DATA ALL THE WAY DOWN: AN INTRO TO DATASCRIPT & DATAHIKE

Created: 2022-10-25 Tue 10:13

IT'S ALL JUST VERY WET CRUD

POSTGRES

```
CREATE TABLE users (
  id INTEGER PRIMARY KEY,
  first_name varchar(255) NOT NULL,
  last_name varchar(255) NOT NULL,
  email varchar(255) NOT NULL
);

INSERT INTO users (first_name, last_name, email)
VALUES ('Bilbo', 'Baggin');

SELECT first_name, last_name
FROM users
WHERE email = 'bilbo@the-shire.com';

UPDATE users
```

CLOJURE

```
(ns middle-earth.db.users
  (:require [clojure.set    :as set]
            [triangulum.logging  :refer [log]]
            [triangulum.database :refer [call-sql
                                         sql-primitive
                                         p-insert-rows!
                                         insert-rows!]]
            [triangulum.type-conversion :as tc]
            [triangulum.utils           :as u]))

(defn get-user [email]
  (-> (call-sql "get_user" email)
      (set/rename-keys {:first_name :first-name
                        :last_name  :last-name})))
```

CLOJURESCRIPT

```
(ns middle-earth.users
  (:require [middle-earth.utils :refer [call-clj-async]]))

(defn add-user [email first-name last-name]] ...)

(defn update-user [user-id {:keys [email first-name last-name]

(defn get-user [email]
  (call-clj-async "get-user" email))

(defn delete-user [user-id] ...)
```

JAVASCRIPT/REACT

```
function get_user (email) {
  fetch("/user?email=" + email)
    .then(p => ...);
}

function create_user (email, first_name, last_name) {
  fetch("/user", {method: "POST",
                  body: JSON.stringify({'email': email,
                                        'first_name': first_na
                                        'last_name': last_name
    .then(p => ...);
}

function update_user (user_id, first_name, last_name, email) {
```

OVERVIEW

- 1 SQL Definition
- 4 SQL Functions
- 4 Clojure -> SQL Functions
- 4 API Routes
- 4 CLJS/JS -> API Functions
- N CLJS/JS Data Validation Functions
- M CLJS/JS Data 'Massage' Functions to State/View
- 1 'Concept', 16+N+M Functions, 50 lines min.

HOW TO STAY DRY

- Query data close to the point of use.
- De-complect State management from UI components
- Eliminate manual 'data shuffling'.

DATALOG

- Declarative logic based on a subset of Prolog.
- Stores data in the form of 'triples' (Subject-Predicate-Object / Entity-Attribute-Value).
- Enables formal reasoning/deduction based on data (not covered).

DATA FORMAT

- Triple Format: `[<subject> <predicate> <object>]`
- Datalog Format: `[<entity-id> <attribute> <value>]`

ACCUMULATION OF FACTS

```
[42 :user/first-name "Bilbo"]
[42 :user/last-name "Baggins"]
[42 :user/email "bilbo@the-shire.com"]

[43 :user/first-name "Samwise"]
[43 :user/last-name "Gamgee"]
[43 :user/email "sam@the-shire.com"]

;; Make Samwise a friend of Frodo
[43 :friend 42]
```

QUERIES & UNIFICATION

```
[?e :user/last-name "Baggins"]
[?e :user/first-name ?first-name]

[?e1 :user/email "sam@the-shire.com"]
[?e1 :friend ?e2]
[?e2 :user/first-name ?first-name]
```

DATALOG BENEFITS

- Schema is not set in stone.
- Can represent collections / high cardinality data out of the box.
- Query language enables `JOIN`'s without multiple tables.

DATASCRIPT

Datalog in the Browser. (DataScript Repo)

DATASCRIPT CRUD

```
(ns middle-earth.data
  (:require [datascript.core :as d]))

(def db (d/conn))

; CREATE
(d/transact db [{:user/first-name "Bilbo"
                 :user/last-name  "Baggin"
                 :user/email      "bilbo@the-shire.com"}])

; READ - Query
(d/q '[:find  ?first-name ?last-name
       :in    $ ?email
       :where [?e :user/email ?email]
              [?e :user/first-name ?first-name]
```

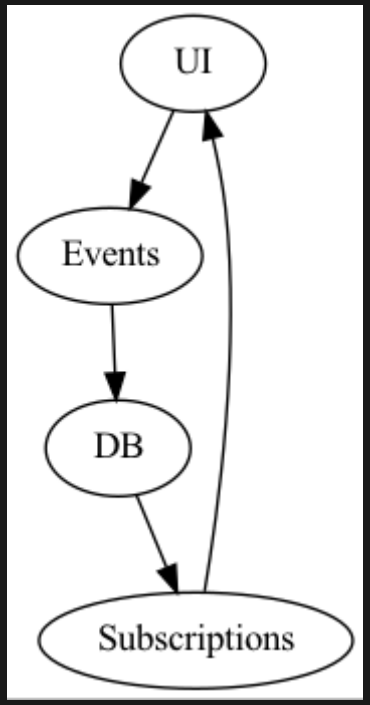
DE-COMPLECTING YOUR UI

- State management and UI components become tightly coupled
- One application -> 20 different ways to manage state + UI
- Copying over re-usability

THE STATE OF (MOST) STATE

```
(ns middle-earth.hobbits
  (:require [reagent.core :as r]
            [middle-earth.utils :refer [call-clj-async!]]))

(def ^:private the-hobbits (r/atom []))

(defn- get-all-hobbits []
  (reset! the-hobbits (call-clj-async! "get-all-hobbits")))

(defn show-hobbits []
  [:div
   [:button {:label    "Gather the Hobbits"
             :on-click get-all-hobbits}]
   [:h1 "The Hobbits"]
   [:ul
```

USING RE-FRAME TO MANAGE EVENTS/STATE



RE-FRAME: SUBSCRIPTIONS

```
(ns middle-earth.subs
  (:require '[re-frame.core :as rf]))

(rf/reg-sub
 :hobbits/all
 (fn [db _]
   (:hobbits db)))
```

RE-FRAME: EVENTS

```
(ns middle-earth.events
  (:require [re-frame.core :as rf]
            [middle-earth.utils :refer [call-clj-async!]]))

(rf/reg-event-db
 :hobbits/create
 (fn [db [_ {:keys [email] :as hobbit}]]
   (assoc-in db [:hobbits email] hobbit)))

(rf/reg-event-db
 :hobbits/pull-from-db
 (fn [db _]
   (->> (call-clj-async! "get-hobbits")
        (assoc db :hobbits))))
```

COMPONENT W/ RE-FRAME

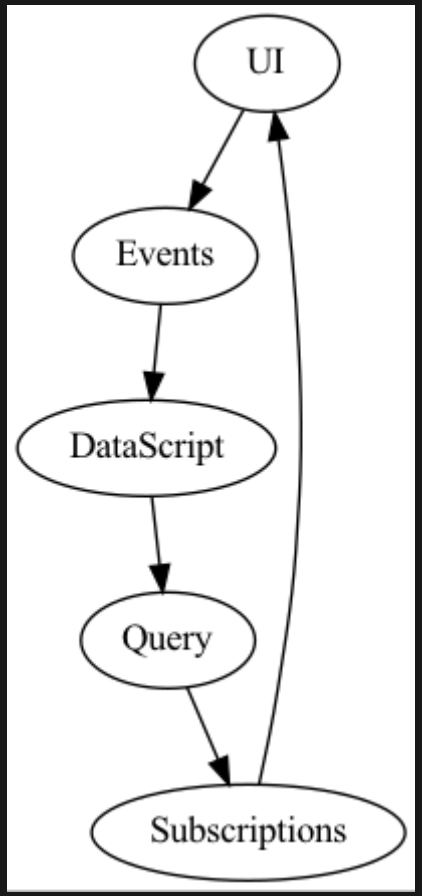
```
(ns middle-earth.hobbits
  (:require [re-frame.core :as rf]
            [middle-earth.utils :refer [call-clj-async!]]))

(defn show-hobbits []
(let [the-hobbits (rf/subscribe [:hobbits/all])])
  [:div
   [:button {:label    "Gather the Hobbits"
             :on-click #(rf/dispatch [:hobbits/pull-from-db])}
   [:h1 "The Hobbits"]
   [:ul
    (for [email {:keys [first-name last-name]} @the-hobbits]
      [:a {:href (str "mailto:" email)}
       [:li (str first-name last-name)]])]])
```

BENEFITS OF RE-FRAME

- One "source of truth" for state (subscriptions), and managing state (events)
- Separate the UI from the "Data flow"

RE-POSH: RE-FRAME + DATASCRIPT

- Subscribe to a DataScript DB using queries
- Events modify the DataScript DB


UI
Events
DataScript
Query
Subscriptions

RE-POSH: SUBSCRIPTIONS W/ DATASCRIPT

```
(ns middle-earth.subs
  (:require [re-posh.core as rp]))

(rp/reg-sub
 :hobbits/all
 (fn [_ _]
   {:type  :query
    :query '[:find ?email ?first-name ?last-name
             :where [?e :user/first-name ?first-name]
                    [?e :user/last-name ?last-name]
                    [?e :user/email ?email]]}))
```

RE-POSH: EVENTS W/ DATASCRIPT

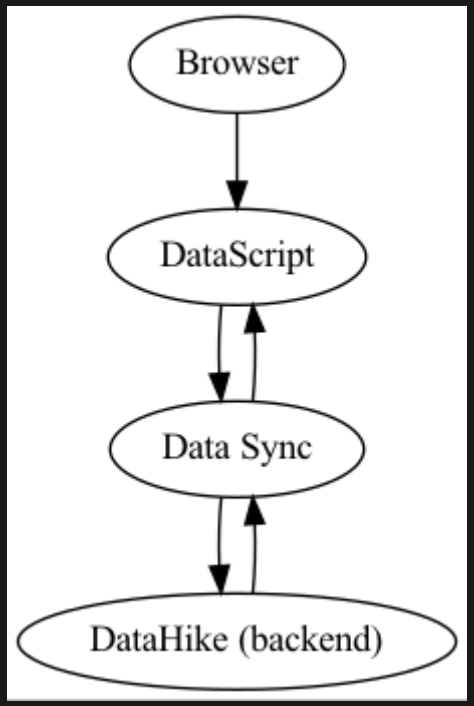
```
(ns middle-earth.subs
  (:require [re-frame.core :as rf]
            [re-posh.core as rp]))

(rp/reg-event-fx
 :hobbits/create
 (fn [_ [_ hobbit]]
   {:transact [hobbit]}))

(rf/reg-event-fx
 :hobbits/pull-from-db
 (fn [db _]
   {:transact (call-clj-async! "get-hobbits")}))
```

COMPONENT W/ RE-POSH

```
(ns middle-earth.hobbits
  (:require [re-frame.core :as rf]))

(defn show-hobbits []
(let [the-hobbits (rf/subscribe [:hobbits/all])])
  [:div
   [:button {:label    "Gather the Hobbits"
             :on-click #(rf/dispatch [:hobbits/pull-from-db])}
   [:h1 "The Hobbits"]
   [:ul
    (for [[email first-name last-name]} @the-hobbits]
      [:a {:href (str "mailto:" email)}
       [:li (str first-name last-name)]])]])
```

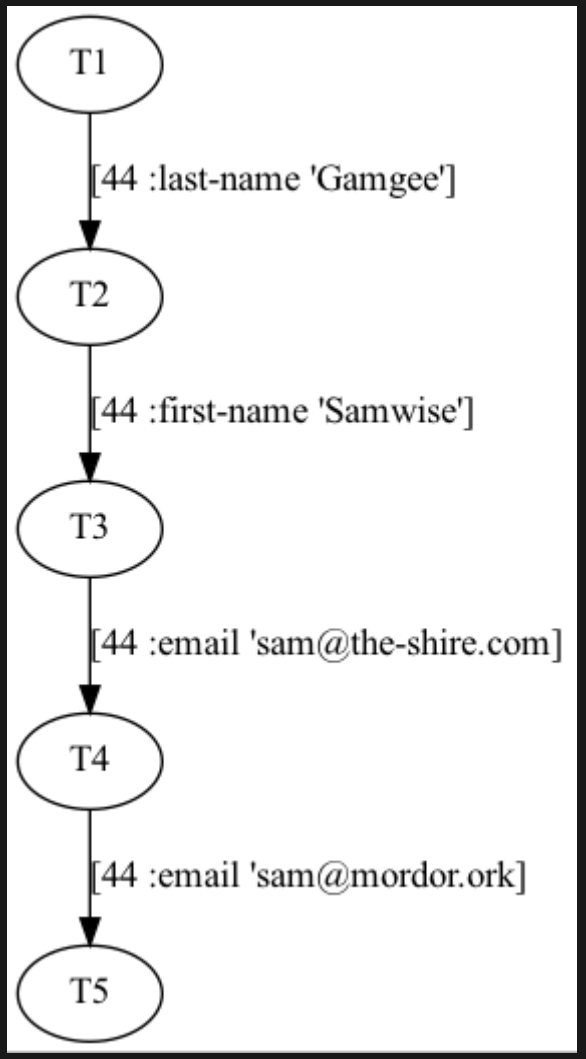
ELIMINATE MANUAL DATA SHUFFLING

BehavePlus App Achitecture



DATA SYNC

- Common data format (`[entity attribute value tx-id operation]`)
- Ordered transactions with timestamps


T1
[44 :last-name 'Gamgee']
T2
[44 :first-name 'Samwise']
T3
[44 :email 'sam@the-shire.com]
T4
[44 :email 'sam@mordor.ork]
T5

GETTING DRY

- Start with Re-Frame (90%)
- Look to DataScript when you're noticing deeply nested maps (10%)
- Data Sync (R&D)

Q & A