



Chris McCord on Phoenix's LiveView Functionality

Adam Conrad

From the Editor

In Episode 394 of "Software Engineering Radio," Chris McCord, creator of the Phoenix framework and author of *Programming Phoenix 1.4*, discusses Phoenix's LiveView functionality. Host Adam Conrad spoke with McCord about how LiveView was created, use cases for integrating LiveView with Phoenix applications, the benefits and drawbacks of LiveView in comparison to such frameworks as React, Angular, and Vue, and the internal workings of LiveView. To hear the full interview, visit <http://www.se-radio.net> or access our archives via RSS at <http://feeds.feedburner.com/se-radio>.—Robert Blumen

Adam Conrad: Please give an overview of Elixir and Phoenix.

Chris McCord: Elixir is a programming language developed in 2011. It's been successfully used in production for at least five years. It runs on the Erlang virtual machine (VM) and can use everything that's available in Erlang. In addition to being a functional language, it has a concurrency model that has preemptive scheduling, allowing us to scale to millions of users per server and build distributed systems in a different way than most modern languages.

Phoenix is the de facto Web framework for Elixir. I had worked on building primarily Ruby-on-Rails

applications for six years before starting Phoenix. I had found that making the real-time Web work with Rails was not viable. I had worked on a library to do real-time updates with Rails, and I knew it wasn't going to scale or be reliable. We applied both our positive and negative experiences with Rails to the framework.

I found an article on Erlang and remembered that Jose Valim, from the Rails core team, had started a language called *Elixir* to run on the Erlang VM. I got excited about the prospect of scalability with Erlang. I got hooked on Elixir immediately, and all I needed to do to use it for everything I wanted to build was to write a Web framework, because none existed in Elixir at the time. People started using my framework, and here we are today.

LiveView is now a big part of that experience. What is LiveView and what inspired you to create it?

LiveView is a way to build rich, interactive applications without having to write JavaScript. LiveView has been in development for about a year. We had to build all the plumbing with the Phoenix Web layer, Phoenix's real-time layer, and Phoenix PubSub. These building blocks allowed us last year to accomplish what I was trying to solve with Ruby—to write real-time applications, server rendered, but with almost all of the benefits of a single-page app.

What are the shortcomings of JavaScript and JavaScript development for client-to-server interactions?

JavaScript allows us to do a lot of things that are impossible to do otherwise.

But with modern JavaScript development, every time we've gone all in on a single-page app, it has always been more complex than its server-rendered counterpart, buggier, and harder to evolve with the pace of change.

A lot of us accept the complexity of JavaScript development for good reasons, but it's a huge burden. The pain comes from overly complex solutions. Single-page apps are inherently more complex, and that's what leads to the pain. LiveView provides an alternative that gives us these bits of rich user experience (UX) that we can operate with a normal server-rendered model.

Can you use JavaScript in some places and LiveView in others, or even on the same page?

Yes. We give you an escape hatch to write JavaScript as needed. Those two worlds coexist.

Is LiveView JavaScript or Elixir or a combination of both?

A combination. At the moment, it's like 1,300 lines of JavaScript that we write for you that powers all of this. It's not a lot of JavaScript. The real innovation happens on the server with LiveView, but it does require JavaScript on the client to update the page.

What we send on the initial page render is just HTML. So, even if you're a Web crawler or you have JavaScript disabled, you will get a rendered HTML page. There's a lot of benefit that we get even if we have JavaScript running behind the scenes once we connect to the server. Since you have a full HTML page, you can now immediately distribute that for search-engine optimization (SEO) purposes.



SOFTWARE ENGINEERING RADIO

Visit www.se-radio.net to listen to these and other insightful hour-long podcasts.

RECENT EPISODES

- 396—Barry O'Reilly of Black Tulip Technology discusses Antifragile Architecture, an approach for designing systems that actually improve in the face of complexity and disorder with host Jeff Doolittle.
- 395—Host Justin Beyer discusses security and privacy concerns as they relate to machine learning with Katharine Jarmul of DropoutLabs.
- 393—Jay Kreps, chief executive officer of Confluent, discusses an enterprise integration architecture organized around an event log with host Robert Blumen.

UPCOMING EPISODES

- Pat Helland on data management at scale.
- Jeremy Miller on waterfall compared to agile.
- Rich Harris on Svelte.

LiveView allows us not only to skip all the single-page-app JavaScript complexity, but it also removes an entire abstraction layer of just Web development. With JavaScript, even from the simplest feature that you could write for a client-side app, you have some level of server communication. You have to create an HTTP endpoint and write code between the client and server. It will also involve setting up the route in your router and figuring out the naming for the controllers.

These are all easy to do, but you're making decisions just to get these client-server worlds glued together. Then you're having to write what the payload looks like on the server, how you serialize the data structures in Elixir or whatever language you're using, and how you get them back to the client. The client has to also be aware of this kind of payload contract. Whereas,

with LiveView, all of this falls away, which vastly simplifies the programming model of building a Web app.

Is there an ideal application that LiveView is suited for?

It's especially well suited for the boring business problems that we're all solving repeatedly, which is a large class of applications. If you're writing a static HTML template application today that just renders some tabular data, you have the option to make that real time, out of the box, with the same amount of work. It unlocks potential for applications that you're already used to writing, and it also allows you to accomplish more complex user experiences in less time.

Dashboards are a great example. Any time you're trying to display information that's frequently changing and you want the client



ABOUT THE AUTHOR



ADAM CONRAD is a director of engineering at Indigo with more than a decade of experience in software development. His interests include user experience, human-computer interaction, front-end Web development, and augmented/virtual reality. Conrad received his engineering degree from Brown University. Contact him at conradadam.com.

to update, you can just write a single line in your LiveView that subscribes to some PubSub events and then two more lines of code. Your dashboards will just update with the current requests per second, the current weather, current tickets, whatever you need for your business.


The only thing that it's not good for would be something that

requires offline mode because, if you can't connect to the server, that leaves you in a read-only state. But we haven't found out where it's not a good fit yet, provided that you don't need offline support or desktop-like functionality.

What would be a business case for why LiveView is preferable to React or plain

JavaScript? What kind of performance gains have you seen with LiveView?

I don't have hard numbers yet. If I had to guess, I believe it's an order-of-magnitude reduction in lines of code. That also doesn't account for the server-side code that you may not have to write. Speed of development and productivity are going to be unbeatable compared to a single-page app.

With LiveView there is no HTTP overhead, no parsing the session, no authentication. We don't have to fetch the current user from the database, because we already have them. Although it is counterintuitive, we can have a faster, less-latent user experience with a server-rendered app compared to a single-page app, and we also produce less data on the wire. 

IEEE Software (ISSN 0740-7459) is published bimonthly by the IEEE Computer Society. IEEE headquarters: Three Park Ave., 17th Floor, New York, NY 10016-5997. IEEE Computer Society Publications Office: 10662 Los Vaqueros Cir., Los Alamitos, CA 90720; +1 714 821 8380; fax +1 714 821 4010. IEEE Computer Society headquarters: 2001 L St., Ste. 700, Washington, DC 20036. Subscribe to *IEEE Software* by visiting www.computer.org/software.

Postmaster: Send undelivered copies and address changes to *IEEE Software*, Membership Processing Dept., IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854-4141. Periodicals Postage Paid at New York, NY, and at additional mailing offices. Canadian GST #125634188. Canada Post Publications Mail Agreement Number 40013885. Return undeliverable Canadian addresses to PO Box 122, Niagara Falls, ON L2E 6S8, Canada. Printed in the USA.

Reuse Rights and Reprint Permissions: Educational or personal use of this material is permitted without fee, provided such use: 1) is not made for profit; 2) includes this notice and a full citation to the original work on the first page of the copy; and 3) does not imply IEEE endorsement of any

third-party products or services. Authors and their companies are permitted to post the accepted version of IEEE-copyrighted material on their own web servers without permission, provided that the IEEE copyright notice and a full citation to the original work appear on the first screen of the posted copy. An accepted manuscript is a version that has been revised by the author to incorporate review suggestions, but not the published version with copyediting, proofreading, and formatting added by IEEE. For more information, please go to: http://www.ieee.org/publications_standards/publications/rights/paperversionpolicy.html. Permission to reprint/republish this material for commercial, advertising, or promotional purposes or for creating new collective works for resale or redistribution must be obtained from IEEE by writing to the IEEE Intellectual Property Rights Office, 445 Hoes Lane, Piscataway, NJ 08854-4141 or pubs-permissions@ieee.org. Copyright © 2020 IEEE. All rights reserved.

Abstracting and Library Use: Abstracting is permitted with credit to the source. Libraries are permitted to photocopy for private use of patrons, provided the per-copy fee is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.