept in-memory. So, we keep the operations what you did and we merged them when you go back online.

In a future version, yes, we will probably use IndexedDB to save the operations similar to what Figma is doing.

[00:11:11] JM: Use what database?

[00:11:12] GS: IndexedDB, I think. It's on the local storage.

[00:11:15] JM: And so, let's talk a little bit more about the kinds of problems that you need to solve building a real-time collaboration API. Can you describe some more of those issues?

[00:11:30] GS: So, while we talk about the frontend and the operations, I think what's it's extremely hard to build in a multiplayer environment is to have like a good undo/ redo system. So, if we take undo/redo in a traditional app, in a single player app, it's basically a time machine.

When you press undo, you go back to a version that was already on your laptop. Even when you do a redo, it's not something new. It's not a state that the user never saw. So, if you are working with React, it's fairly simple to read because you basically keep the state somewhere and you can just replace all states when you press undo.

When you are building multiplayer app, it doesn't work like that. Because the undo/redo is a local stack. So, let's say we are collaborating together on a design file. When I'm pressing undo, I'm doing only my changes, not yours, even if you did some changes to the state after me. So, we need like a lot of algorithm and to be able like to undo only changes. So, we need to have the reverse operation of anything that you can do on one of our data structure to be able to implement that. And you get it by default when using Liveblocks.

And the other thing more in the backend that is fairly complex to build is the scalability of WebSocket. So [inaudible 00:12:59] balancing. And usually you don't want – It's really simple when you just spin a Socket.IO and you have like a few users working together. But to make that scale with like thousands of users, it's pretty complex to split in different homes and when they connect/disconnect and assuring that there is like a ping system to make sure that even on like unstable connection you can reconnect easily. We are taking care of that. So, it's the kind of system that we are providing so our users can focus on the core product [inaudible 00:13:34] features.

[00:13:36] JM: So, have there been any engineering problems thus far that you found particularly difficult?

[00:13:43] GS: Yes, to be honest, building Liveblocks, I feel it's extremely complex. We try to be inspired by a lot of like research strategies and OT. I don't know if you're familiar with strategies and OT. But it's operational transformation, and strategy is what is used for solving conflict in a completely distributed system. Not centralized servers. And we try to find the right tradeoff between all this research to make something that people can use without providing their own backend. And it's scales well depending on the data structure and how you want to solve conflicts. So, I feel like it's pretty complex.

And also, not directly related to the conflict solving, it also provides a really good API that you can use directly in the frontend. And that works well with existing state management library. We don't want to be like just a socket provider that's solves conflict. We want to be as close as possible to the UI. At some point, we might also release components, series patterns that are similar in a lot of collaborative apps.

So, for example, the cursors, and the avatars that show on the top right and finding the exact right APIs so people can just integrate multiplayer features in minutes, it's extremely hard. We try to be close to React developers. And we are already preparing packages for like Vue and Svelte. I think providing a really good like full stack experience that goes from the UI to the scalability of the WebSocket, it's what makes life look unique.

[00:15:28] JM: So, you talked about CRDTs and operational transforms, the research primitives that help with distributed systems problems. Can you illustrate an engineering problem that you need to solve with those that you can't just solve with the more naive approach to user synchronization that we've been talking about?

[00:15:54] GS: Sure. So, let's imagine the most basic example that I have in mind is probably a counter. Let's say you want to build like a counter where you can increment a value locally. I can increment it. And when every operation that we did, like, our send and every client get all the operation made by other clients, we have the same value. It's not just as simple as saving like a number in a database.

If you like increment by three your counter, and I increment by five, once everything has been like propagated to other clients, we want to see eight. So, if it was like just a database where every time you click you like set the number, at the end it will be the last person that send the data that will have like five or three. It will be the last version. The last version of the counter. But it's not something that you want.

And so, by describing at the data structure level what kind of – How the conflict will be solved? You can like compose the data structure to build the state that you want. In the same way that we are not using array every time, sometimes we are using like map set, different kind of data structure depending on how you want to interact with them. With strategies, this data structure also encode metadata to how the conflict will be solved. This is the complexity that you will have with strategies that you can't really build with like a more traditional approach. That makes sense?

[00:17:41] JM: Totally. So, if you're trying to debug those kinds of issues, it seems like creating the race conditions to appropriately debug those sorts of things could be difficult. What's the process for debugging an intense distributed systems problem like that?

[00:18:03] GS: So, it's a great question. So, we do a lot of testing. And obviously, we do a lot of unit tests to like try to unit test integration test where you can define properly the order of operations. And you can even like simulate offline if your code base is already like well-structured.

But one thing that we do at Liveblocks that it's pretty cool to troubleshoot these issues, we are using like end-to-end first testing. I don't know if you're familiar with first testing. But basically, we have an app that goes through all the stack end-to-end. And we do a lot of random operations really fast. We spin like five clients. They all like edit the same piece of data in different ways. They all do undo/redo, goes offline, go back online. And in the end, we make sure that every client, we have the same state.

And because we do a lot of iterations, it helped us in the past discover a lot of subtle issues that we didn't found with unit test and traditional integration test. So, we do first testing [inaudible 00:19:11] all operations. If there is a bug, we can look at all the logs to understand what happened. And yeah, it's definitely hard to troubleshoot especially if it's in a client environment where we don't have all the logs. But we make sure that we have enough end-to-end tests to find the issues before they go to production.

[00:19:32] JM: Gotcha. So, now that we've talked a little bit about the basic use cases and some of the rough engineering, let's talk a little bit more about the broader vision for the product. So, you're trying to piece together some of the multi-user collaboration primitives. So, for example, shared presence, where you get the sense that two people are collaborating on the same document. Shared storage, where you can synchronize data. And you're also working on analytics.

Let's zoom out, and I'd like to get a sense for what are the other primitives that – If you think about this as like a large span of APIs that you could potentially build, a large span of products, how many primitives for multi-user collaboration are there?

[00:20:33] GS: So, there is also the first thing that comes to my mind and then a lot of our customers' mind is to have a chat and a comment system. And so, instead of using the storage or the person to build your own comment system or own chat, we could deliver like chat components that you don't have to think it's always working real-time. It's of the conflicts. And you get notifications.

As soon as there is collaboration, there is some kind of external communication necessary. Or because it's communication, it's not necessarily like a chat. It could be also the audio or video, like a Zoom.

Currently, to collaborate, most of the time we open Zoom and we show our screen and we interact on an app when there is no multiplayer capability like Figma. I think in the future, we could have like a way better UX if you could start like an audio call or a video call directly inside an app. Because you get more context. You see where the people are in the screen. And they can interact with the data.

So, I think more and more this kind of multiplayer features will be 99% of applications. And it's one way that Liveblocks loss could evolve toward. But we could also work more vertically on our existing features. Currently, we are providing like APIs and packages for JavaScript for the web. But we could also like build a client for Swift or whatever like native platform where there is. And, yeah, there is a lot of different ways that Liveblocks will evolve. The vision is to try to make the web more collaborative. But we will adapt based on the market.

[00:22:24] JM: So, there have been a lot of APIS for – I think like kind of the predecessor to what you're working on with Liveblocks is something like PubNub, where they focus on basic pub/sub and they provide some real-time stuff like location. Or they do some chat APIs. Do you feel like is there some way you can engineer differently? Or maybe like if you think about the product marketing approach, maybe you start with these collaboration primitives and people will naturally use your APIs over competitors. I guess, if you have product overlap with other companies, how do you expect the competitive positioning to look?

[00:23:23] GS: It's a great question. So, to have done that in the past in previous companies, usually you have to use like a WebSocket infrastructure like Pusher if you want to rely on external services, or Ably, and in combination with Twilio. And you need to build your own chat and you need to wire everything together and generate it in your local stack.

With Liveblocks, we try to really be like a tool belt. Everything that is related to collaboration, we want to provide like basic primitive to really solve every use cases without going too deep at first. It's really to not have to wire everything together. Instead of you have a single WebSocket connection and you will have your notifications, your chat possibly. It will solve the conflict. And for the storage, like we already are doing, and also the cursors for the persons. And that we really want to be a tool belt and way closer from the UI than the competition is doing.

For example, we have a provider. And I'm pretty sure we are the only one to do that, a provider for Redux or Zustand, state-management library that was really popular in React. And by just putting a middleware, you can decide which part of the states need to be synchronized in real-time and it will work in a few minutes. You don't have to rewrite your whole application to make it real-time. So, this is our positioning right now, to be more full stack than the competition.

[00:24:57] JM: What's your programming language choice for building Liveblocks?

[00:25:03] GS: So, we are mainly using TypeScript right now, both on the backend and on the frontend. For now, we did have like performance issues. But in the backend, probably we will use Rest or Go when the times come where we need to do like trusting performance improvement.

[00:25:24] JM: So, the performance issues you have with TypeScript, are they just related to latency?

[00:25:32] GS: Basically, we have like multiplayer servers. We need to have like low-memory footprint. And there is some limit that you have with JavaScript. I think it's more related to memory and the way we access data with TypeScript. If we had to use like Go or Rust to do this kind of optimization, it would be way more performant.

[00:25:55] JM: Got it.

[00:25:56] GS: You have more control basically.

[00:25:58] JM: And can you imagine a world where you would want to build a full stack with Rust and use WebAssembly to push it to the browser?

[00:26:06] GS: Yes, definitely. Currently, we are solving the conflict with TypeScript. And the background is also built with [inaudible 00:26:11]. But if you look at Figma, it's basically what they are doing. Figma, all the engine is built with WebAssembly. I think it's not Rust. But I think it's C++ if I remember correctly.

But definitely, because we are sharing some code between the backend and the frontend, and definitely having something built in Rust, where the conflict also being in WebAssembly for some specific clients would make sense. But to be honest, WebAssembly is not a silver bullet. As soon as you handle the data on the frontend, you need to be cautious about how the data flows between WebAssembly and JavaScript. And sometimes it's easy to think that it will solve performance issues. But it's also possible to introduce performance [inaudible 00:26:58] other places at the boundary between JavaScript and WebAssembly. So, before we do this move, we will try to find a really valid use case where WebAssembly is the only solution.

[00:27:09] JM: Do you know how difficult is it to write something in Rust and port it to WebAssembly? Is the build path complicated at all? Are there browser incompatibilities? Or is it a pretty smooth process these days?

[00:27:26] GS: To be completely honest, we don't do it at Liveblocks. So, I can say if it's really easy or not, I did it like in small projects. But I'm probably not the best person to give you like a state of first right now. But if I recall, it was fairly simple.

[00:27:43] JM: So, the memory issues that you could potentially have, you're talking about that you've dealt with memory constraints. Is there – I guess, with real-time applications, you can potentially have a ton of information if you're just streaming in, for example, all of the data of

people's cursors or the operational transform logs on a text-based document that's changing. Can you talk a little bit more about those memory allocation issues?

[00:28:14] GS: Yeah, sure. So, on the backend, we are relying on Cloudflare, Cloudflare Durable Object. And Cloudflare Durable Object are already constrained in-memory. I think it's 128 megabytes. So, you need to be cautious of what you keep in the cache or not in-memory.

So, it limits the size of documents. You need to find the right balance between algorithmic complexity and memory footprint. It's really hard for me to get into details without a good understanding of data structure that we are using behind the scene. But it's one of our concerns. Every time we are adding a feature, we need to think about how it would scale if there is like 1,000, I don't know, object on the backend.

[00:29:00] JM: Right. So, in that perverse case where you have some world where you could have a thousand concurrent users, do you have to do anything like prioritize certain user data? Like, prioritize some subset of those users? Do you have to do some round-robining to check those different users' changes? Yeah, like I think about the difficulty of handling some concurrent world like Fortnight, and it sounds tremendously difficult, maybe you could tell me about the constraints around developing for the thousand-user case.

[00:29:38] GS: Good question. For example, we don't support 1000 user in a single room right now. You can be 1000 people editing the same document. We scale horizontally. So, you can have like 1000 documents with like five or ten people working in separated documents. But one way that we are making that scale is to – And we are going to release this feature soon, is to a little bit like Google Doc is doing.

Google Doc, as soon as you go above like a certain threshold of editor, the document is going in read-only mode or viewer mode. So, you can't edit. You limit the number of editor. Most of them are reader. So, it's a way to make an architectural scale better. And at some point, if you don't have like any memory to connect other WebSockets and just too many users, you can have like a read-only version of the document. That doesn't – I think that you need to pull. So, you go back to the old technique of just pulling every like one minute or two minutes.

So, it's a way to decorate the experience progressively to make sure that it's scale. And I think, right now, we are focusing more on the small room collaboration. If you do a comparison with the physical world, it will be like the small meeting room that you have in an office. We are not focusing on like the last stage conference where you have like 1000 people watching. At some point, we'll probably tackle it. But not right now.

[00:31:10] JM: Have you spoken with anybody who works on like multi-user video games or a collaboration-intense video games, anything like that, to talk about their methods?

[00:31:22] GS: So, I didn't talk about with people working on video games. But I talked to people that have been working on similar architecture. So, people that have been working on Google Doc, on applications like Figma. So, it's not exactly the same constraint. Definitely, the architecture that are behind like Fortnite, for example, would be really interesting. But I didn't get the chance. I read about it. But I didn't had the chance to talk to developers of large multiplayer games.

[00:31:57] JM: If you look at your customer base, have you seen any use cases or behaviors that have surprised you?

[00:32:11] GS: So, right now, our customers base are mainly focused on application that live in like a 2D space, like a canvas. So, it could be a video editor. It could be like some kind of virtual office where people are collaborating on a whiteboard. And depending on the vertical that they are, the patterns are completely different. Sometimes they just need collaboration for a specific use case inside their app like a few minutes every week. And some verticals every time they are launching applications, there will be multiple people. And finding the right balance in term of architecture and pricing, I feel like it surprised me a lot. It's really difficult to – How do you define the pricing if people are using like collaboration only two minutes per day, or three minutes, or if it's a full eight hours? It's scaled completely differently in the backend. So, I feel like it's something that surprised this. And we are still working on this.

[00:33:11] JM: When I think about the engineering challenges of building something like Liveblocks, it's as much about the difficulties of implementing like the core APIs and the distributed systems problems as it is about making a really good developer experience and

making sure the documentation is really good. Making sure the primitives are really good for actually consuming the APIs. Like, making sure the React components are really good. Can you talk about designing the user-facing API surface?

[00:33:50] GS: Yeah, it's one of our core focus. And we take that really seriously. We admire companies like, for example, Vercel or Stripe. They are so far away in developer experience. Every time like you face an issue, it's not just it should have been a string [inaudible 00:34:07]. It's not just error message standard arms or like that. It's you, personal, but it's probably because you did that, and that in this context, we recommend you looking at this documentation.

We try to be like the most junior developers possible. And it's pretty hard when you're focused about the internal details and to see how people will understand your API. And we do a lot of brainstorming with people in the team. And we talk to the community on Discord to find the right API. We do a lot of back and forth. And we try to not release APIs too soon. So, we are using TypeScript. A lot of like JSDoc to make sure that the autocomplete and all the editors are great so you can get as many information as possible.

And we built our documentation on Liveblocks.io ourselves because all the documentation tool that we saw were not like – Are not customizable enough or like beautiful enough, I think.

[00:35:12] JM: So, I guess the main way that people are consuming these API blocks are through React components, right?

[00:35:21] GS: And JavaScript components. We have React packages, JavaScript vendor packages, and also integration with a Redux [inaudible 00:35:29]. So, in theory you can use Redux not in the React world, for example. But, yeah, basically every frontend ecosystem. And we also have a package for node if you want to use it from the backend.

[00:35:40] JM: Gotcha. Can you give an overview of the communication stack between the frontend and the backend? Like, if we want to go through one of these collaborative examples, what's being sent back and forth? And are you using like GRPC? Or what's the communication structure using to stream data back and forth?

[00:36:04] GS: We are using WebSockets. We have like an HTTP call for authentication. If you want to make like your home private. And after that, it's only WebSocket.

[00:36:13] JM: Gotcha. Can you talk a little bit more about the data structures that you're using?

[00:36:18] GS: It's completely custom. So, currently, we are parsing JSON. And it's good enough for what we are doing. But at some point, we might like uncut them like in a more like performant way, especially for stuff like cursors. We could like win a few bites. But yeah, that's pretty much it. It's not really fancy. Just JSON over like WebSockets.

[00:36:40] JM: Well, as we begin to wind down, I would like to get a sense for just more about where you're going in the future and how you're architecting the company. Tell me about some of the initiatives that you have planned right now. What you're working on?

[00:36:59] GS: So, right now we just finished a launch, a marketing launch. And we made the storage block one of our core products [inaudible 00:37:08]. Before [inaudible 00:37:09]. And now, we are going to focus on more data structure for rich-text editing. And also, improve the scalability of our backend to support more congruent users. It's probably the two main focus right now that we have. And also, better integration if you want to sync your database with Liveblock server. So, webhooks and – Yeah, webhooks.

And we are hiring. We are currently going. We just announced our seed round. We are a remote-based company. And we are looking like for software developers and design engineers mainly to tackle these issues from the data structure to how to present it in a best way possible to other developers.

[00:37:58] JM: Cool. So, anything else you want to add before we close off?

[00:38:02] GS: I mean, no, we are a remote-based company. We have a lot of fun building these APIs and trying to make this kind of complex problem as simple as possible. And if some

people are interested to solve these complex issues with us, feel free to go on Liveblock.io and apply.

[00:38:18] JM: Great. All right. Well, thanks for coming the show. It's been a real pleasure.

[00:38:21] GS: Thank you for having me.

[END]