Choosing a realtime collaboration library

Alec LaLonde

2018-03-19 12:09:21 -0700

Here at Convergence Labs, we’re excited to witness the increased adoption of realtime collaboration in software across the entire industry. This has given rise to a whole suite of technology solutions to help developers get their needs met, no matter the size and scale of their applications.

Of course, Convergence is one of those technology solutions, but this post is not about selling you on Convergence. We think it can provide a massive amount of value to a large set of people, but it’s certainly not the best fit for everyone. So let’s dive into a few use cases and give you a framework to make good technology provider decisions.

## Realtime collaboration or just realtime ?

The standard definition of realtime collaboration is the ability for multiple people to be able to work on the same piece of data at the same time. This may be a rich text document, a presentation, or maybe just the text describing a task. Many, dare I say most, applications don’t have this requirement. For example, an application that merely pulls and aggregates data in real-time (think a real-time dashboard) doesn’t fit our definition of collaboration, because there’s no data being written. Even a lot of apps that are heavily write-intensive don’t fit this definition, if they have been designed to allow only a single user at a time or implement “locks” or other measures to avoid conflicts.

Conversely, there are collaborative applications that are not real-time. Git, for example, is the industry standard for collaborating on source code, but it is by no means a real-time solution. Two team members can’t really be working on the same piece of code at once because they don’t have the context to see what the other is doing *at that moment*.

This distinction is critical when evaluating third-party libraries. If you spend six months architecting your backend around Firebase, but neglect that small co-editing feature of your app, you will be very disappointed when that feature’s implementation begins and Firebase is the only option. You’ll have to pull in something else, or try and roll your own, which is exceptionally difficult.[[1]](#footnote-21)

## Unique Constraints

So you’ve determined that true real-time collaboration is a requirement for your application. Congratulations – you’re ahead of the curve already! RTC is still a relatively undeveloped space, so let’s get into a few unique constraints that you may not have had to consider before:

* Data models. The *shape* of your data heavily affects the difficulty of managing state between multiple users. For example, one of the most common types of data, rich text, has yet to see a RTC implementation that supports a wide variety of off-the-shelf rich text editors. In fact, the *only* good example[[2]](#footnote-23) of a realtime collaborative rich text editor is Google Docs. [A](#yjs) [lot](#convergence) of libraries have focused on supporting data that can be represented by JSON, but make sure you read the documentation closely to make sure it will support the data constructs your app will require.
* Networking infrastructure. By definition, RTC requires communication between multiple remote machines. This [doesn’t have to](#swellrt) follow a client-server pattern, but from the very beginning you’ll have to worry about things like data security, guaranteed uptime, and scalability.

## Platforms

This article will be primarily focused on web-compatible solutions, for a variety of reasons:

* We know it well and it has by far the most options
* It is the most ubiquitous programming platform on the planet
* It is a natural fit for realtime collaboration

Please [contact us](mailto:contact@convergencelabs.com) if you’re interested in hearing about other real-time collaboration solutions available for other platforms and programming languages.

# Types of projects

## Toy or experimental

Engineers love playing around with new technology. To that end, the goal of these toy projects is intellectual curiosity. This is one reason why open source has been so successful: If something doesn’t work as it appears, you can read the source! Documentation missing? Read the source! You can see what you are getting, and probably learn a thing or two from the source.

If your project is primarily driven by curiosity, by all means, check out [Automerge](#automerge) or [Yjs](#yjs) for a [CRDT](https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type) or [OT](https://en.wikipedia.org/wiki/Operational_transformation)-based open source RTC solution, respectively. The underlying algorithms are quite interesting, and you can get a simple app up and running quickly.

## Startups / Lean, growth-based SaaS

Startups live and die by their speed of execution. Yet MVPs are typically developed on a shoestring budget. Technical founders need to be able to leverage existing technology, with a balance between off-the-shelf utility and speed of integration. Cost is another critical concern.

We’ve directly worked with two types of startups in this space: those with real-time collaboration as a core feature, and those treating it as an add-on differentiator.

### RTC as a core product capability

Companies needing RTC from day one fall into two camps: those with existing technical expertise with RTC technology, and those knowing they’ll need to get smart on it quickly. People identifying as the former probably won’t be consulting guides such as this :D but those falling into the latter category will have some difficult choices to make:

* Roll your own / use open source. With more and more off-the-shelf API providers emerging and gaining market share, this is becoming an increasingly difficult decision. There would be a strong argument towards needing to gain deep domain knowledge in order to make long-term architecture decisions, and building your own system would force you and your team to gain this knowledge. That road is long, expensive and fraught with mistakes, but with the right combination of talent and perseverance it would be the right call.
* Buy. When speed and getting to market quickly are paramount, it’s hard to argue against using an off-the-shelf solution. If you find product/market fit, you may be able to afford to revisit the decision, but maybe not.

Regardless of your choice, we’d recommend consulting with an expert in the space.

### RTC as a differentiating feature

* Roll your own / use open source. Everybody’s situation is different, but it’s harder to argue in favor of this decision in this case. Unless you’re a very large organization with deep pockets, rolling your own RTC will almost certainly be prohibitively expensive.
* Buy. The cost to build RTC into your app with an off-the-shelf solution will be the lowest possible. You can keep more of your developer cycles working on your core product, and not researching the ins and outs of data synchronization algorithms.
* Mitigation. Try to design your architecture such that a user’s experience is minimally impacted by the failure of your RTC provider. This means keeping the RTC data flow separate from the core application’s. If there is downtime or a disconnection, you can just display a warning message and switch over to the default single-user mode.

# Commercial vs Open Source

While having access to the code is undeniably valuable, open-source solutions tend to run into the same common problems as projects mature and their requirements change. Any experienced software architect is already aware of these tradeoffs, but here they are:

* The “Tip of the Iceberg” Effect: Home-grown, single-maintainer projects tend to put the vast majority of documentation on getting developers started, and that’s it. You can get a nice whiz-bang demo running with the simplest possible use case, but it doesn’t take long until you’re running into problems that weren’t anticipated by the original author(s). A few months later, and you’re as much of an expert on the codebase as the original author, and you’ve accidentally inherited a codebase that you now realize only addresses 20% of your project’s needs [[3]](#footnote-36).
* Scalability and Growth: Web-based software is getting increasingly complex, and many OSS solutions weren’t designed to accommodate the demands of modern web applications: Multiple integrated third-party and home-rolled systems, robust security, rapid scaling, etc. Especially in the RTC space, a lot of solutions have come out of academia and haven’t seen much exposure in real-world, commercial settings. Even with the recent renaissance of auto-scaling DevOps infrastructure, *designing* scalable software is still extremely difficult.

# Solution RoundUp

## Open Source Ad hoc Solutions

### [Automerge](https://github.com/automerge/automerge)

Automerge provides automatic syncing and merging for a small set of data types. Object, arrays, and strings are all supported. Doesn’t dictate the transport layer. Uses CRDTs under the hood.

#### Pros

* Peer-to-peer, decentralized support with the MPL (Magic Persistence Layer) library
* Change history
* Offline support

#### Cons

* No rich text support
* No authentication or authorization support
* Lack of integrity checking may lead to unchecked, corrupted edits

|  |  |
| --- | --- |
| License | Popularity |
| MIT | 5172 stars |

### [SwellRT](http://swellrt.org/)

SwellRT is an open source backend-as-a-service designed to ease building realtime collaborative applications. It is a fork of the now-defunct [Apache Wave](http://incubator.apache.org/wave/) project, which was originally open-sourced when Google Wave shut down.

SwellRT is OT-based, and thus requires a master node to preserve a global order of operations. [It claims to be decentralized](https://github.com/P2Pvalue/swellrt/wiki/Comparing-SwellRT-with-Etherpad#scalability) but it has yet to be proven.

SwellRT is also the technology powering Jetpad, a realtime collaborative rich text editor. Unfortunately the UI appears to be tightly tied to the backend implementation.

#### Pros

* Built-in user management and authentication
* Included storage with MongoDB

#### Cons

* Relatively untested in production
* No support for [collaborative cues](https://convergencelabs.com/blog/2017/02/building-realtime-collaborative-applications)

|  |  |
| --- | --- |
| License | Popularity |
| Apache 2.0 | 125 stars |

## Specialty Solutions

### [TogetherJS](https://togetherjs.com/)

TogetherJS was a Mozilla Labs project aiming to make it “surprisingly easy” to add realtime collaboration to a website. Essentially, it synchronizes DOM state between multiple browsers (See Webstrates below for some pitfalls of this approach). It is open source and provides a UI widget with text chat and rudimentary audio chat to get developers going quickly.

It has a few rather large users including [JSFiddle](https://jsfiddle.net/) and [.NET Fiddle](http://dotnetfiddle.net/).

The original creator of TogetherJS, [Mozilla Labs](http://www.mozillalabs.com/), is unfortunately no longer active, and [the project is in need of a maintainer](https://github.com/mozilla/togetherjs/issues/1127).

|  |  |
| --- | --- |
| License | Popularity |
| Mozilla Public License, Version 2.0 | 6k stars |

### [Webstrates](https://webstrates.github.io)

Similar to TogetherJS, Webstrates does DOM-level synchronization. It also includes a general purpose messaging framework between connected clients. The actual data synchronization is performed behind the scenes by ShareDB.

DOM syncing is a fairly straightforward approach towards providing a feel of realtime collaboration. However, web developers have for a long time been aware of the drawbacks of basing their applications on the DOM rather than a domain-specific javascript (or typescript, etc) model.

|  |  |
| --- | --- |
| License | Popularity |
| Apache 2.0 | 73 stars |

### [Flip](https://irisate.com/)

The Flip framework was extracted from Ohm Studio, a richly featured digital audio workstation with realtime collaboration at its core. It is attacking the problem from a depth-first perspective, with heavy use of system invariants and a focus on guaranteed resolution for complex data types.

|  |
| --- |
| License |
| Proprietary |

## General-purpose solutions

### [ShareDB (formerly ShareJS)](https://github.com/share/sharedb)

ShareDB is an open-source realtime backend. It uses OT to synchronize data. [This repository] contains the types supported by ShareDB: JSON, plain text, and rich-text (scoped to work with Quill).

#### Pros

* Used in several production systems
* Extensible via middleware with community-provided modules for authorization, etc.
* Flexible storage options
* Offline support

#### Cons

* Supporting other data types requires writing transforms from scratch, which is extremely time-intensive and error-prone
* No support for [collaborative cues](https://convergencelabs.com/blog/2017/02/building-realtime-collaborative-applications)

|  |  |
| --- | --- |
| License | Popularity |
| MIT | 1.7k stars |

### [Yjs](http://y-js.org/)

Yjs is a framework for shared editing. It is structured to allow customization of storage and transports. It uses CRDTs behind the scenes and supports a fixed set of types out of the box.

#### Pros

* Support for decentralized, peer-to-peer shared data editing
* Fairly rich set of supported types: Maps/arrays, text, and rich text ([Quill](http://quilljs.com/) only)
* Out of the box support for a variety of common UI libraries (Ace, Quill, CodeMirror, etc)
* Offline support

#### Cons

* Relatively untested[[4]](#footnote-71) in production settings
* No support for [collaborative cues](https://convergencelabs.com/blog/2017/02/building-realtime-collaborative-applications)
* No authentication or authorization support

|  |  |
| --- | --- |
| License | Popularity |
| MIT | 587 stars |

### [Convergence](https://convergence.io)

We of course couldn’t write this article without mentioning our own horse! Convergence is the first and only all-in-one turnkey solution for adding realtime collaboration to web applications. After designing and implementing many realtime collaborative applications, Convergence was built to deliver the functionality that *every* RTC app needs.

#### Collaborative Cues

Where every library in this list only addresses the data synchronization problem, Convergence goes beyond, providing APIs for building collaboration awareness, presence, and chat. Any usable multi-user application will need some sort of collaborative cueing to help users avoid conflicts, and Convergence provides APIs for references and activities.

#### Pros and Cons

The primary advantage to Convergence is that it is a full-fledged backend-as-as-service, so you get storage, user management, and authentication/authorization a la Firebase. The primary disadvantage is that it is closed-source, though this comes with its own advantages, like guaranteed support. Some folks don’t want to trust their backend to a for-profit company, and we respect that. That’s why this guide exists.

|  |
| --- |
| License |
| Proprietary |

## Related solutions

For this article, we’ve focused on *general-purpose* RTC solutions that can be both be easily plugged into an existing codebase and support a relatively wide variety of data. However, there are a few related products out there that may fit some people’s needs:

* [CKEditor 5](https://ckeditor.com/ckeditor-5-framework/) is an open-source rich text editor that supports RTC out of the box. Still in alpha.

## Dead/retired projects

These projects had their time in the sun but have unfortunately been abandoned or explictly cancelled.

* [OT.js](https://github.com/Operational-Transformation/ot.js)
* [Apache Wave](http://incubator.apache.org/projects/wave.html)
* [Google Realtime API](https://developers.google.com/google-apps/realtime/overview)

### Footnotes

# NOTES

Goal: Provide an overview of the existing solutions and categorize them. Then go through a few use cases and apply a category to each.

Topics: - type of project (toy? new? existing? usage? expected longevity?) - Toy project / experiment - New project or rewrite, well-funded - Feature addition to large existing project - buy vs build - Comparison matrix (how to do this mobile-friendly?) - supported data types (plain text, rich text, JSON, etc) - proprietary / open source - group and local undo / redo - Transient data support (remote cursors, remote pointers)

1. The pace of innovation in the software industry wouldn’t be what it is without open source, and we are hugely indebted to the OSS we use on a daily basis. There is merit to contributing to and growing an open source codebase, but the vast majority of open source development is still funded through for-profit software companies. We just want to remind people of what they may be getting into. [↑](#footnote-ref-21)
2. Our colleagues over at [CKEditor](https://ckeditor.com/ckeditor-5-framework/) are [well on their way](https://github.com/ckeditor/ckeditor5/releases). [↑](#footnote-ref-23)
3. The pace of innovation in the software industry wouldn’t be what it is without open source, and we are hugely indebted to the OSS we use on a daily basis. There is merit to contributing to and growing an open source codebase, but the vast majority of open source development is still funded through for-profit software companies. We just want to remind people of what they may be getting into. [↑](#footnote-ref-36)
4. [Get in touch](mailto:contact@convergencelabs.com) if you’ve built something significant and we’ll update this! [↑](#footnote-ref-71)