

Reagent

a ClojureScript interface to React



React Amsterdam Meetup
12 Feb. 2015

Michiel Borkent

Twitter: [@borkdude](https://twitter.com/borkdude)

Email: michielborkent@gmail.com

- Clojure(Script) developer at  **FINALIST**
open IT oplossingen
- Clojure since 2009
- Former lecturer, taught Clojure

Full Clojure stack example @ Finalist

Commercial app.

Fairly complex UI

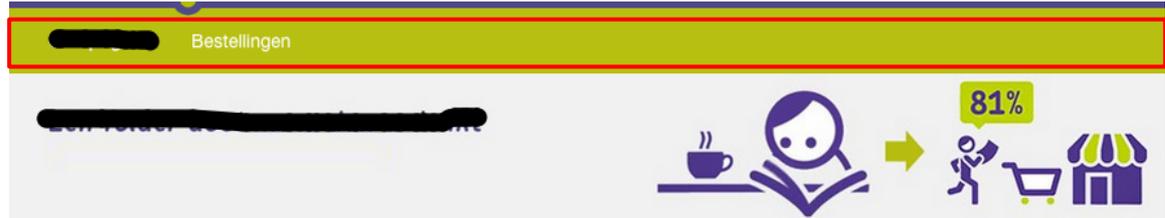
- Menu: 2 "pages"

Page 1:

Dashboard. Create new or select existing entity to work on.

Then:

- Wizard 1
 - Step 1..5
 - Each step has a component
- Wizard 1 - Step2
 - Wizard 2
 - Step 1'
 - Step 2'



← Terug naar dashboard

Full Clojure stack examples @ Finalist

Step 2 of inner wizard:

- Three dependent dropdowns + backing ajax calls
- Crud table of added items + option to remove
- When done: create something based on all of this on server and reload entire "model" based on what server says

Because of React + Om we didn't have to think about updating DOM performantly or keeping "model" up to date.

Bestellingen

Step 2.

Omschrijving Bla
Dataset

Na het toevoegen van één of meerdere variabelen kan de regel worden opgeslagen.

Categorie
type

Subcategorie
type

Variabele
type

Voeg toe

Hoofdcategorie	Subcategorie	Beschrijving	
Automotive			✕
		gezin jongste ki	✕

Opslaan

Agenda

- Intro
- A little Clojure syntax
- Hiccup
- ClojureScript atoms
- Reagent

Syntax

$f(x) \rightarrow (f\ x)$

Syntax

```
if (...) {  
    ...  
} else {  
    ...  
}
```

->

```
(if . . .  
    . . .  
    . . .)
```

Data literals

Symbol: :a

Vector: [1 2 3 4]

Hash map: {:a 1, :b 2}

Set: #{1 2 3 4}

List: '(1 2 3 4)

Hiccup

```
[ :a { :href "/logout" }  
  "Logout" ]
```

```
<a href="/logout">Logout</a>
```

```
[ :div#app.container  
  [ :h2 "Welcome" ] ]
```

```
<div id="app" class="container">  
  <h2>Welcome</h2>  
</div>
```

ClojureScript atoms

```
(def my-atom (atom 0))  
@my-atom ;; 0  
(reset! my-atom 1)  
(reset! my-atom (inc @my-atom)) ;; bad idiom  
(swap! my-atom (fn [old-value]  
                 (inc old-value)))  
(swap! my-atom inc) ;; same  
@my-atom ;; 4
```

Reagent

ClojureScript library around React

Uses RAtoms for state (global or local)

Components are 'just functions'TM that

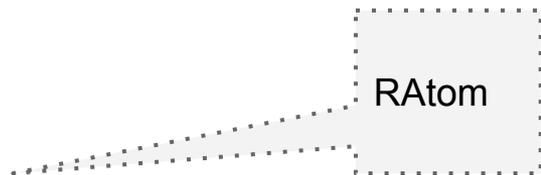
- **must** return something renderable by React
- **can** deref RAtom(s)
- **can** accept props as args
- **may** return a closure, useful for setting up initial state



Reagent

- Components should be called like
[component args] instead of
(component args)
- Components are re-rendered when
 - props (args) change
 - referred RAtoms change
- Hook into React lifecycle via metadata on component functions

```
(def component
  (with-meta
    (fn [x]
      [:p "Hello " x ", it is " (:day @time-state)])
    {:component-will-mount #(println "called before mounting")
     :component-did-update #(.focus (reagent/dom-node %))} ))
```



RAtom

```
(def count-state (atom 10))
```

```
(defn counter []  
  [:div  
    @count-state  
    [:button {:on-click #(swap! count-state inc)}  
      "x"]])
```

```
(reagent/render-component [counter]  
  (js/document.getElementById "app"))
```

10



```
(defn local-counter [start-value]
  (let [count-state (atom start-value)]
    (fn []
      [:div
       @count-state
       [:button {:on-click #(swap! count-state inc)}
        "x"]]))))
```

local
RAtom

10



```
(reagent/render-component [local-counter 10]
  (js/document.getElementById "app"))
```

CRUD!

Name	Species		
Aardwolf	Proteles cristata	Edit	×
Atlantic salmon	Salmo salar	Edit	×
Curled octopus	Eledone cirrhosa	Edit	×
Dung beetle	Scarabaeus sacer	Edit	×
Gnu	Connochaetes gnou	Edit	×
Horny toad	Phrynosoma cornutum	Edit	×
Painted-snipe	Rostratulidae	Edit	×
Yellow-backed duiker	Cephalophus silvicultor	Edit	×
<input type="text"/>	<input type="text"/>	Add	

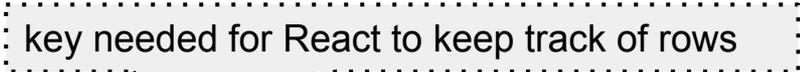
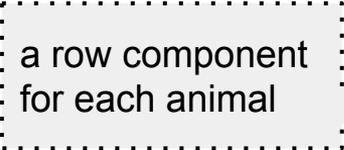
RAtom with set containing
animal hash-maps

```
(def animals-state (atom #{}))
```

```
(go (let [response  
        (<! (http/get "/animals"))  
        data (:body response)]  
    (reset! animals-state (set data))))
```

```
(...  
  {:id 2,  
   :type :animal,  
   :name "Yellow-backed duiker",  
   :species "Cephalophus silvicultor"}  
  {:id 1,  
   :type :animal,  
   :name "Painted-snipe",  
   :species "Rostratulidae"})
```

Render all animals from state

```
(defn animals []  
  [:div  
    [:table.table.table-striped  
      [:thead  
        [:tr  
          [:th "Name"] [:th "Species"] [:th ""] [:th ""]]]  
      [:tbody  
        (map (fn [a]  ^{:key (str "animal-row-" (:id a))}  
           [animal-row a])  
        (sort-by :name @animals-state)   
        [animal-form]]]]))
```

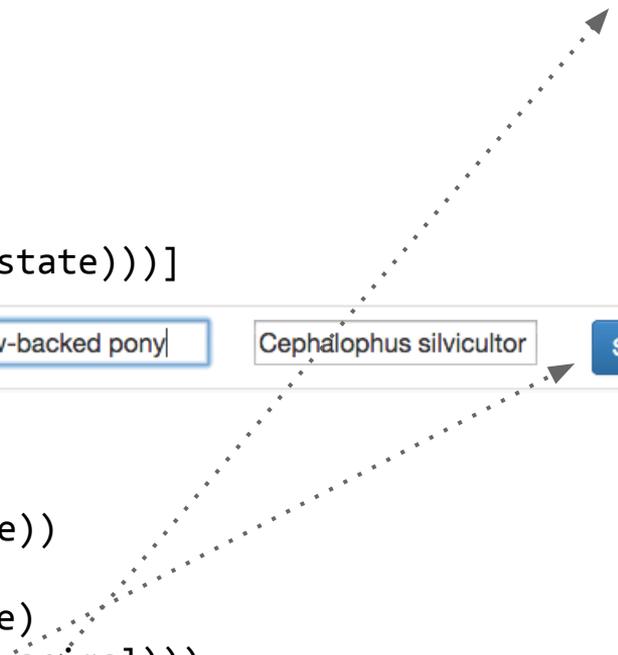
```

(defn animal-row [a]
  (let [row-state (atom {:editing? false
                        :name      (:name a)
                        :species   (:species a)})]
    current-animal (fn []
                     (assoc a
                            :name (:name @row-state)
                            :species (:species @row-state))))
    (fn []
      [:tr
       [:td [editable-input row-state :name]]
       [:td [editable-input row-state :species]]
       [:td [:button.btn.btn-primary.pull-right
             {:disabled (not (input-valid? row-state))
              :onClick (fn []
                        (when (:editing? @row-state)
                          (update-animal! (current-animal)))
                        (swap! row-state update-in [:editing?] not))}
             (if (:editing? @row-state) "Save" "Edit")]]]
       [:td [:button.btn.pull-right.btn-danger
             {:onClick #(remove-animal! (current-animal))}
             "\u00D7"]]]])))

```

Yellow-backed duiker	Cephalophus silvicultor	<button>Edit</button>	<button>×</button>
----------------------	-------------------------	-----------------------	--------------------

<input type="text" value="Yellow-backed pony"/>	<input type="text" value="Cephalophus silvicultor"/>	<button>Save</button>	<button>×</button>
---	--	-----------------------	--------------------



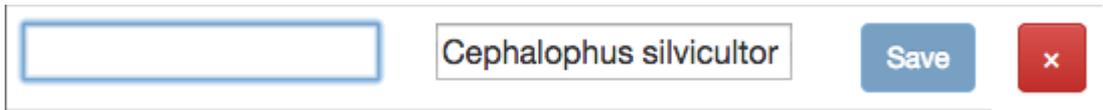
```
(defn field-input-handler
```

```
  "Returns a handler that updates value in atom map,  
  under key, with value from onChange event"
```

```
  [atom key]
```

```
  (fn [e]
```

```
    (swap! atom  
          assoc key  
          (.. e -target -value))))
```

A screenshot of a web form. It features a text input field with a blue border, a button with the text "Cephalophus silvicultor", a blue "Save" button, and a red "x" button. A dotted arrow points from the "input-valid?" function in the code below to the input field.

```
(defn input-valid? [atom]
```

```
  (and (seq (-> @atom :name))  
        (seq (-> @atom :species))))
```

```
(defn editable-input [atom key]
```

```
  (if (:editing? @atom)  
      [:input {:type "text"  
              :value (get @atom key)  
              :onChange (field-input-handler atom key)}]  
      [:p (get @atom key)]))
```

```
(defn remove-animal! [a]
  (go (let [response
           (<! (http/delete (str "/animals/"
                               (:id a))))]
      (if (= (:status response)
              200)
          (swap! animals-state remove-by-id (:id a)))))))
```

if server says:
"OK!", remove
animal from
CRUD table

```
(defn update-animal! [a]
  (go (let [response
           (<! (http/put (str "/animals/" (:id a))
                        {:edn-params a}))
          updated-animal (:body response)]
      (swap! animals-state
              (fn [old-state]
                (conj
                 (remove-by-id old-state (:id a))
                 updated-animal)))))))
```

replace updated
animal retrieved
from server

Code and slides at:

<https://github.com/borkdude/react-amsterdam>

Learn more at <https://github.com/reagent-project>