


4 months ago

Reactive Rails: Comparing StimulusReflex and Hotwire

By Ben Vandgrift

Comparing StimulusReflex and Hotwire

Over the past few posts we've been talking about Reactive Rails and our experiences with Hotwire and StimulusReflex. If you'd like some detail, you can catch up:

- [Reactive Rails in Context](#)
- [Reactive Rails from Bare Bones](#)
- [Reactive Rails: StimulusReflex](#)
- [Reactive Rails: Hotwire](#)
- [Reactive Rails: Comparing StimulusReflex and Hotwire](#) 

This will be the last article in the series, and we'll draw some conclusions. Our goal was to examine the current state of the art in tooling for rails that would provide comparable functionality to a single page app without writing as much JavaScript as the typical SPA. In that, we found two new tools for our belts. They're not quite equal however, and each is suited to a different use case.

As an aside, we could be entirely wrong about *all* of this. Not just the takeaways, but the technical details too. We didn't do the deepest of dives, and will likely ship some experience reports after we've had the chance to use these in anger. These opinions are the results of a fairly limited experiment, and so YMMV.

With that out of the way, let's dig in.

Major Takeaways

Both Hotwire and StimulusReflex performed well enough to accomplish our goals for the experiment, and we didn't run into any major disqualifiers for our purposes.

Cost to Implement

Edge: Hotwire

We implemented our reactive chat room with live updates and some user conveniences in about 30 lines of custom code using Hotwire, using JavaScript minimally. StimulusReflex required four times that amount to accomplish the same thing, with more JS code than the entire Hotwire implementation. When it comes to high-level 'out of the box' functionality, Hotwire provided more up-front value. Likewise, in our goal to minimize JavaScript, Hotwire is the clear winner.

Both installed easily enough and without any surprises. StimulusReflex required somewhat more hand-wiring, though that wiring wasn't especially difficult to do.

Functional Range

Edge: StimulusReflex (probably)

Both StimulusReflex and Hotwire depend on Stimulus for much of their front-end functionality. Stimulus can take any project pretty far, and do so in a fairly contained way (even with a quirky syntax).

StimulusReflex provides a broader range of callbacks, and more flexibility about what parts of your page get updated and under what conditions. Likewise, if you want more control over the connection itself, CableReady can be pretty flexible.

Our application didn't need these kinds of features, so there was no real advantage in using StimulusReflex. In other projects we've worked on, it would've been handy to have the additional flexibility. It's easy to imagine scenarios where StimulusReflex could've provided an advantage over Hotwire.

Support and Adoption

Edge: short term, StimulusReflex. long term, too soon to tell

At the moment, StimulusReflex wins hands down. The documentation is thorough, there are many working examples to draw from, and there is a Discord channel if you get stuck. SR boasts 20 active developers, and the project has been available in some form for a couple of years.

By contrast, Hotwire's documentation (which is to say, the documentation for Stimulus and Turbo) is a little bit thin. We're not sure how long that will

be true however, given that the project is backed by Basecamp.

If we had to put money on an eventual winner in the 'adoption' category, it would likely be Hotwire if for no other reason than the name recognition associated with its authors and Basecamp.

Core Assumptions

Edge: Dead Heat

Both of these tools use fundamentally the same communication approach: real-time communication underpinned by WebSockets, JavaScript wrapping HTTP/S requests. Same languages, same frameworks, largely the same tooling. Even the The only key difference in the core assumptions made by the tool is how they choose to change the screen, but we'll tackle that in the next section.

Introduced Uncertainty

Edge: Hotwire

Hotwire replaces on-screen elements via directives targeting specific `turbo-frame` elements, counting on the uniqueness of the `id` attribute in the DOM. It's quite precise, even when Turbo is intercepting outgoing requests and replacing the entire `body`. Limited mystery.

Stimulus does this via morphs, depending on [morphdom](#) to do the heavy lifting by comparing the current and upcoming pages. By default this is the entire page, but you can limit the target by specifying a `morph`. So you *can* be precise, but you don't *need* to be. While the convenience appeals to us, it's easy to see potential difficulties in debugging an unexpected change to the screen.

Another area where SR introduces uncertainty is the Reflex abstraction. Because the Reflex closely mirrors the functionality of a Controller in Rails, it may become unclear where to look for problems, or where to situate a piece of functionality.

Security

Edge: 6/5 and Pick'em

Neither of these technologies impact security in novel ways. Both use the same underlying communication technology (with its own security concerns), so we'll call this a draw.

Development Speed

Edge: Not a Fair Comparison

Getting started with Hotwire was simple and easy. Getting started with StimulusReflex was not. There is an associated learning curve that while not steep, is steep *enough* to interfere with your first few StimulusReflex Projects.

This has a lot to do with the range of the tool. StimulusReflex provides mechanisms that can be used in a wide variety of ways, whereas Hotwire does one thing pretty well. In this respect, it's difficult to compare the two.

Happiness

Edge: Both!

In both cases, we manage to obviate the need for a single page app and the associated JavaScript ecosystem. That ecosystem is still *there*, but it's

hidden away and we largely don't have to mess with it. In that respect, either package accomplishes that goal.

Also, neither StimulusReflex nor Hotwire introduced additional *unhappiness*. The most eyebrow-raising thing we encountered was the Stimulus syntax, which is common to both.

Adding to Existing Projects

How much client-side interactivity is apparent in your project? How much have you invested in JavaScript to date? For projects with quite a bit of either, a Reactive Rails approach is likely to be expensive to add to your existing project.

If you have a single-page app making API calls to a Rails backend however, it may be worth iteratively replacing with Hotwire (if small/simple) or StimulusReflex (if complex)--particularly if your organization's strengths are in Ruby and not JavaScript.

Conclusion

We hope you've enjoyed exploring these two Reactive Rails technologies with us. We're using these tools internally now, and enjoying each in its place:

If you have a few simple pieces you want to swap out, Hotwire is trivial to add. If you're building a more complex application with SPA-like needs, either of these tools will make your life easier. As those needs become more complex with lifecycle callbacks, timing issues, cross-user communication and other complicating factors--or if you just hate controllers--StimulusReflex may be worth the learning curve.

Nate Hopkins (@hopsoft) has created [a single file Rails application to help you learn about StimulusReflex and CableReady](#).

All of the posts in this series are listed here:

- [Reactive Rails in Context](#)
- [Reactive Rails from Bare Bones](#)
- [Reactive Rails: StimulusReflex](#)
- [Reactive Rails: Hotwire](#)
- [Reactive Rails: Comparing StimulusReflex and Hotwire](#) 