Checklist:

- Laptop, charger
- Speaker notes
- Glasses, case
- Freshly restarted computer
- Only VS Code and Safari running
- presentation project
  - Closed all editors except hello.md
  - Jacked in to demo project
  - Zen mode **ON**

Clojure Lunch Stockholm #4 - feat. Replicant

Replicant

Replicant is one of those librares that makes you exclaim *“I can’t believe it’s not React!”*.

But there is no dependency on React in there. It pulls in zero dependencies! I copied this dependency map from Replicant’s `deps.edn` file.

Before Replicant, Christian Johansen’s wrote Dumdom, which was API compatible with Quiescent. But where Quiscent relied on React, Dumdom used snabbdom, a super tiny (and fast) virtual DOM library written in JavaScript.

Replicant is very similar to Dumdom, but even more data oriented, and it implements its own virtual DOM, completely in Clojure.

If you want to understand where Replicant comes from, you should watch Christian’s talk *Stateless Datadriven UIs* from JavaZone Oslo last year. Before he wrote Replicant.

Replicant 2

Like snabbdom and Dumdom, Replicant is a library, not a framework. It doesn't manage state, it doesn't even have components. It's a rather lean library, at that, with only four API functions, of which only one is needed in all Replicant apps. All Replicant does is render hiccup to the DOM.

Christan loves data. He made Replicant so that you can build apps where views are only about data, and where data travels in only one direction, from your model, to your views. There are no reactive atoms, and no subscriptions. Your views always render the same thing, given the same data.

Because views are just functions. Pure functions. And if you wire up your application carefully, your functions can return only data, and nothing but data.

Replicant 3

Christian also loves performance. So Replicant will transform your data to DOM in a very efficient manner. The Replicant code has disclaimers about some of it not being idiomatic Clojure or ClojureScript, for the sake of performance.

It’s entirely possible to write a Reagent app that would also work with Replicant, with only a few changes. You just can’t use reactive atoms, and of course no local state. If you can’t live without local state, Replicant is not for you.

Without a framework you are on your own. Yours is the responsibility to architect your app. And you are free to do it the way that makes most sense to you.

I am a big fan of re-frame. Since Replicant is not a framework, it allows me to use it as a building block for my own tiny, re-frame like framework. It won’t let me mimic the subscriptions part. But that’s a feature, not a bug.

API

There are four functions in the Replicant API:

1. You can render hiccup to a DOM element
    - For some simple applications, `render` is the only function you need to use.
2. You can unmount what you have rendered to a DOM element
    - Mostly useful when you have rendered something to 3rd party libraries.
3. If you use data oriented event handlers – you really should – you need to register a dispatch function
4. You can render hiccup to a string

Replicant Renders

The render function efficiently takes care of all hiccup -> virtual DOM -> real DOM business. In doing so, it:

- Manages the lifecycle of the elements in the hiccup
  - It even manages elements while they are monting and unmounting, giving you the opportunity to use CSS for transioning elements in and out
- Looks for life cycle hooks and calls your dispatch function when they are triggered
  - Hooks can be attached to any element in the hiccup
    * If the hook is data, it will cause your dispatch function to be called with information about the hook, plus the data
- Wires up event handlers
  - If the handler is a function, it is attached “as is”
  - If the handler is data, the event will cause your dispatch function to be called with information about the event, plus the data

Further Reading

It says something about Replicant that it has a rather small README, and that that README also constitutes the full documentation.