EPISODE 1356

[INTRODUCTION]

[0:00:00.3] KP: As our guest today points out, most enterprise software applications are essentially forms for collecting data. The form tag and related components started appearing in HTML fairly early on. Those same concepts are still in use in modern web browsers. However, the technology for capturing the state of that form, validating its inputs and providing other common services for management to form data has continued to evolve in many languages and frameworks. Erik Rasmussen is the author of many popular open source libraries, including Redux Form, and React Final Form. In this interview, we discussed the need those tools fill in the market, and some of the modern approaches for form state management.

[INTERVIEW]

[00:00:47] KP: Erik, welcome to Software Engineering Daily.

[00:00:49] ER: Thank you.

[00:00:50] KP: I'd love to start off with a quick summary of your history as a developer. Could you let listeners know about your journey?

[00:00:56] ER: Well, let's see. I started off doing database stuff. When I was a teenager, I worked with a company that managed data for a mental health progress, using FoxPro and old database tech. In college, I played around with C++ and I had some courses on Lisp and stuff like that. When I got out of college, my first job was with Java and Java Server Pages and servlets, and all of that jazz. That was pretty cool at the time.

I stayed with that job, basically, doing Java. It evolved a little bit, but it was basically, Java servlets for 15 years. During that time, I really got into the JavaScript community and building more single-page web applications, first with Angular. Then as soon as React was open sourced, I immediately saw that that was the one true way going forward, at least for a while. I've been big in the React community since then.

[00:02:04] KP: The way it was originally presented to me is HTML would be static data primarily. Then, anything more interesting would be a servlet, or something like that. Of course, that's not the way things ended up. What changed?

[00:02:17] ER: I mean, I even remember, I did some work, when you had to put the backend stuff in the CGI bin directory, where it was the server was literally running a compiled executable on every request, which was thinking back on it, is wild. Although, what is serverless? Pretty close. Yeah, what changed is that, at first, the World Wide Web was pages. It was cool, because we had this metaphor for what a page, a physical piece of paper is. Now, we had a way that you could touch one of the words and go to another page. That was really game-changing at the time, of course, especially it really took off in academic circles, where you could click through your bibliography sources. That's great for reading.

As soon as you want to get data from the user, you have to load the data from the server to present into the page, but then also, receive the data back. That's when the web application was born. It really opened the door for a lot more powerful stuff to be done on this dumb terminal of the web browser, that you didn't have to install an application, I don't know, to look at your bank statements or whatever.

[00:03:35] KP: We've seen a progression of different approaches to this problem of getting user data and making that easier. We've converged on things, like React. When you look at the big picture, have we materialized on the right design pattern?

[00:03:50] ER: Everyone thinks they're at the end of history. All the physicists 100 years ago, were sure that we've discovered all of the laws, and now we knew everything, which is hilarious, looking back. I guarantee you, in 20 years, we're not going to be using React. We're going to be using something that evolved from everything that we learned using this declarative UI paradigm. Already, you're seeing the pendulum swing back towards hydrating templates, rather than trying to render the whole page on one go. For sure, we're not at the end stage. I look forward to seeing where we end up going.

[00:04:32] KP: Well, I don't know if you would consider yourself an early adopter of React. It seems that label would be fitting. When you started looking at the framework, what did you see as the opportunity for improvements in how developers gathered data?

[00:04:45] ER: Well, for sure, I would consider myself an early adopter of React, pretty much as soon as it was open sourced, I started building stuff with it. I realized that we had a state management problem. You could keep your state in your component, but there still needed to be some higher level state. Everyone writing React at that time suffered through that. Then when Facebook came out and said, “Hey, we've invented this paradigm called Flux.” The Flux model. It's this one-way data flow. You have these stores, and you can put stuff into the stores.

This is the way that we think you should manage your state in React. They said, “Here's a paper about it. We're not giving you any source code, or anything.” Everyone in the React community immediately started building Flux libraries. I built one for my own use. There were some popular open source ones. Eventually, all of that combined into what became Redux.

You're asking about getting data from the user. Angular had this two-way data binding system, where you put an input, and you said, “This is the variable that I want to be able to access this value in my JavaScript,” and then it just kept that in sync for you. React had this concept of a controlled component, which meant that you had to manage the value of your input. That was all that the React documentation gave you was, look, this is how to make a controlled input. Good luck.

I like to joke sometimes that, of course, Facebook didn't think about having more than one input on their forms, because Facebook's primary interface is just a single input. That's of course, silly, because their actual customer-facing stuff is all the ad stuff in the background. React made it really hard to get data from the user. If you had more than three inputs, it was complicated.

At the time, I was building an application that needed 15 inputs on one page. There wasn't a really obvious way to do this in React. This was right around the time that Redux was taking its throne, as the way to manage state in React. I was in one of the React chat groups with the creators of Redux. They were there talking all the time about state management and stuff. I asked Dan Abramoff. I said, “Hey, I need to figure out how to manage my form values. Is Redux going to be fast enough to dispatch an action on every single key press?”

I recall his response being, “I don't see why not. Try it.” So I did. I built a little reducer to manage the manage multiple form values. It worked pretty well. I open sourced it. It was really my first, I'd open sourced a couple things, but this was really the first one that I thought, “Hey, I might be onto something. It quickly became clear that the community, this was an itch that everyone needed, everyone had. It really filled a need that the community had.

Redux Form was the name of the library. It got pretty popular, especially, it got so popular that people needed extra use cases and stuff. When I first wrote it, I was, as with all open source stuff, the initial author is solving their own problem, and then they open source it. Then, people come along with slightly different problems.

It had never occurred to me that you wouldn't know all of the inputs that your form was going to need at programming time, right? I thought, you're always going to know if you need 10 inputs, or seven inputs, or whatever. No. Then people came to me and they said, “Well, we need a button to add a new input.” That dynamic form shape thing had never occurred to me. It was an interesting problem to try and solve. With the community, I helped build Redux Form into a pretty extensible library for managing form state, which was a great learning experience for me.

[00:09:05] KP: Releasing something open source, I guess, it's as simple as just changing the status on GitHub or something like that. What does it take to get your library to be seen and get people to adopt the work?

[00:09:17] ER: I don't know. I mean, I think it's just one of those right place, right time scenarios. There's a couple things that worked to my advantage. At least once, Dan tweeted about it, which got a lot of eyes. Also, my previous open source work was I built this – It was meant to be an example template for starting a React web app. It was using express for the server, and it was using Webpack and the Webpack Dev hot reloader technology. It was also server side rendered. With every request, it would render it once on the server and ship that to you and then rehydrate.

These are all basic table stakes now. At the time, it was like, the technology was just barely able to handle this. I had this open source repo for this template of how to build stuff in React. It was fairly popular at the time. It's still, I think, one of my repos with the most stars, and sometimes people still add a star to it for some reason, even though I haven't touched it in five years. I used a little bit of that fame there, and I made an example of this is how to do a form, and I used Redux Form.
I think, I got a lot of people that just forked that repo to start whatever they were working on. By default, they had Redux form in their application. I think, that helped a lot in getting popularity. The rest is just yeah, it was a problem that people had. When you looked around at the time for how to do this, that was the most used way of doing it.

[00:11:03] KP: Well, since the time you released Redux forms, the React project has continued to evolve quite a bit. I don't think there were hooks back then, for example.

[00:11:11] ER: No.

[00:11:12] KP: What's the current state of Redux Forms? Is it still relevant, and/or has the project evolved in certain ways?

[00:11:18] ER: Right. Redux Form still works. It still has millions of downloads every month. It's still being used in lots of places. Redux Form had several issues. One, is that as the manager of the open source library, I was in the habit of saying yes to all the feature requests. People said, “Hey, can I do this?” I would say, “Well, let's see.” Then I would solve the problem and I would say, “Hey, now it does this.” It's got to the point where the bundle size for the library got pretty large, because it did a whole bunch of stuff.

People would complain, “Hey, I just need a login form, or I just need a thing with three inputs. Why do I have to have this huge library that my users have to download?” That was one issue. Another question that I got a lot was, there are people using Redux in other frameworks. Why is this specific to React? Why can't we use it in Angular, or some other framework? Another question that people had was, because of the way that it was done with the binding to the map state to props or whatever, it couldn't support render props. This was the time when render props were gaining momentum as a thing. This is way before hooks, of course.

People wanted to pass in a function. Render prop is a function that describes what to render, that you can give to a component. All Redux form could do was you could give it a whole component, not just a little function to render your input. I had all of this feedback, and I started thinking about it. I realized that what form state is, is not specific to React at all. It's what your form state is, is it’s, obviously, the values of your inputs. It's also, are they valid? Have you run validation on them?

Also, things like, which of the fields has the user visited? Which of the fields has the user visited, and then blurred? Have we tried to submit yet? All of this stuff, there's a whole bunch of things that makeup what form state is. It's not related to React. It's going to be the same set of information that we're going to be using, when we're no longer using React 20 years from now, when we're all filling out forms on our VR headsets, or whatever. Whatever our application is, it’s still going to have to track all of those specific things.

I thought, if I could just write a standalone engine to manage those values, that would be really powerful, because then, it could be used with any framework. I set about building such a form engine. I released another form library, this one called Final Form. Because I love React, I also released a wrapper around it that maps it to React, called React Final Form. I was able to implement render props with that.

Another goal that I had with Final Form was, remember how people, when they wanted a small form, they didn't want to have to download all of code? I made it modular. If you need the code to manage an array of fields, you can add on another library and build the form library that you need for your application. Final Form has also become pretty popular. Because I released it a couple months after Formic was released, a lot of the – Like I said, remember how with Redux Form, there was a lot of hunger for the ability to manage form state and React.

At the time, there was a lot of hunger to manage form state and have render props. A lot of the people that were using Redux form whenever to use Formic, because they supported render props, Formic took off in a way that there's no way that Final Form is ever going to catch it in popularity. There still are some real diehard final Form Fans.

As I had hoped, when I initially wrote it, people have taken that internal core and they have mapped it onto view and web components. There's several libraries that are using that internal core, and then are building something else around it, which I think is pretty cool.

[00:15:44] KP: Am I correct in assuming that that core is probably in pure JavaScript, and that there are these auxiliary, lightweight wrappers to let me use it in other frameworks, like Angular and React?

[00:15:54] ER: Yes. The other core tenant of Final Form was – some of my complaints with Redux Form was that components would re-render when they didn't necessarily have to. I wanted to make it very explicit that if you needed to, you could really fine-tune Final Form, so that it would only re-render the parts of your page that you needed to render. Again, this was one of those things where when I started out writing Redux Form, I thought, "Oh, yeah. 15 fields is a huge form." Then I had had users come to me and say, "Hey, when we're adding the 400th input, things start to lag a little bit." It's like, wow. Okay.

I wanted to make it specially performant when you need it to be. Yes, it's just a pure JavaScript implementation. Then you can subscribe to either form, state or individual field states as you wish, and you will be notified when those change.

[00:16:58] KP: The use case of a simple login form makes perfect sense to me. What's the use case that demands 400 fields?

[00:17:06] ER: I don't know. Some huge medical survey, or something. I did some consulting work the other day with someone that needed some help with Final Form and hadn't really understood the ability to fine tune for large forms. They had a page, like the form spanned the whole page. On the page, they had six different tabs across the top. In each of the tabs, they had a list of 40 records. Each of those records had 10 inputs on them. Quickly, right there, you're multiplying into many, many inputs.

Enterprise software, is basically, all forms. If you watch the receptionist at your dentist when you go, they're just typing into forms all day. That's what all the government employees are doing all day long is typing into forms. There's a lot of forums out there.

[00:18:04] KP: Well, you'd shared some examples of the way in which user feedback has driven extensions you've made to the project, enough so that you're even worrying about the bundle size. As that bundle gets bigger, your service becomes more robust. It might also be the case that you introduce a learning curve. There could be a balance here between robustness and complexity, versus developer productivity and onboarding. Do you have any design principles for how you mitigate that?

[00:18:31] ER: Yeah, that's tough. Every time someone says, "Hey, this almost does exactly what I want, but it'd be great if I could pass in this Boolean flag, and that when this configuration flag is true, then it doesn't behave in this – then it behaves in this other way." You end up in this configuration hell, where you have all these different potential settings. Maybe mark all fields is dirty when you get an error from the server or something like that.

It does add complexity. That's just the trade-off. You can't have simplicity, and also tons and tons of functionality. That's why you can't sit someone down and have them open up Adobe Photoshop and figure out how to draw a circle, right? It's hard, because it's such a full-featured application. If you give someone Microsoft Paint, they're going to be able to. It's just a engineering trade off.

[00:19:39] KP: Well, most forms need some amount of validation. Sometimes that's something simple, like maybe there's just a regular expression we can do to check. Other times, that could be very complicated, where maybe a remote call has to happen to look something up in a database. Are there any strategies, or best practices for managing complex validation?

[00:20:00] ER: When you're defining the local client-side validation, there's two ways you can do it. I call them record level validation and field level validation. Record level is, here's a JSON object of all of the form values, just like we would submit, if you hit submit, is this valid? For that, the validation function has to return another object that has the errors with the same keys as the object that it was given.

Your other option is field level validation, where you specify this is for this field, minimum value is five, and the maximum value is 10. That field doesn't necessarily have to know about all the other values in the form. Although very quickly, you discover that no, actually, even when you're doing field level validation, sometimes you do need to know about other values, like, does this new password confirmation match the new password that you're given? Or is the checkout date after the check-in date on your hotel or whatever?

Then, there's the other two places you can do validation are on submit, generally, that's happens on the server. That's a thing, where you submit the form and the server check something and gives you back an error. Or you can do a sync validation when a field is blurred. An example of that is like, you're signing up for a service and you choose your screen name that you want, it double checks to make sure that no one has that screen name yet. Yeah, it does get complicated.

The most complicated part of Final Form, for sure, is the async. Validation. Say you've chosen the screen name, and then you've tapped to the next field, and the web page is checking with the server to see if that's available. Before you get that response back, the user hits submit, there's this async race condition stuff that can happen, that can be very complicated. Yeah, it's a hard problem. I mean, that business logic has to happen at some point.

[00:22:08] KP: A couple of times in my life, I've wanted to get some data out of some website, where maybe they had pagination in place. I went under the hood, was able to mock my own request, or alter the form, send a different limit and get a big request back. It's not exactly an epic hack, but it's something a form builder might have to contend with, if they have sneaky or clever users. There's also the sensitivity of the data being put in the form. Do you have to take any security stance as you develop the library?

[00:22:38] ER: Right. My position as the library author is, you give me your on-submit function, and I don't care what you do with the form values. Obviously, the right way to build a form, the full stack form, you have to also validate your values that you received from the form. Just because you have a JavaScript thing that's running, that make sure that for sure the person has entered their street address, and if they hit submit, you really need to be checking that on the backend, too. Because anyone can – most people don't. For sure, you can spoof a post-request, and potentially, get invalid data in your database.

That's one of the reasons that I favor what I described as record level validation, because if you're validating the whole object of all the form values, if you're using node on the backend, you can share that exact function to validate on the backend as well, which is very handy to make sure that you don't get bad data in your database.

[00:23:45] KP: Well, my understanding of the success of Redux Form and also, your inspiration to build Final Form was really about filling a need that didn't exist. Are there any vacuums in the market today that might inspire a next generation of a form library from you?

[00:24:01] ER: That's what Final Form attempts to be. It has nothing to do with the name. There was so much namespace pollution on NPM, with anything using the word form, that the only remotely rememberable praise for forms was final form, so I went with that. Ironically, it could be, like I said, the form library engine that at least as long as we're using JavaScript, could extend into the future.

I had that hypothesis tested way sooner than I expected to, because I wrote this long before hooks came out. Hooks are a totally different way of managing state in React. When I went to rewrite React Final Form to use the hooks state management paradigm, it was interesting how easy that was to do, because the design of Final Form was such an it's all subscription-based, where you say, “Hey, I want to subscribe to these values about this field.” The subscribe function returns the unsubscribe function, which is a pretty standard thing in the observable pattern.

Guess what the API for use effect is. It’s exactly that. It was quite easy to rewrite React Final Form to use hooks. As I was doing that, I had the realization that, “Oh, wait. This field component, all of the hooks that I'm using in this, I can abstract all of that out into a use field hook. Then my field component is nothing, but a call to use field. Then I can export that use field hook.” That allows anyone to turn any component into a field component, connected to Final Form. That was very powerful.

In the form space, yeah, I don't know. I don't think there are really unsolved problems. It's just that does necessarily get messy with async validation, like I was saying. There's no obvious way to get around that. I'm always thinking about it. I have several other ways of thinking about the flow of forum state that I've thought about over the years, but none of them have materialized into anything that's particularly better than this subscription model.

[00:26:24] KP: Well, one quick code level implementation question for you. A lot of people who are new to React hooks find an added challenge in getting to different components to share data amongst each other. If I've built a complicated form, it's built out of a ton of different components. What's the developer experience like, if my state's distributed across a bunch of stuff?

[00:26:47] ER: With React Final Form, you put a form component around your form, and that creates an instance of the Final Form core engine that gets put into context. Then anywhere inside that, you can subscribe to parts of either form state, or field state.

[00:27:08] KP: Well, let's talk about it in the context of the state machine abstraction. It seems like, the state of my form is pretty simple, right? It's just the values I've put in every field. What else is there?

[00:27:20] ER: Yeah. It seems pretty simple. There's really three or four different parallel state machines just within a field. Not even talking about the whole form. There's whether or not you currently have the focus. Then, there's whether or not you currently have an error, whether or not you are currently asynchronously validating. Then, there's your value. There's quite a bit of state just within one particular field. Then there's whether or not you have been visited, you have had the focus ever. That's really useful for only displaying errors after the user has tapped out of an input. You don't aggressively tell the user, “Hey, this field is invalid,” before they've even had a chance to try and fill it out.

I've taken a run couple times at implementing a field with a state machine, like current favorite library X state. I've been learning a lot about state machines. It's been changing the way that I think about how state should flow through an application, and whether it's this top down Redux way of thinking about the state lives up at the top somewhere. From below, you can subscribe to it or update it. With X state and state machines, it's more about this actor model, where each component has its own state, and has a way to talk to other components that need to know about changes. Rather than tell the top level that your state has changed, you can tell an actor that is one level up, or one level up and over.

Because you are coding, it's like, you're anthropomorphizing the component, where the user clicks the button on your component. The code you have to write there is okay, who needs to know about this? Then, you send an event to whoever needs to know about it. Then, as far as you're concerned, as that component, you're done. Then you go to the component that needs to receive that, and you put yourself in that component shoes and you say, “Okay, I've received this event. What does this mean about my state? Who else do I need to tell?” It's very, very interesting. I've introduced it to my team lately, and everyone is really, really enjoying that way of thinking about data flow.

[00:29:44] KP: Are you applying these data flow principles to your process of developing the library, or is this and other aspects of your work?

[00:29:52] ER: Potentially, form library could be using state machines under the hood and the consumer of the library wouldn't really need to know how that was being done. I could conceivably rewrite the core of Final Form as with state machines. I lately have been using them more not as a library developer, but as an application developer to organize my state.

For example, if I were building a form where I had some need for my form to be in different states, I would use a form library. I would use React Final Form. Then when React Final Form gave me the on-submit, then I would do something to orchestrate what happens with that. I would move my form into a submitting state, although React Final Form manages that for you as well. I would then, orchestrate whatever has to happen with my state machine outside of that.

In fact, a couple weeks ago, I gave a talk, where I built the common wizard form, where you've got a form that's on several different pages. I left the form values and whether or not the fields had an error or not, to React Final Form. Then I wrapped that with a state machine that kept track of what page you were on in the form and allowed for in my example, it was, if you're ordering a pizza or something and you choose pickup rather than delivery, then you can skip over the page where you have to give your delivery address. You can use X state and your state machine to manage the user flow, but not worry about the form state itself.

[00:31:42] KP: When working with more state machine-driven design, does that affect the way you unit test?

[00:31:49] ER: Yes. Because you can just test your individual machines. You've broken down a lot of the complex logic to what goes in and out of this actor. If I send this event to this actor, I expect to get this event out of it. It makes that a lot simpler. The fact that you don't – that your component doesn't have to connect to some outside Redux or something, makes it – I don't know, so far, I've seen it to be easier to test.

[00:32:22] KP: What are some of the other advantages you've seen in rolling this out with your team?

[00:32:26] ER: One of the bigger advantages of finite state machines is that you don't get into impossible states. The typical example of this is, when your component mounts, it needs to go fetch some data, and your hook that does that for you, gives you back your data, but it also gives you back a loading flag and also, an error value. Your code that is consuming that is having to check, are we loading? Then do we have an error? Then, here's our data. You can't be in a state where you are loading and you have an error, or you aren't loading, nor do you have an error, nor do you have data.

You can only be in a few states. Once you have a couple Booleans, unless you're thinking about that logic table of well, what if – if I have two Booleans, there's four possible states that they can be in, right? You have to keep track of that. Usually, when you have to Booleans, there's really only three possible states. I mean, I don't know, but usually, but in my experience.

Having things defined as states means that, I don't know, I find myself writing way fewer if statements, because before, I would be like, if A and B and not C, then we need to do this. With a state machine, I know that I'm in the state where two of these things are true, and I can just – I don't have to check, because I'm declaratively – I know that I'm in this particular state. It's a little hard to explain, but it changes. You're not having to check all of these values all the time. You just know that if I'm in the state, then the state of the world is such.

[00:34:10] KP: Yeah. It seems like, I also get some guarantees of what values will be when otherwise, I might not necessarily know exactly.

[00:34:17] KP: Right. Exactly. You know where you are.

[00:34:20] KP: Well, I believe both Redux Form and React Final Form are personal projects. What are you up to in your day job, lately?

[00:34:28] ER: Well, I've just joined the team at Centered, website is centered.app. As developers, we've all experienced this phenomenon, where you're programming, you're really into your code, you've got whatever component that you're working on, you're holding all of its code in your mind and you're really focused and you do this, and then you have to go, and you have to change these other things. You can get into this state where you are so concentrated, that the world drops away, and time flies, and you're super, super productive, and that you don't have any distractions. This is what psychologists call flow.

It's what professional athletes and musicians and dancers and stuff report. When Tiger Woods is lining up his put, he's not thinking about his mortgage. You can be in this level of concentration that makes you the best you that you can be, at whatever it is that you're doing. The goal of Centered is, what can we do to get you into this state, where you're super concentrated?

There's quite a bit of research done on this. For many people, having a little bit of music at a certain range of beats per minute, that don't have lyrics, so it's not messing with the language part of your brain, and it can get you into this mode where you're focused. It's also really important to have a specific list of tasks that you want to get done. What the Centered app does is, it's this thing that you run on your desktop, you list the things that you want to accomplish, and how long you want to spend. You say, “I'm going to spend an hour and a half working on these three tasks, and I want to take a break every 25 minutes,” because taking a break is also really important.

You press go and it starts playing this nice music. There's this concept of a of a flow coach that comes and says, “Okay, get ready to flow.” At the beginning, sometimes there are little stretches that they walk you through. During the breaks, there's a little breathing exercise to do. It really does work. It helps me focus. Lots of people have reported that it really helps them get more stuff done quicker. Anyway, that's where I've been working for the past couple months. I'm really excited about it.

It's a thing that could change the lives of everyone listening to this, because you really can, and with practice – This isn't everything. The more you practice getting into this state and being efficient, the easier it is to do. It's not meditation, but it's in the mindfulness spectrum of metacognition, of thinking about your state of mind and trying to get into best state to be your most productive.

[00:37:34] KP: Yeah. I had some success with something. I think it was called the Pomodoro method.

[00:37:38] ER: Yes.

[00:37:39] KP: I fell off that eventually. I feel this has more of the structure I was looking for. Where can I check it out?

[00:37:45] ER: Well, you go to centered.app, and you can sign up there. I've been given a discount code that I can give to the listeners of Software Engineering Daily. It's swdailyflows, all lowercase. That'll give you a 100% off the first month, so you can really see that it works. Everyone's methods and needs are slightly different. It's flexible to meet everyone's needs.

[00:38:12] KP: Well, anything that can help me with productivity is warmly welcome.

[00:38:15] ER: Right. Because if you can get eight hours of work done in four hours, then that's four hours of free time for you to spend.

[00:38:22] KP: For open source contributions.

[00:38:24] ER: Exactly.

[00:38:25] KP: Well, Erik, thanks for taking the time to come on Software Engineering Daily.

[00:38:30] ER: This has been great. Thank you very much.

[END]