Guiding people into Clojure

by John Stevenson

John Stevenson

@jr0cket

Speaker, author, conference organiser & community obsessed developer. Love Clojure, Emacs, Cats, Cycling & Agile development. @Heroku @SalesforceDevs #Trailhead

London, UK (North Yorkshire)

jr0cket.co.uk

My Experiences

the last 5 years in the Clojure community

Learning by teaching others

I really started thinking in Clojure when I started talking to & teaching others

- Coding dojos
- talks on Clojure (starting with the basics, showing the art of the possible)
- moving on to running conferences
- workshops at hack days

Hack the Tower - London hackday

A hack day for developers to work on projects together, build new apps or just discover how to get the most out of all the new shiny technology out there. There are no rules or prizes to distract you, except the infinitely valuable prize of knowledge, experience and fun.

TESCO

Diversity in Clojure

Teaching is not ‘doing’

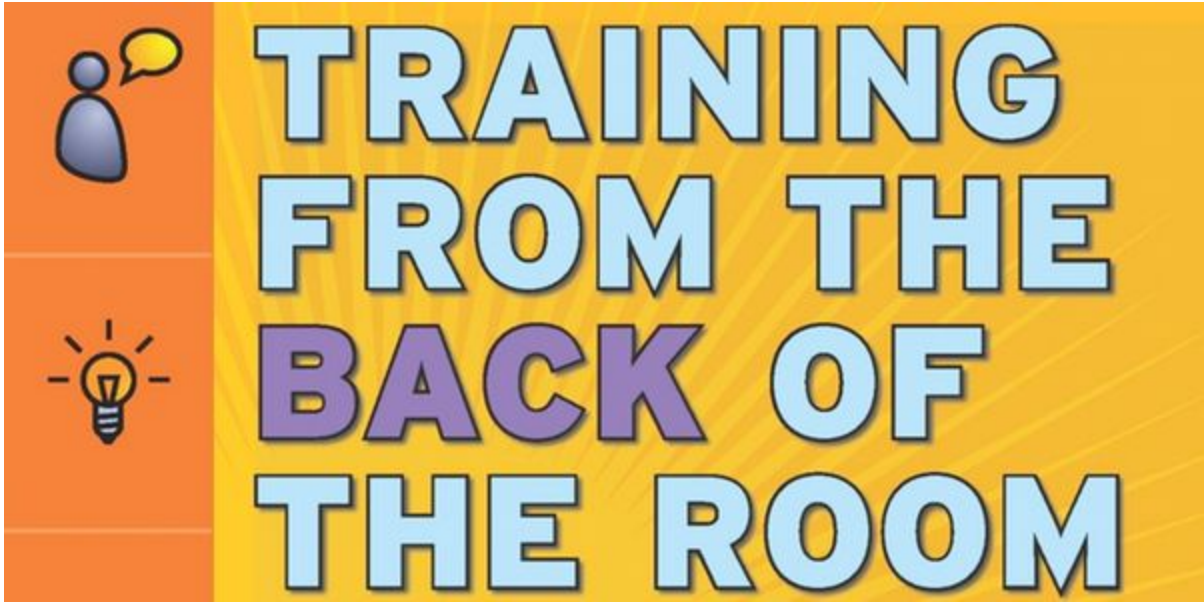
you can’t learn it for them…

Show people examples, give them content

Help them get started with tools & content they need, then get out of the way

Avoid coding for them

Never take over someones computer without permission, even then you should resist

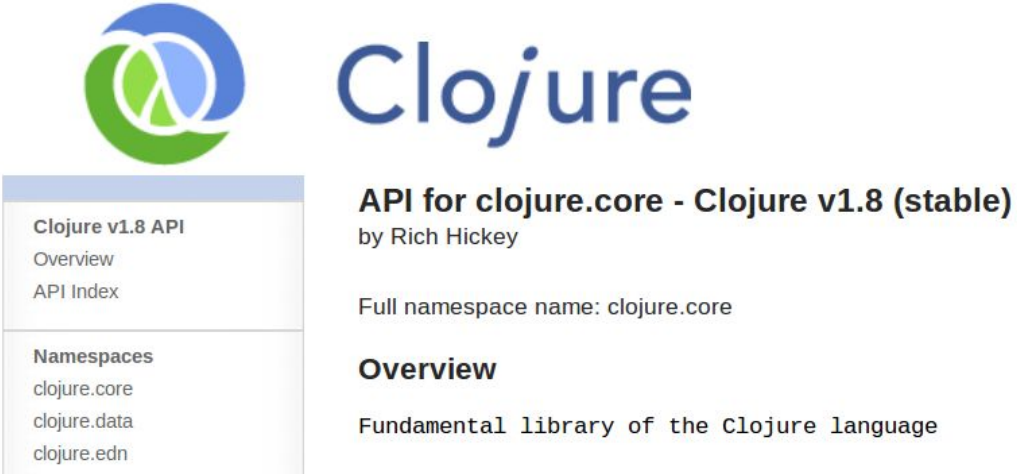


Encourage Focus

Start with Clojure.core

- there are over 600 functions there to start with
- pick some functions from different parts of Clojure.core
- add ‘easy to use’ libraries where appropriate (eg. ring, compojure)

Small steps build confidence quickly and makes it easier to introduce more concepts



Guide them through the experience

Give them a reason to learn

inspire them & build up their motivation

LightTable - Instarepl

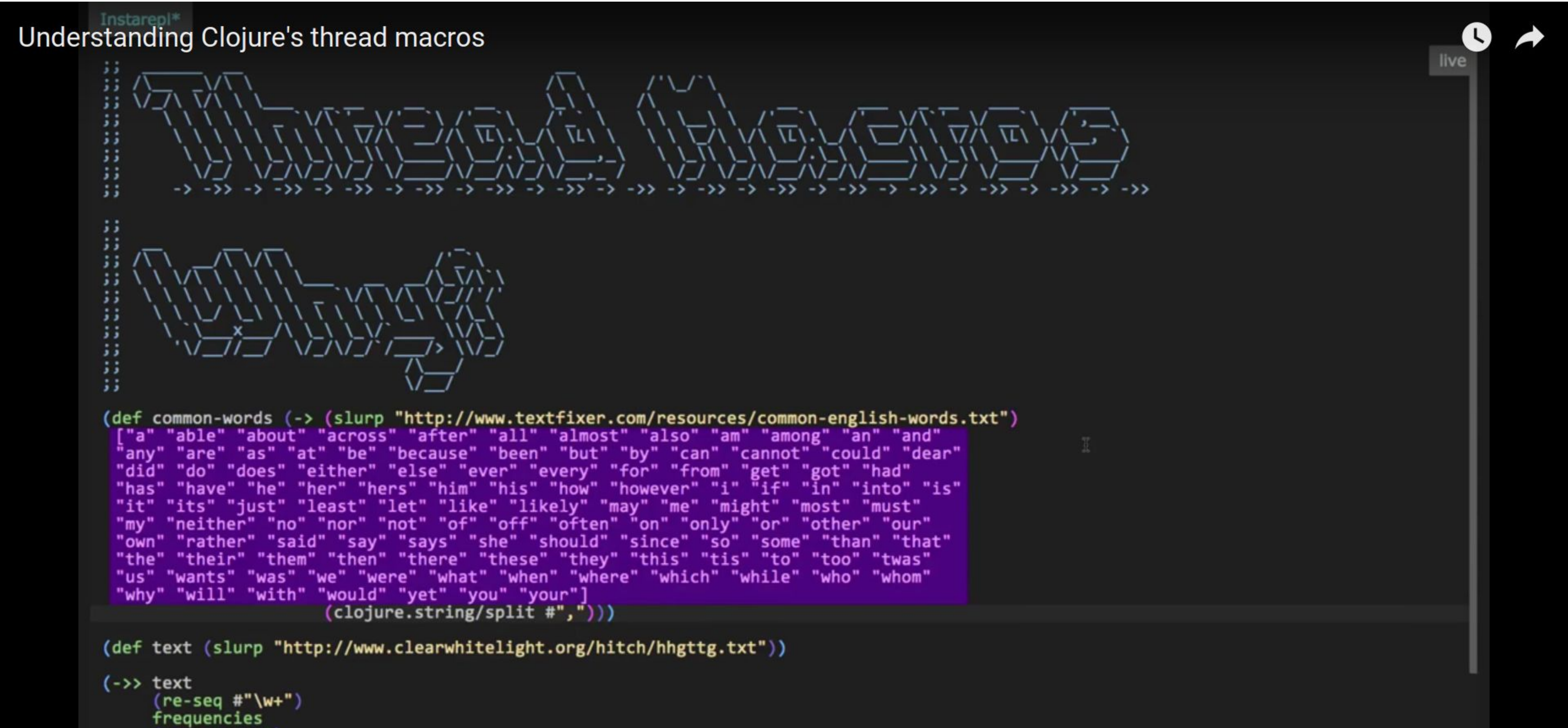
Light Table

Welcome | core.clj

```
(ns interact.core) nil

(+ 1 2 3 4 5) 15

(defn multiply-by-myself
  "Multiple a number by itself"
  [number]
  (/ number number)) #'interact.core/multiply-by-myself

(multiply-by-myself 5) 1
```

2 12 / 1

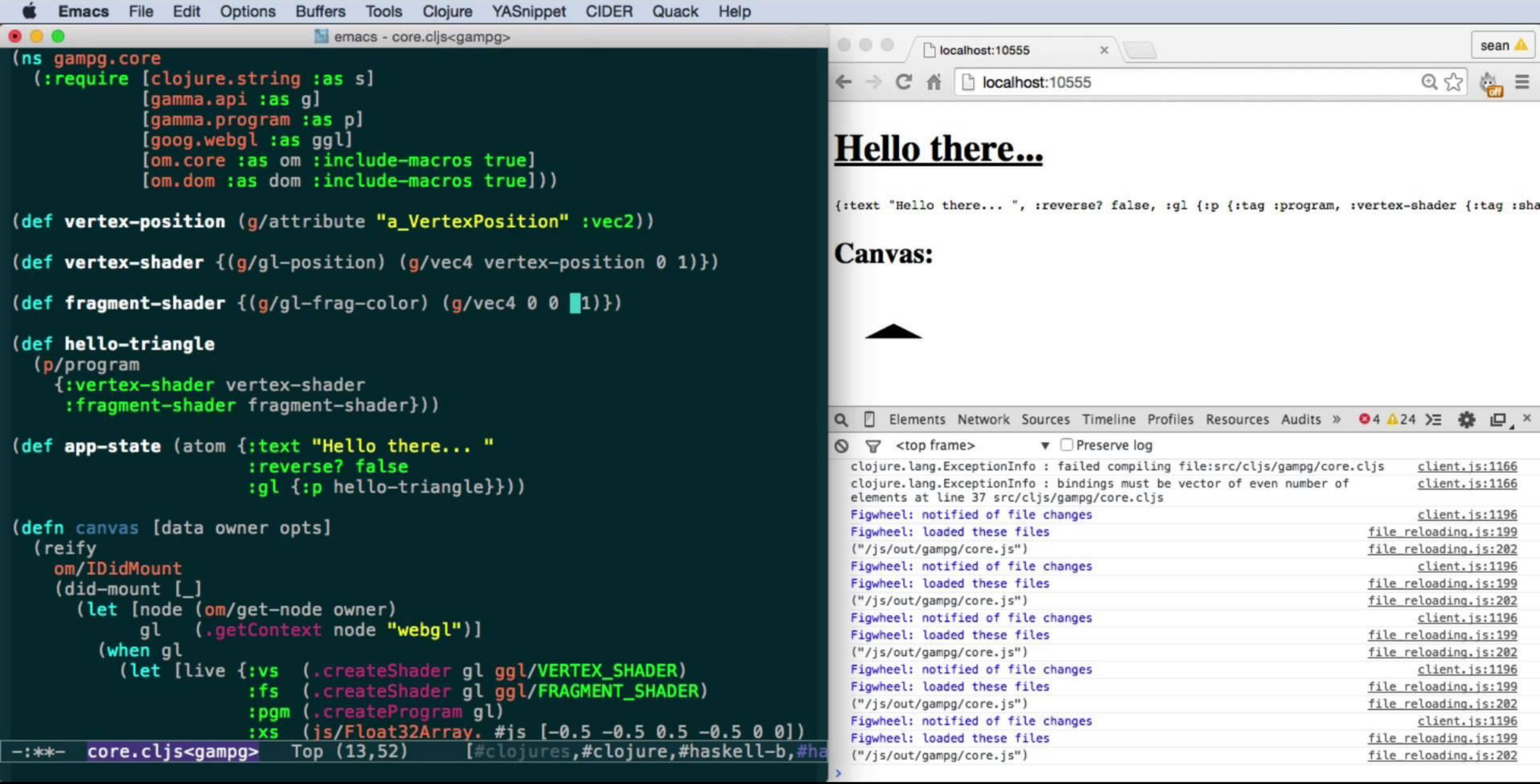
LightTable - Instarepl



```
;;    -> ->> -> ->> -> ->> -> ->> -> ->> -> ->> -> ->> -> ->> -> ->> -> ->> -> ->> -> ->> -> ->> -> ->>

(def common-words (-> (slurp "http://www.textfixer.com/resources/common-english-words.txt")
  ["a" "able" "about" "across" "after" "all" "almost" "also" "am" "among" "an" "and"
  "any" "are" "as" "at" "be" "because" "been" "but" "by" "can" "cannot" "could" "dear"
  "did" "do" "does" "either" "else" "ever" "every" "for" "from" "get" "got" "had"
  "has" "have" "he" "her" "hers" "him" "his" "how" "however" "i" "if" "in" "into" "is"
  "it" "its" "just" "least" "let" "like" "likely" "may" "me" "might" "most" "must"
  "my" "neither" "no" "nor" "not" "of" "off" "often" "on" "only" "or" "other" "our"
  "own" "rather" "said" "say" "says" "she" "should" "since" "so" "some" "than" "that"
  "the" "their" "them" "then" "there" "these" "they" "this" "tis" "to" "too" "twas"
  "us" "wants" "was" "we" "were" "what" "when" "where" "which" "while" "who" "whom"
  "why" "will" "with" "would" "yet" "you" "your"]
                      (clojure.string/split #",")))

(def text (slurp "http://www.clearwhitelight.org/hitch/hhgttg.txt"))

(->> text
     (re-seq #"\w+")
     frequencies
```

Flappy birds demo (modified)

```
(defn score [{:keys [cur-time start-time] :as st}]
  (let [score (- (.abs js/Math (floor (/ (- (* (- cur-time start-time) h
oriz-vel) 544)
                                  pillar-spacing)))
                 4)]
  (assoc st :score (if (neg? score) 0 score))))

(defn time-update [timestamp state]
  (-> state
      (assoc
          :cur-time timestamp
          :time-delta (- timestamp (:flappy-start-time state)))
      update-flappy
      update-pillars  ;; 006 lets hide the pilars
      #_collision?    ;; 005 lets forget about collisions
      score))

(defn jump [{:keys [cur-time jump-count] :as state}]
  (-> state
      (assoc
          :jump-count (inc jump-count)
          :flappy-start-time cur-time
          :initial-vel jump-vel))
```



```
(ns gampg.core
  (:require [clojure.string :as s]
            [gamma.api :as g]
            [gamma.program :as p]
            [goog.webgl :as ggl]
            [om.core :as om :include-macros true]
            [om.dom :as dom :include-macros true]))

(def vertex-position (g/attribute "a_VertexPosition" :vec2))

(def vertex-shader {(g/gl-position) (g/vec4 vertex-position 0 1)})

(def fragment-shader {(g/gl-frag-color) (g/vec4 0 0 1)})

(def hello-triangle
  (p/program
    {:vertex-shader vertex-shader
     :fragment-shader fragment-shader}))

(def app-state (atom {:text "Hello there... "
                      :reverse? false
                      :gl {:p hello-triangle}}))

(defn canvas [data owner opts]
  (reify
    om/IDidMount
    (did-mount [_]
      (let [node (om/get-node owner)
            gl   (.getContext node "webgl")]
        (when gl
          (let [live {:vs  (.createShader gl ggl/VERTEX_SHADER)
                      :fs  (.createShader gl ggl/FRAGMENT_SHADER)
                      :pgm (.createProgram gl)
                      :xs  (js/Float32Array. #js [-0.5 -0.5 0.5 -0.5 0 0])
-:**-  core.cljs<gampg>   Top (13,52)     [#clojures,#clojure,#haskell-b,#ha
```

Hello there...

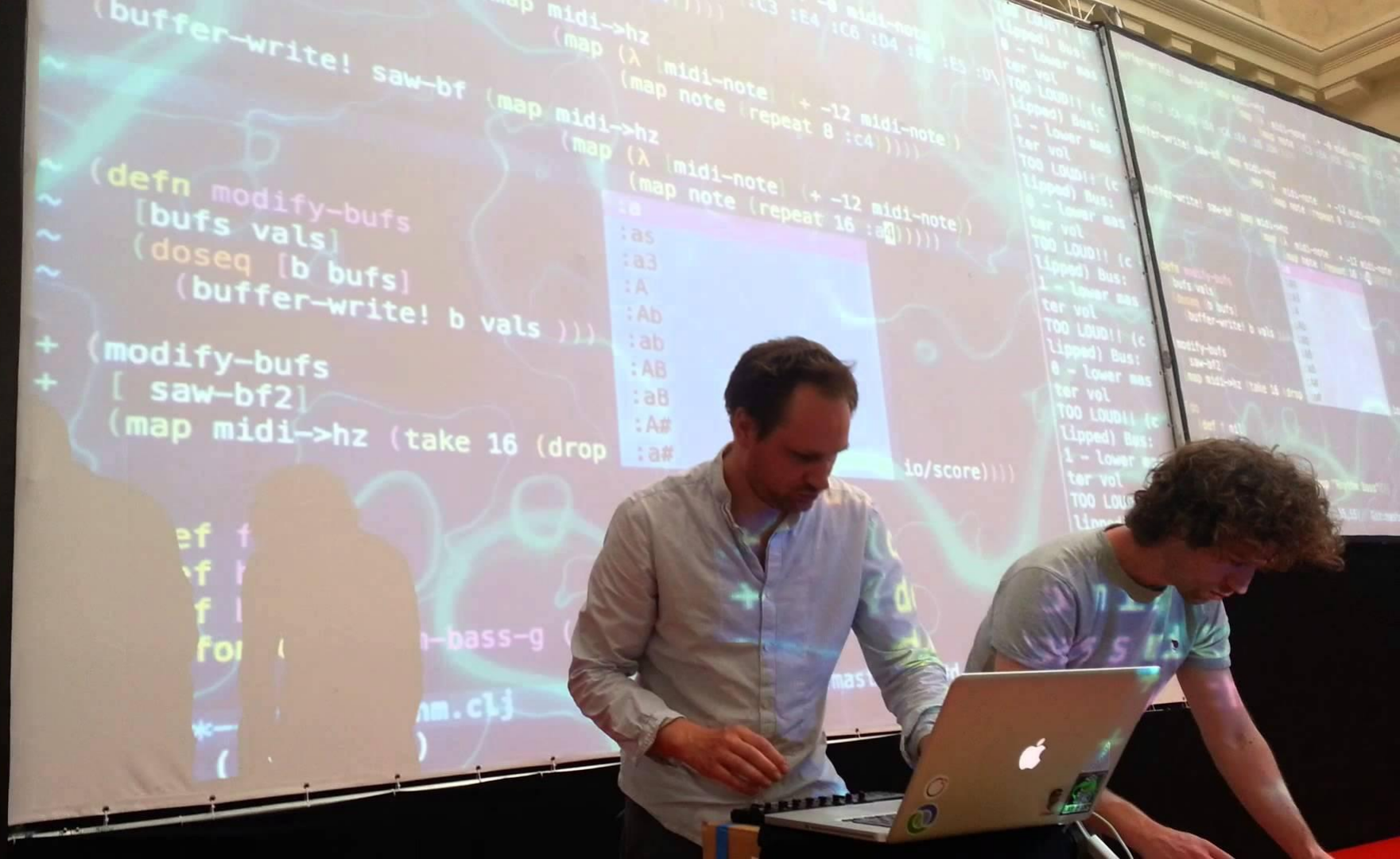
```
{:text "Hello there... ", :reverse? false, :gl {:p {:tag :program, :vertex-shader {:tag :sha
```

Canvas:

```
clojure.lang.ExceptionInfo : failed compiling file:src/cljs/gampg/core.cljs      client.js:1166
clojure.lang.ExceptionInfo : bindings must be vector of even number of           client.js:1166
elements at line 37 src/cljs/gampg/core.cljs
Figwheel: notified of file changes                                               client.js:1196
Figwheel: loaded these files                                               file_reloading.js:199
("/js/out/gampg/core.js")                                                  file_reloading.js:202
Figwheel: notified of file changes                                               client.js:1196
Figwheel: loaded these files                                               file_reloading.js:199
("/js/out/gampg/core.js")                                                  file_reloading.js:202
Figwheel: notified of file changes                                               client.js:1196
Figwheel: loaded these files                                               file_reloading.js:199
("/js/out/gampg/core.js")                                                  file_reloading.js:202
Figwheel: notified of file changes                                               client.js:1196
Figwheel: loaded these files                                               file_reloading.js:199
("/js/out/gampg/core.js")                                                  file_reloading.js:202
Figwheel: notified of file changes                                               client.js:1196
Figwheel: loaded these files                                               file_reloading.js:199
("/js/out/gampg/core.js")                                                  file_reloading.js:202
```

Overtone live performance - MetaX

```
        snare (+ snare (bpf (* 4 snare) 2000))]
    (clip2 snare 1))))


(defcgen wobble
  "wobble an input src"
  [src {:doc "input source"}
   wobble-factor {:doc "num wobbles per second"}]
  (:ar
   (let [sweep (lin-exp (lf-tri wobble-factor) -1 1 40 3000)
         wob   (lpf src sweep)
         wob   (* 0.8 (normalizer wob))
         wob   (+ wob (bpf wob 1500 2))]
     (+ wob (* 0.2 (g-verb wob 9 0.7 0.7)))))


(definst dubstep [bpm 120 wobble-factor 1 note 50]
 (let [freq (midicps (lag note 0.25))
       bass (apply + (saw (* freq [0.99 1.01])))
       bass (wobble bass wobble-factor)
       kick (kick-drum bpm :pattern [1 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0])
       snare (snare-drum bpm)]

   (clip2 (+ bass kick snare) 1)))

;;(dubstep)
;;(ctl dubstep :wobble-factor 3)
;;(stop)
-UU-:@----F2  composed_dubstep.clj   Bot (38,0)    Git-master  (Clojure -1-
```

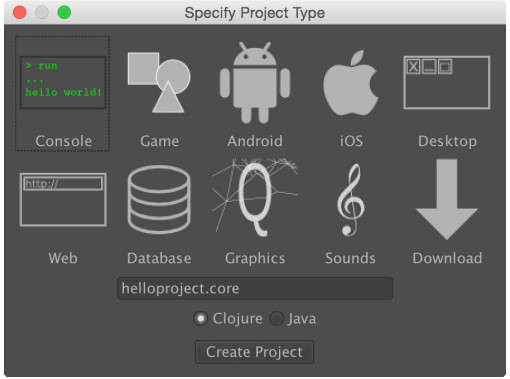

(buffer-write! saw-bf (map midi->hz
(map (λ [midi-note] (+ -12 midi-note))
(map note (repeat 8 :c4))))
(defn modify-bufs
[bufs vals]
(doseq [b bufs]
(buffer-write! b vals)))
(modify-bufs
[saw-bf2]
(map midi->hz (take 16 (drop
:a
:as
:a3
:A
:Ab
:ab
:AB
:aB
:A#
:a#

Examples, examples, examples

we learn by example...

Set up the most appropriate environment

Avoid drowning people in choices, suggest the most appropriate, based on what you know about them



Clojure-through-code Git repository

```clojure
;; start with (slurp ...) and what it returns is used as the argument to read-
string...

;; Get the contents of the project.clj file using `slurp`
;; Read the text of that file using read-string
;; Select just the third string using nth 2 (using an index starting at 0)


;; You can format the code differently, but in this case its not much easier t
o read
(nth
 (read-string
  (slurp "project.clj"))
 2)


;; the same behaviour as above can be written using the threading macro
;; which can make code easier to read by reading sequentially down the list of
 functions.

(->
 "./project.clj"
 slurp
 read-string
 (nth 2))

;; Using the threading macro, the result of every function is passed onto the
next function
;; in the list.  This can be seen very clearly usng ,,, to denote where the va
lue is passed
;; to the next function

(->
 "./project.clj"
 slurp ,,,
 read-string ,,,
 (nth ,,, 2))
```

```clojure
;; So we have a map where each value is itself a map.
(def dev-event-details
  {:devoxxuk     {:URL                  "http://jaxlondon.co.uk"
                  :event-type           "Conference"
                  :number-of-attendees  700
                  :call-for-papers      true}
   :hackthetower {:URL                  "http://hackthetower.co.uk"
                  :event-type           "hackday"
                  :number-of-attendees  60
                  :call-for-papers      false}})

;; lets call the data structre and see what it evaluates too, it should not be
 a surprise
dev-event-details

;; We can ask for the value of a specific key, and just that value is returned
(dev-event-details :devoxxuk)

;; In our example, the value returned from the :devoxxuk key is also a map,
;; so we can ask for a specific part of that map value by again using its key
(:URL (dev-event-details :devoxxuk))

;; Lets define a simple data structure for stocks data
;; This is a vector of maps, as there will be one or more company stocks
;; to track.  Each map represents the stock information for a company.

(def portfolio [ { :ticker "CRM" :lastTrade 233.12 :open 230.66}
                 { :ticker "AAPL" :lastTrade 203.25 :open 204.50}
                 { :ticker "MSFT" :lastTrade 29.12  :open 29.08 }
                 { :ticker "ORCL" :lastTrade 21.90  :open 21.83 }])

;; We can get the value of the whole data structure by refering to it by name
portfolio

;; As the data structure is a vector (ie. array like) then we can ask for a sp
ecific element by its position in the array using the nth function

;; Lets get the map that is the first element (again as a vector has array-lik
```

Creating Blogs, Tutorials & Workshops

Guiding people into ... | ClojureScript... by A... | Theory: Introducing ...

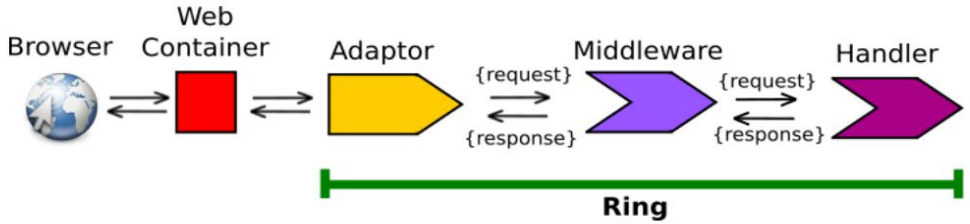
practicalli.github.io/clojure-webapps/introducing-ring/index.html

Introducing Ring

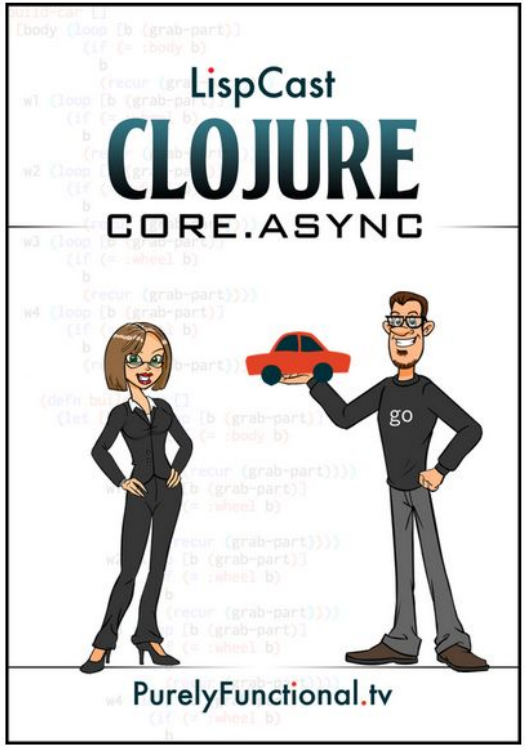
Web applications typically run on a web or application server, such as Tomcat or Jetty that provide a Java Servlet Container. The Ring library provides a way to use these servers without being tied to any specific implementation. Ring provides a common way to

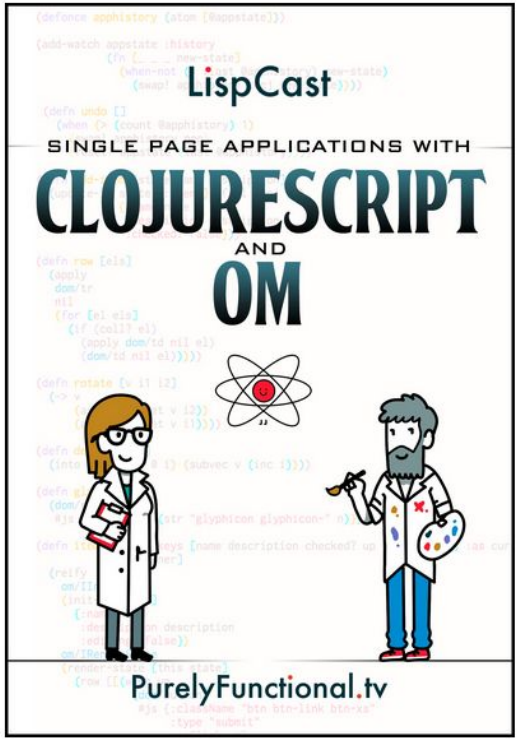
- Write your application using Clojure functions and maps
- Run your application in an auto-reloading development server (wrap-reload)
- Compile your application into a Java Servlet application
- Package your application into a Java war file
- Use a selection of middleware functions
- Deploy your application in cloud environments like Heroku

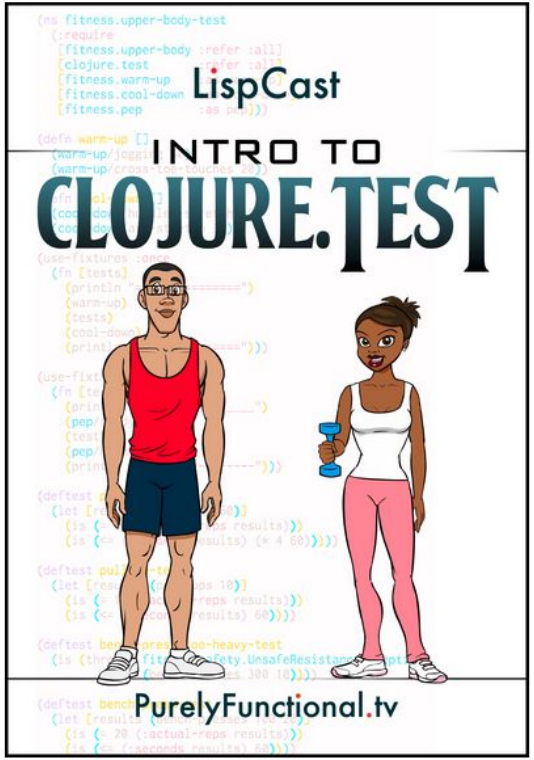
In essence, Ring converts the requests that come from the browser into a Clojure map, the request map. The request map may be passed through one or more middleware functions before being converted to a response map by a handler. The response map may be processed by one or more middleware functions before being converted by Ring to a web server response.



PurelyFunctional.tv







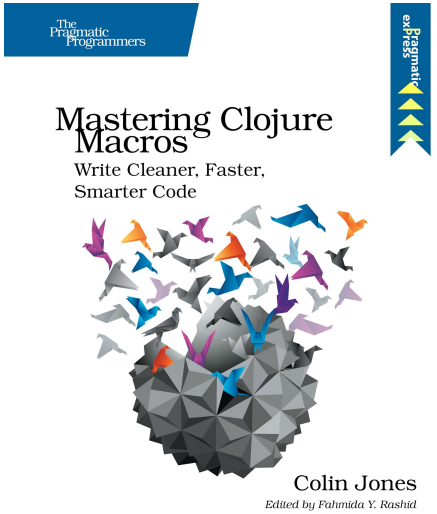
Keep it practical

we learn by doing...

Avoid “Death by theory overload”

There are many abstractions, design patterns, and concepts underlying Clojure that are all important to learn…

these are typically easier to understand through practice & specific application



Example:

Macros are really cool, but they are not something you need to master or fully understand in the first few months.

All the things practical...

CLOJURE KOANS

exercism.io

Give them different ways to learn

no one book, tool or technique fits all...

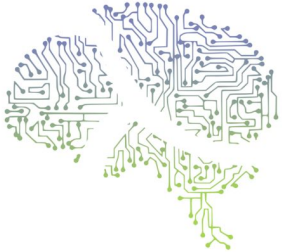
Over 20 Books on Clojure...

Where to start with Clojure will be different...

Example:

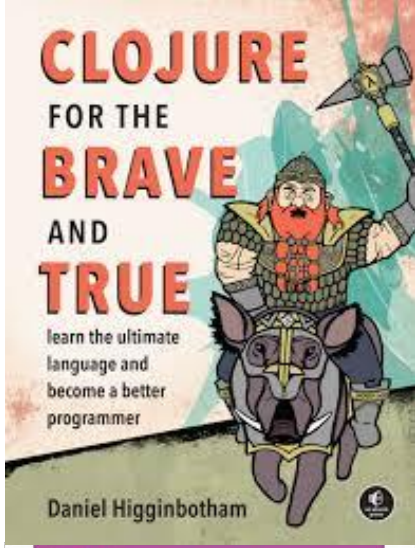
I typically suggested BraveClojure.com as a starting point, however many people prefer LivingClojure or ClojureScript Unraveled...

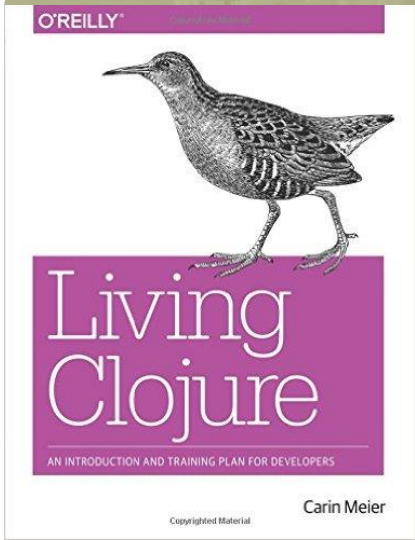
Help people understand the relevance of a book and if it's the right thing for them at that time.



ClojureScript
Unraveled

```
(& "Andrey Antukh"
   "Alejandro Gómez")
```





Engage them with the community

many positive voices provide an engaging experience & more opportunities to learn & discover

Clojure.org & ClojureDocs.org

Clojure

OVERVIEW REFERENCE API RELEASES GUIDES COMMUNITY NEWS

The Reader

Clojure is a homoiconic language, which is a fancy term descri
Clojure programs are represented by Clojure data structures.
important difference between Clojure (and Common Lisp) anc
programming languages - Clojure is defined in terms of the ev
structures and **not** in terms of the syntax of character streams
common, and easy, for Clojure programs to manipulate, trans
produce other Clojure programs.

That said, most Clojure programs begin life as text files, and it
reader to parse the text and produce the data structure the co
This is not merely a phase of the compiler. The reader, and th
representations, have utility on their own in many of the same
might use XML or JSON etc.

One might say the reader has syntax defined in terms of chara
Clojure language has syntax defined in terms of symbols, lists,
etc. The reader is represented by the function read, which rea
(not character) from a stream, and returns the object represe

Since we have to start somewhere, this reference starts where
starts, with the reader forms. This will inevitably entail talking

ClojureDocs

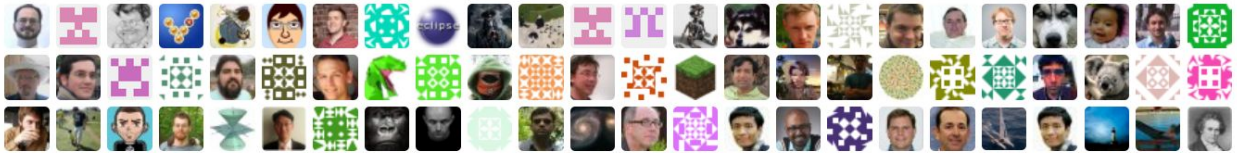
Core Library Quick Reference Log In Star 352

ClojureDocs is a community-powered documentation and examples repository for the Clojure programming language.

Looking for?

Can't find what you're looking for? Help make ClojureDocs better.

TOP CONTRIBUTORS



Migrate your old ClojureDocs account

Clojars

LOGIN REGISTER

Clojars is a **dead easy** community repository for open source Clojure libraries.

Search projects...

Search

To get started pushing your own project register and then check out the tutorial. Alternatively, browse the repository.

jr0cket
community developer

9 MAR 2015 · DEV-TOOLS

Clojure Templates Are Easy With Leinigen

Using templates to create your Clojure projects can save you a lot of setup time and ensure your team is using the same base configuration and dependencies. There are templates on Clojars.org, however I'll show you how easy it is to create your own with Leiningen.

I'll create a simple template based on the leiningen default template, adding a section in the project.clj to give a custom propmt when run in the repl.

> *Templates used to be a Leinigen pluging called lein-newnew and its repo was the only doucmentation I found and was a little outdated. The plugin is now part of Leiningen and there are a few built in templates. There is also information via* `lein help new`.

> *If you want to create a template in a more automatic way from a more complete project you created, take a look at the lein-create-template Leinignen plugin.*

Creating templates

A Clojure template is created in the same was as a Clojure project, however a template called `template` is used

```
lein new template your-template-name
```

Pull requests Issues Gist

Search

clojure

Search

Repositories	17,992
Code	1,105,706
Issues	29,969
Users	144

Languages

Clojure	14,277
JavaScript	710
Java	408
Emacs Lisp	176
CSS	152
Shell	149
Ruby	101
HTML	100
Python	86

We've found 17,992 repository results

Sort: **Best match**

clojure/ **clojure**

Java ★ 5,054 ⑂ 879

The **Clojure** programming language

Updated 23 days ago

LightTable/ **Clojure**

Clojure ★ 72 ⑂ 46

Light Table **Clojure** language plugin

Updated 17 days ago

functional-koans/ **clojure** -koans

Clojure ★ 2,341 ⑂ 1,369

A set of exercises for learning **Clojure**

Updated 18 days ago

Guiding people into ... | the force awakens - ... | 600_444462309.jpeg... | meta-x overtone - G... | Google Image Resul... | maxresdefault.jpg (... | Google Image Resul... | ldnclj | clojurians Slack

https://clojurians.slack.com/messages/ldnclj/

clojurians
jr0cket

STARRED
ldnclj

CHANNELS (223)
announce-to-everyone
beginners
boot
cider
clojure
clojurebridge
clojurescript
code-reviews
datomic
docs
editors
euroclojure
events
jobs
jvm
ldnproclodo
off-topic
other-lisps
overtone
proclodo-20-oct
proclodo-oct
slack-help

DIRECT MESSAGES (4838)

ldnclj

131 | Search

February 10th

malcolmsparks 11:11 AM
thanks @paulspencerwilliams , typo fixed

we have a project update coming out soon that provides a full cljs+sscomp dev env powered by a single build.boot

getting the tooling fixed is a big win

paulspencerwilliams 11:23 AM
agreed

mccraigmccraig 11:25 AM
ha, so really om/next ditches REST... thinking in terms of my current app, `send` could as easily be implemented over the websocket my app always has open

malcolmsparks 12:26 PM
yes, but websockets 😞

the point is to reduce all the various parts of your app all calling REST-based services

but I think this calls for additional aggregation services on the back-end, so more REST rather than less 😉

malcolmsparks 12:33 PM
to clarify, these Om Next Remotes (or state aggregation services for re-frame) all next to be quick and responsive, and chances are they'll have to make lots of calls out to other services (micro-services or not) - so HTTP conditional requests and cacheing and things like that are going to become more relevant

and clojure has a sweet-spot for cacheing, because of hash and immutable data structures in general - really good to produce e-tags for (the same argument why Om is faster than React, but at the backend)

mccraigmccraig 12:36 PM
ah, maybe i'm misunderstanding - i thought the om/next remotes were far coarser than a typical REST endpoint would be, encompassing whole graphs of objects, and therefore likely less cacheable...

but then i guess if the remote is an aggregation of further REST services, then e-tags etc would be important for those services

glenjamin 1:03 PM
my understanding is to decouple your components from your backend via query fragments, so you're free to fulfil them via REST or whatever

i've worked on React apps with a working set small enough to just keep it in the browser and continually sync via websocket

Om next is similar in flow to https://github.com/Day8/re-frame#subscribe - but with that declarative query layer, and an interpreter for it

> GitHub
> **Day8/re-frame**
> re-frame - A Reagent Framework For Writing SPAs, in Clojurescript.

Clojurian Community in Person

Probably the most active language-specific developer communities in London

Setting them free...

Letting them go on their own journey

practical.li

A collection of open books on software development

London, UK http://practical.li john@practical.li

Practicalli.github.io

Thank you

@jr0cket
jr0cket.co.uk