DataScript for Web Development
Nikita Prokopov
tonsky.me
Novosibirsk, Russia
Rationale

Server-side traditionally benefits from relational storage + queries
Why web developers should limit themselves?
Datomic model feels right and was within my reach
DataScript is a post-modern DB: someone else’s good idea redone on a more primitive level
What’s DataScript?

<entity, attribute, value>
triple store
Database as an immutable value
Completely in-memory
Datalog queries over it
Tries to mimic Datomic API
Written in ClojureScript, JS bindings available
(def conn (d/create-conn
  {:aka {:db/cardinality
         :db.cardinality/many}}))

(d/transact! conn [{:db/id 1
                      :name "Ivan"
                      :aka ["The Terrible", "Durak"]}])

(d/q '[:find ?name
       :where [?e :aka "Durak"
               [?e :name ?name]]
      @conn)
How it works?
(defn create-conn [schema]
  (atom (empty-db schema)
    :meta { :listeners (atom {}) })))
(defrecord DB [ schema eavt aevt avet max-eid max-tx ])
(defn with [db datoms]
  (→ db
  (update-in [:eavt] into datoms)
  (update-in [:aevt] into datoms))
(defn transact [conn datoms]
  (swap! conn with datom))
BTSet
In-house implementation of B+ tree (sorted set)
Performance comparable with `sorted-set`:

~ 50% slower `conj`

~200% faster `iterate`
Binary search lookups,
fast range scans,
reverse iteration,
fast first-time creation
ClojureScript, but heavy use of JS arrays and APIs
DataScript is lightweight

700 loc btset
550 loc query engine
1700 loc total
1200 loc tests
Unfortunate to be associated with the word “database”:
no networking,
query over memory,
no log,
no history & compound indexes,
no external segments,
constant space operation
DataScript is more of a persistent data structure.

It’s just three sorted sets, literally.
How to use it

Every app has an ad-hoc state
Put everything in a database
Non-trivial SPA has complex state
KV stores do not cut it
No inherent hierarchy, very natural for any data: sparse, irregular, hierarchical, graph
Faster data retrieval from big datasets: indexes + hash joins
Uniform solution for storage encourages decoupling and reusable components:
server sync, undo/redo, local caching, audit
Datoms, DB and transactions have standard way to be serialized
No need to invent format for delta exchange
Immutability
+ db as a value
+ transactions
= always consistent render
Encourages to write decoupled, pure renders

Good fit for React and Flux
Example architecture: CatChat

github/tonsky/datascript-chat
I don't belong here, we gotta move on dear, escape from this...
Raw react render, pure, decoupled
Re-render triggered by database mutations, always top-down, full re-render
“Server” sync, db cleanup all decoupled from render code
Operates in a bound space via cleanup
Second DB used to emulate server
Example architecture: Acha-Acha

acha-acha.co
Achievements

Save the Day
Use word “fix” in a commit message
clojure/38d7572

Flash
Two different commits within 15 seconds
clojure/03cd9d1

Write Once. Run. Anywhere
Add Java file to the repo
clojure/2a09172

From Russia with Love
Commit on Russia Day
clojure/7806e33

Lucky
Consecutive 777 in SHA-1
clojure/8b07773

Owl
Commit between 4am and 7am local time

Rich Hickey
richhickey@gmail.com
Whole DB prefetch:
no server fetch on any navigation, all queries and aggregations happen on a client
Always up-to-date server sync: initial dump + deltas via websocket (tricky to do correctly)
Non-hierarchical layout:
user page, repo page, main page
14K datoms in 100..150ms
Example architecture: Menu [WIP]
github/tonsky/datascript-menu
**Guests**

- **Matilda Young**
  - Order #98406
  - Pan Fried Sea Bass Fillet
  - Chicken And Mushroom Pie
  - Fish And Chips

- **Thomas Mitchell**
  - Order #98406
  - Pan Fried Sea Bass Fillet
  - Chicken And Mushroom Pie
  - Fish And Chips

- **Archie Davis**
  - Order #98406
  - Pan Fried Sea Bass Fillet
  - Chicken And Mushroom Pie
  - Fish And Chips

- **Lola Baker**
  - Order #45355
  - Pan Fried Sea Bass Fillet
  - Fish And Chips

---

**Menu**

- Roasted Pork Belly
- Beef Stew
- Grilled Salmon Fillet
- Higgledy Veggie Pie (v)
- Gnocchi Sorento Style (v)
- Chicken Breast Milanese Style
- Pan Fried Sea Bass Fillet
- Chicken And Mushroom Pie
- Classic Beef Burger
- Fish And Chips
Component can listen for a specific [e a v] pattern
Optimized for huge amount of listeners, e.g. every person component ever can subscribe to
[(:db/id person) :person/name _]
Similar to Om ref-cursors
Have to figure out how to play nicely with top-down rendering
Project status
Considered alpha stage
No docs, must know/learn Datomic
Extensive acceptance tests suite helps
Almost no validation, silent fails/unexpected results instead
Occasional breaking changes (not significant though)

If it’s in Datomic and DataScript already, it probably won’t change
Future plans

Cover all Datomic APIs

components
lookup refs
find specifications
pull API
Entry barrier: docs, tutorials, error reporting
JVM Clojure port
Thanks! — Questions?

Nikita Prokopov

github/tonsky/datascript
tonsky.me