# CLOJURE WEB APP

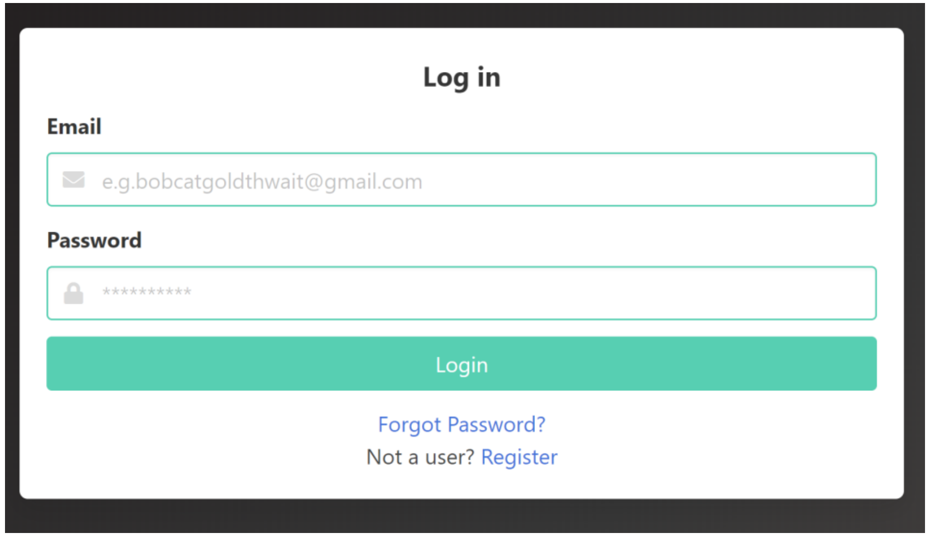
Max Coursey

Vanderbilt 6385, Summer 2020

# Overview

Post App is a “sort of” single page application with some server-side rendering for authentication/authorization and a single-page main post page

Utilizes the re-frame/re-agent framework with a mongo backend



Log in

Email

e.g.bobcatgoldthwait@gmail.com

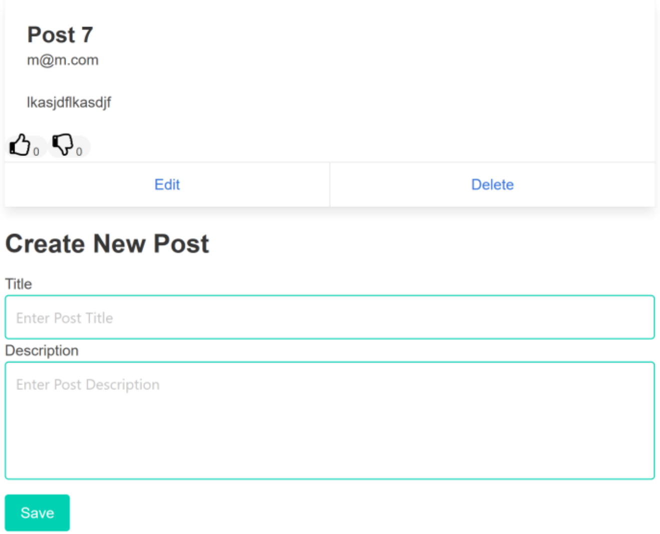
Password

**********

Login

Forgot Password?

Not a user? Register



Post 7

m@m.com

lkasjdflkasdjf

👍0 👎0

Edit Delete

Create New Post

Title

Enter Post Title

Description

Enter Post Description

Save

# Modules/Libraries

- Luminus Template – Leiningen Template
- Re-frame – UI clojurescript framework
- Ring – Clojure HTTP abstraction module
- Reitit Router - clojure(script) based router module
- Buddy Auth – authorization/authentication module for ring based apps
- Monger – MongoDB clojure client module
- Muuntaja – HTTP API clojure library
- Hiccup – HTML formatting for clojure
- Bulma – CSS formatting

# Functionality

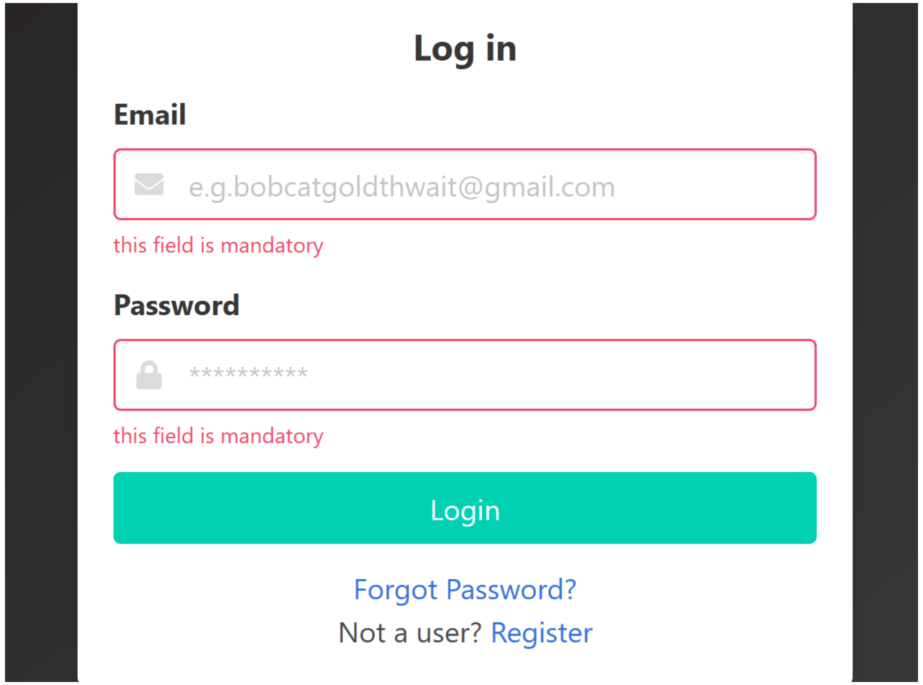
- Register
  - Registers new emails with password (encrypted utilizing Buddy, bcrypt)
- Login Page
  - Allow authentication with unique email/password combination
- Forgot Password
  - Generates automatic email with reset password link (partially implemented – gmail auth issue)
- Post Page
  - Retrieves posts from mongodb, subscribes to changes in posts utilizing front-end re-frame app-db
  - Allows creation of new posts by user
  - Allows up/down votes

# Challenges

- Re-Frame Framework
  - Front-end perpetual data loop = domino-type effects (Event dispatch, Event handling, Effect handling, Query, View, DOM)
    - Hard to get up to speed on dispatch event listeners, how they impact the app-db, when to subscribe to changes
  - Utilizes app-db (really just nested map structure for all data on front-end)
    - Hard to understand what state the app-db is currently in
    - Hard to error check the data
  - Clojurescript
    - Figwheel helped frontend, but error checking and understanding nature of errors were "backward" looking
      - As opposed to clojure back end, where I could error check as I programmed
    - Figwheel automatically reloaded – but still 5-15 second lag for each change made
  - HTTP request
    - Since app-db in re-frame was difficult to understand – formatting data correctly request/responses to backend was challenging

# Learned/Next

- Develop more functionality for posts by user, category, etc.
- Develop profile page with more user information
- Develop front end SPA changes
  - Allow immediate updates to UI when user requests (in app-db)
  - Send changes to backend after UI has already updated.
  - Understand relationship between app-db/mong db changes better
- Utilize re-frame's app-db to make UI more dynamic (toggle modals, colors, etc)
- Allow user customization of colors and components
- Expand Schema & testing to posts and more robust testing on user profile/posts
- How to speed up load time



Log in

Email

e.g.bobcatgoldthwait@gmail.com

this field is mandatory

Password

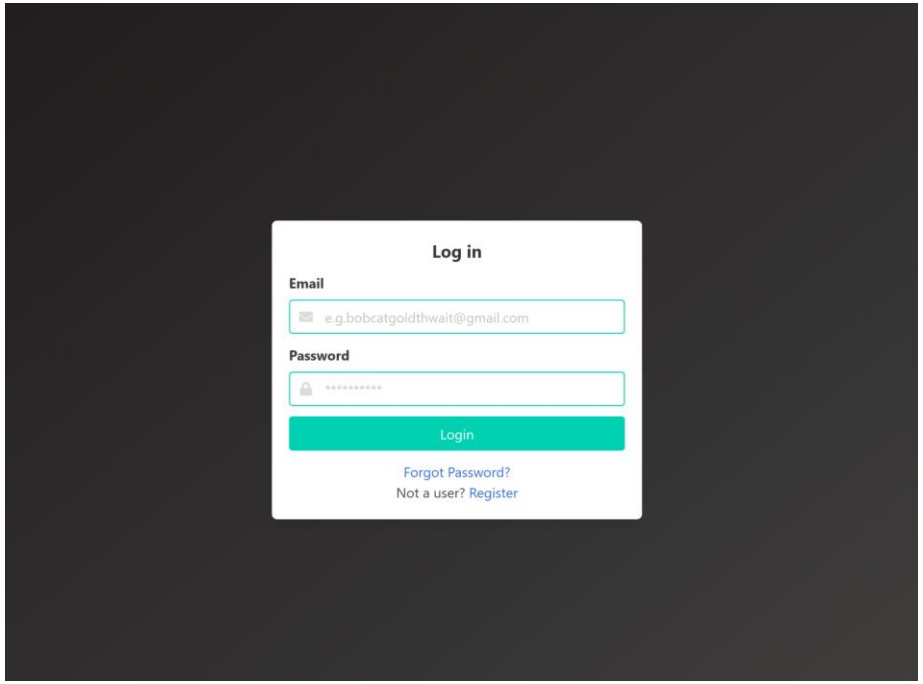
**********

this field is mandatory

Login

Forgot Password?

Not a user? Register



Log in

Email

e.g.bobcatgoldthwait@gmail.com

Password

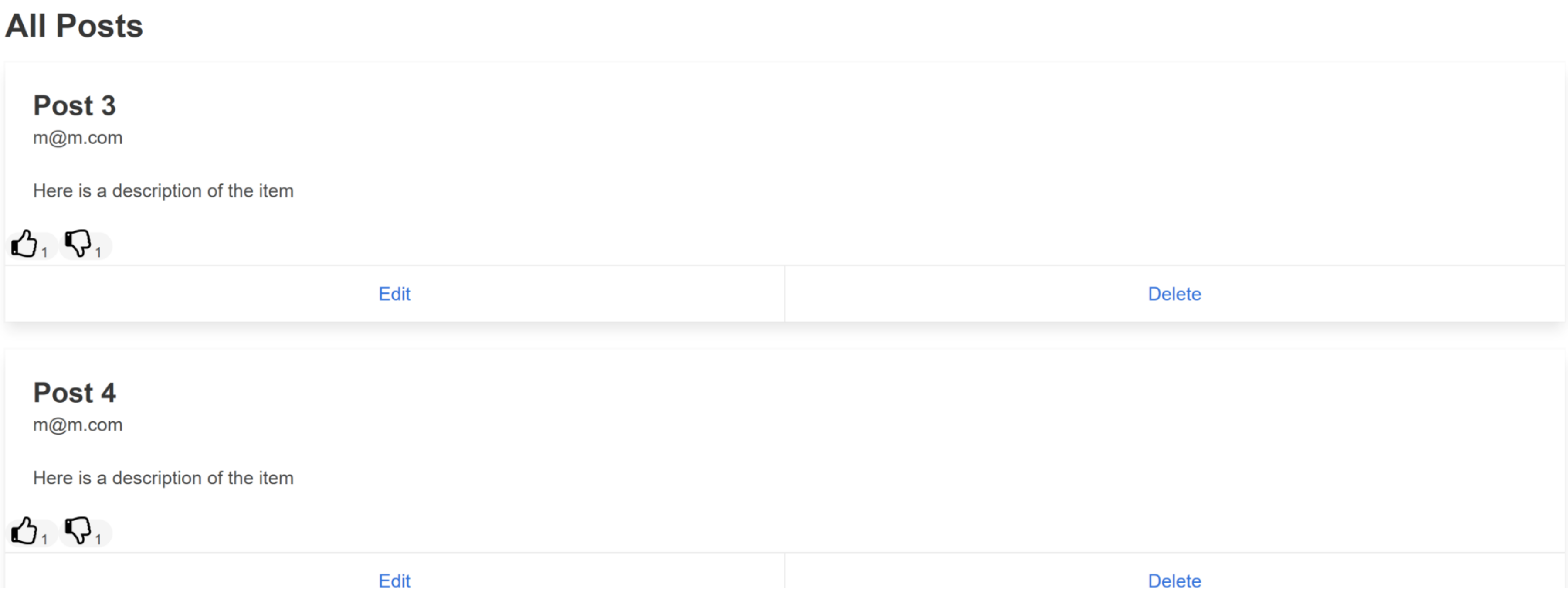
**********

Login

Forgot Password?

Not a user? Register

# SCREENSHOTS



All Posts

Post 3

m@m.com

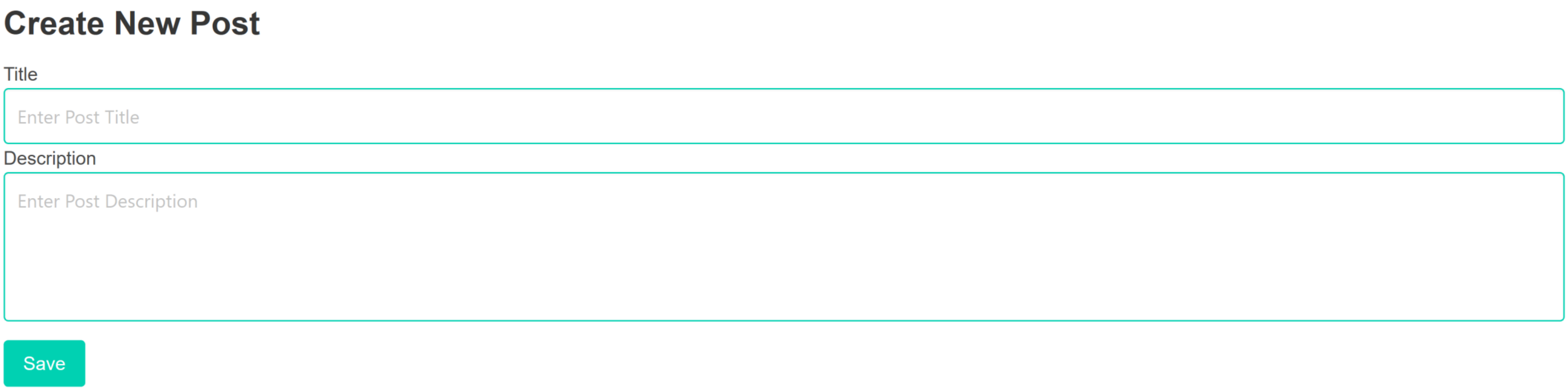
Here is a description of the item

Post 4

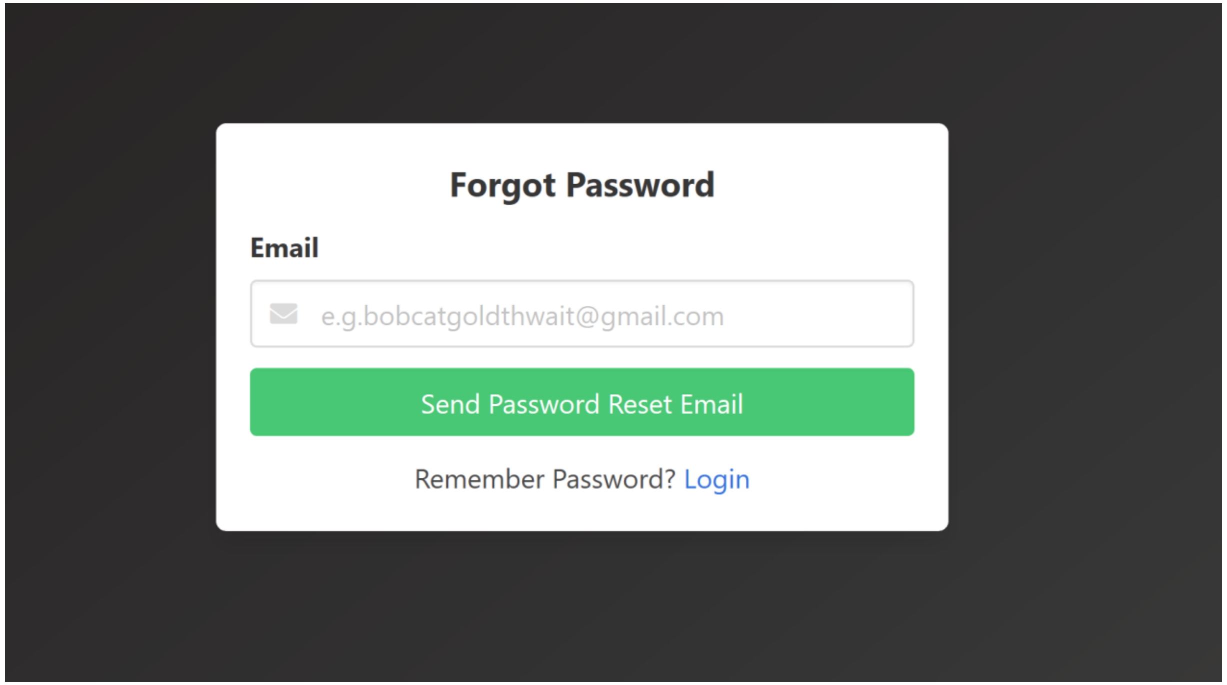
m@m.com

Here is a description of the item

# SCREENSHOTS



# SCREENSHOTS



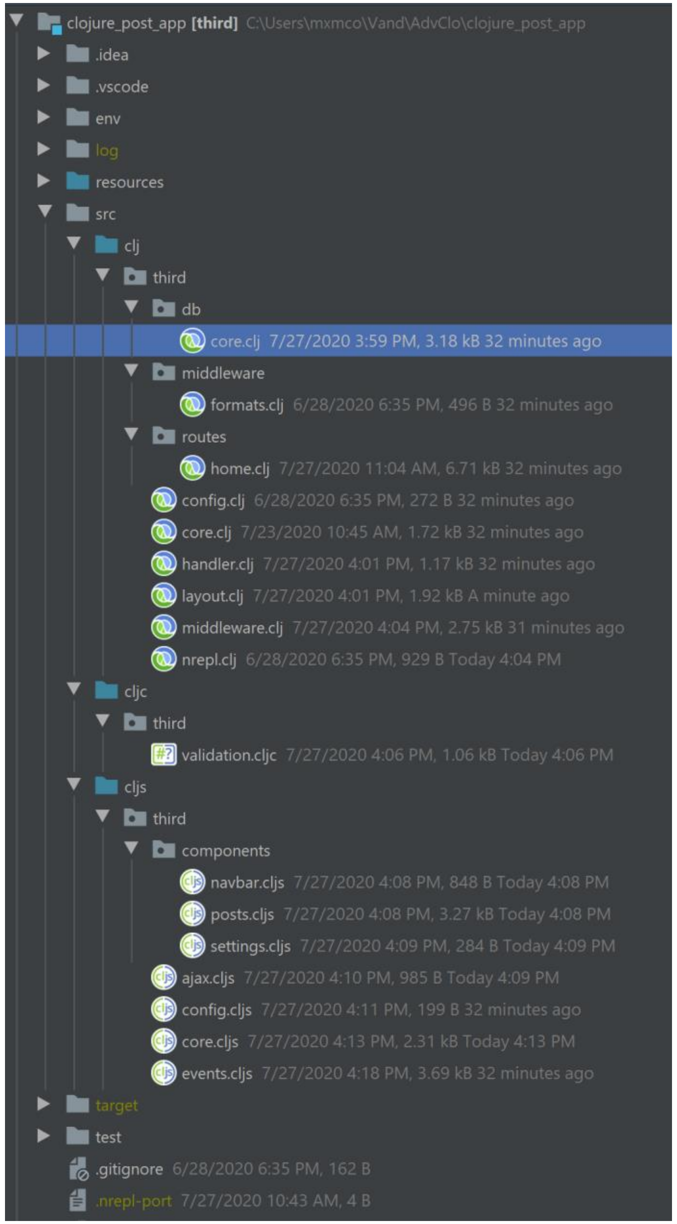
Forgot Password

Email

e.g.bobcatgoldthwait@gmail.com

Remember Password? Login

# SCREENSHOTS



```
clojure_post_app [third]  C:\Users\mxmco\Vand\AdvClo\clojure_post_app
  .idea
  .vscode
  env
  log
  resources
  src
    clj
      third
        db
          core.clj  7/27/2020 3:59 PM, 3.18 kB 32 minutes ago
        middleware
          formats.clj  6/28/2020 6:35 PM, 496 B 32 minutes ago
        routes
          home.clj  7/27/2020 11:04 AM, 6.71 kB 32 minutes ago
        config.clj  6/28/2020 6:35 PM, 272 B 32 minutes ago
        core.clj  7/23/2020 10:45 AM, 1.72 kB 32 minutes ago
        handler.clj  7/27/2020 4:01 PM, 1.17 kB 32 minutes ago
        layout.clj  7/27/2020 4:01 PM, 1.92 kB A minute ago
        middleware.clj  7/27/2020 4:04 PM, 2.75 kB 31 minutes ago
        nrepl.clj  6/28/2020 6:35 PM, 929 B Today 4:04 PM
    cljc
      third
        validation.cljc  7/27/2020 4:06 PM, 1.06 kB Today 4:06 PM
    cljs
      third
        components
          navbar.cljs  7/27/2020 4:08 PM, 848 B Today 4:08 PM
          posts.cljs  7/27/2020 4:08 PM, 3.27 kB Today 4:08 PM
          settings.cljs  7/27/2020 4:09 PM, 284 B Today 4:09 PM
        ajax.cljs  7/27/2020 4:10 PM, 985 B Today 4:09 PM
        config.cljs  7/27/2020 4:11 PM, 199 B 32 minutes ago
        core.cljs  7/27/2020 4:13 PM, 2.31 kB Today 4:13 PM
        events.cljs  7/27/2020 4:18 PM, 3.69 kB 32 minutes ago
  target
  test
  .gitignore  6/28/2020 6:35 PM, 162 B
  .nrepl-port  7/27/2020 10:43 AM, 4 B
```

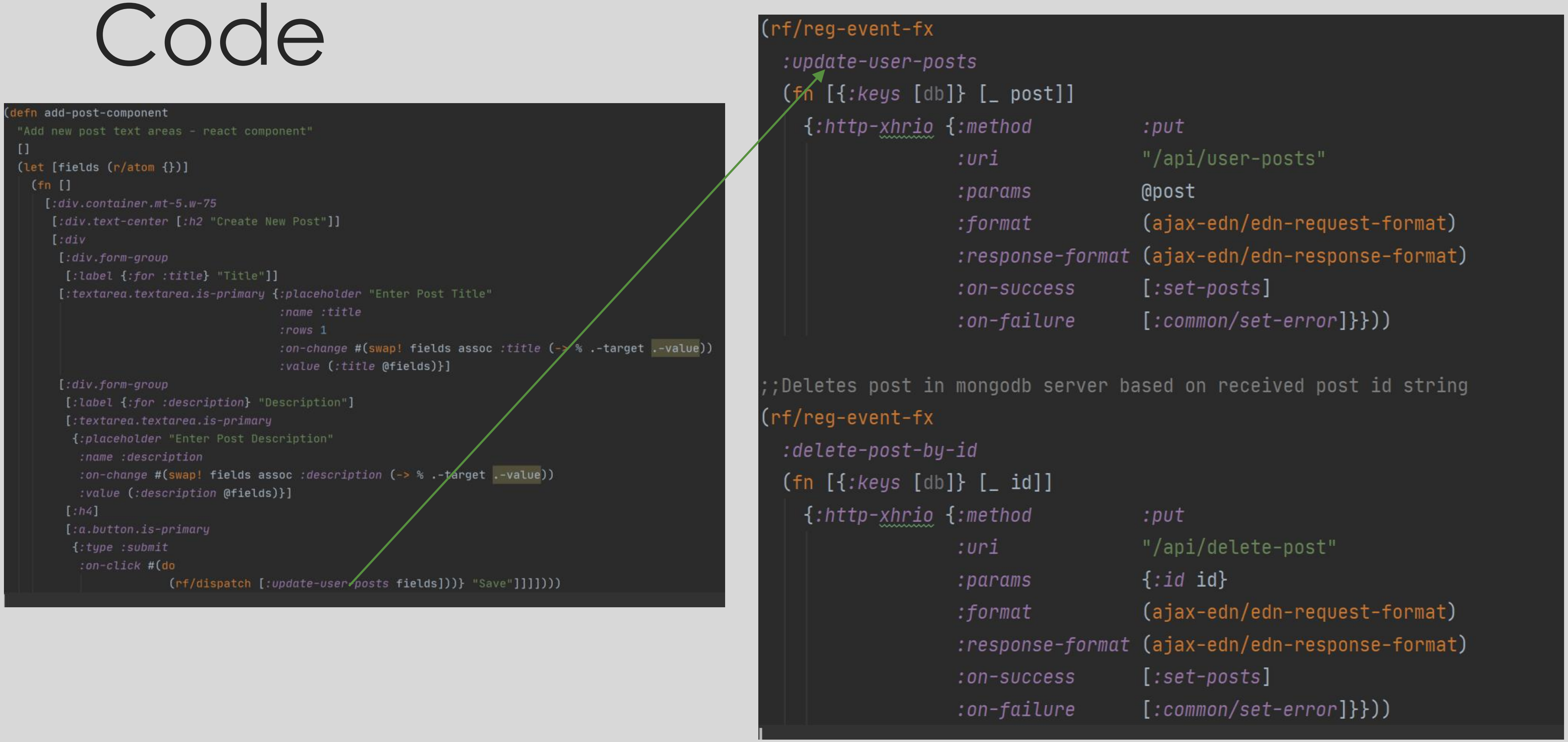
# Project Struct.

- CLJ
  - db
  - Middleware
  - Routes
  - Other

- CLJC
  - Validation

- CLJS
  - Components
  - Events/HTTP/Config

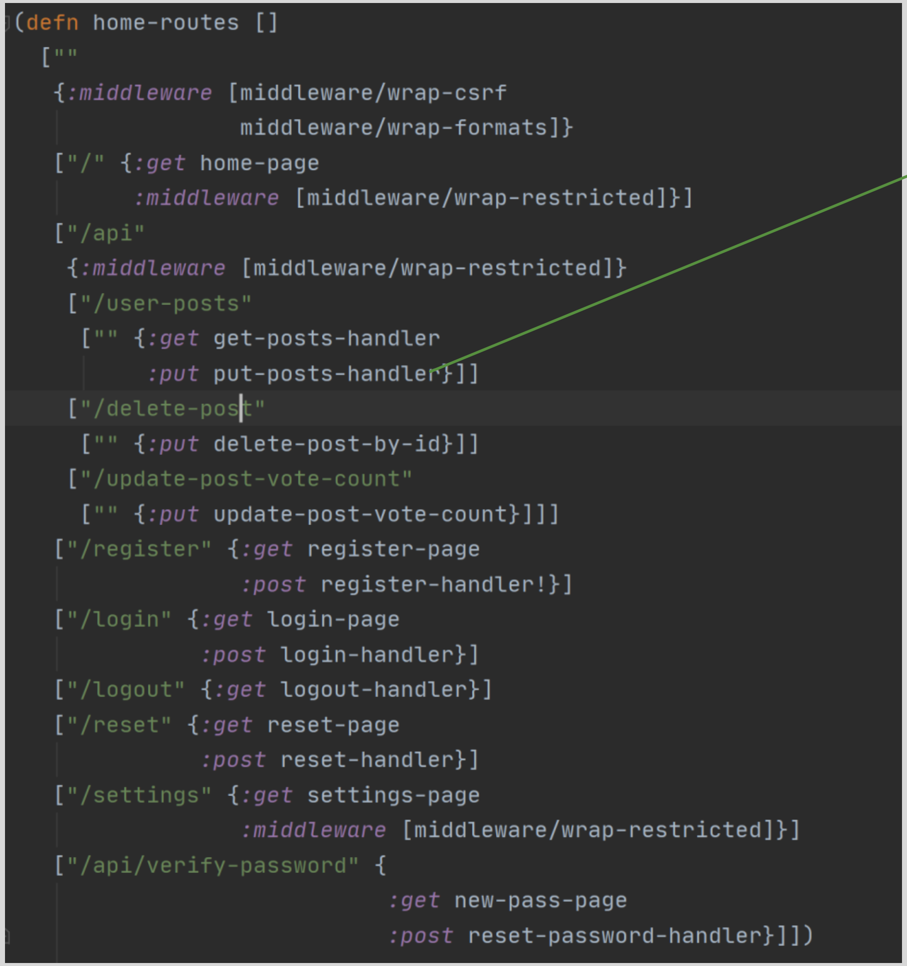
# Code



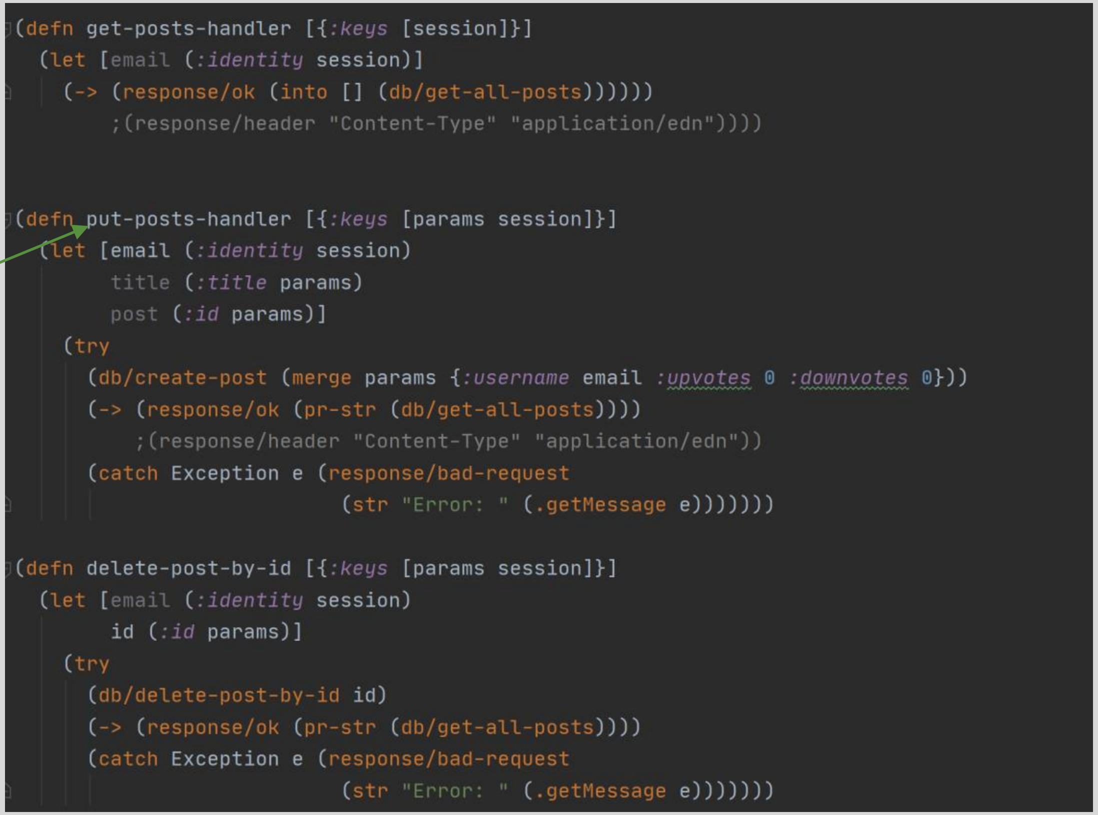
```clojure
(defn add-post-component
  "Add new post text areas - react component"
  []
  (let [fields (r/atom {})]
    (fn []
      [:div.container.mt-5.w-75
       [:div.text-center [:h2 "Create New Post"]]
       [:div
        [:div.form-group
         [:label {:for :title} "Title"]]
        [:textarea.textarea.is-primary {:placeholder "Enter Post Title"
                                        :name :title
                                        :rows 1
                                        :on-change #(swap! fields assoc :title (-> % .-target .-value))
                                        :value (:title @fields)}]
        [:div.form-group
         [:label {:for :description} "Description"]
         [:textarea.textarea.is-primary
          {:placeholder "Enter Post Description"
           :name :description
           :on-change #(swap! fields assoc :description (-> % .-target .-value))
           :value (:description @fields)}]
         [:h4]
         [:a.button.is-primary
          {:type :submit
           :on-click #(do
                        (rf/dispatch [:update-user-posts fields]))} "Save"]]]])))
```

```clojure
(rf/reg-event-fx
  :update-user-posts
  (fn [{:keys [db]} [_ post]]
    {:http-xhrio {:method          :put
                  :uri             "/api/user-posts"
                  :params          @post
                  :format          (ajax-edn/edn-request-format)
                  :response-format (ajax-edn/edn-response-format)
                  :on-success      [:set-posts]
                  :on-failure      [:common/set-error]}}))

;;Deletes post in mongodb server based on received post id string
(rf/reg-event-fx
  :delete-post-by-id
  (fn [{:keys [db]} [_ id]]
    {:http-xhrio {:method          :put
                  :uri             "/api/delete-post"
                  :params          {:id id}
                  :format          (ajax-edn/edn-request-format)
                  :response-format (ajax-edn/edn-response-format)
                  :on-success      [:set-posts]
                  :on-failure      [:common/set-error]}}))
```

# Code

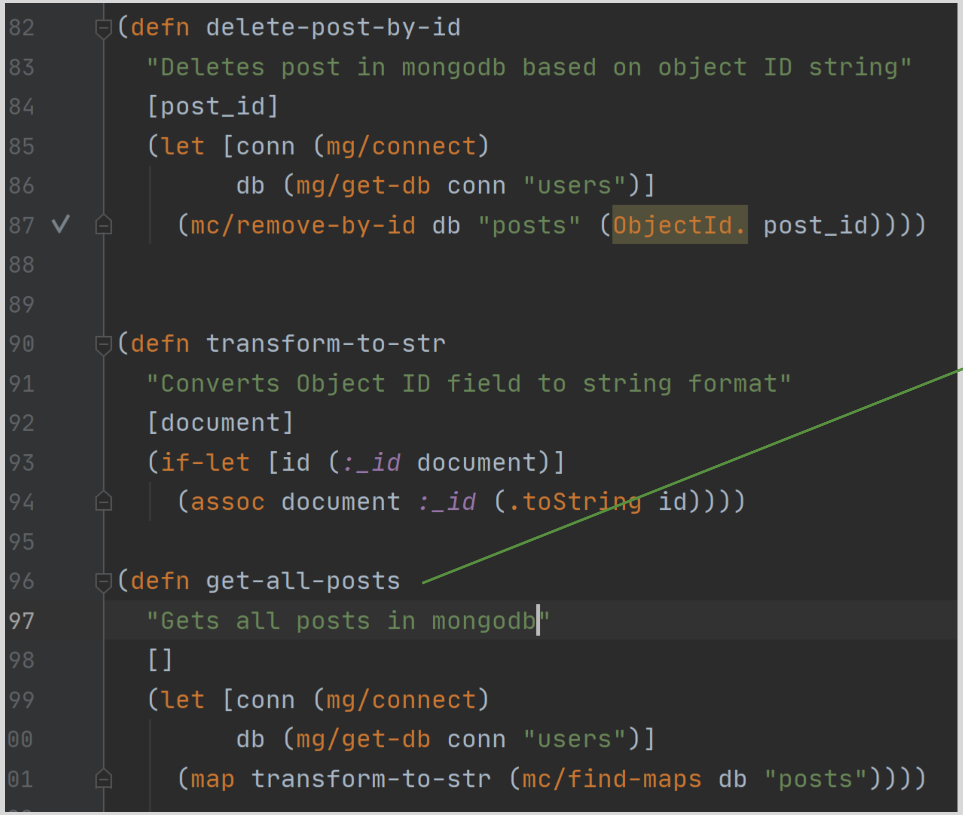


```
(defn home-routes []
  [""
   {:middleware [middleware/wrap-csrf
                 middleware/wrap-formats]}
   ["/" {:get home-page
         :middleware [middleware/wrap-restricted]}]
   ["/api"
    {:middleware [middleware/wrap-restricted]}
    ["/user-posts"
     ["" {:get get-posts-handler
          :put put-posts-handler}]]
    ["/delete-post"
     ["" {:put delete-post-by-id}]]
    ["/update-post-vote-count"
     ["" {:put update-post-vote-count}]]]
   ["/register" {:get register-page
                 :post register-handler!}]
   ["/login" {:get login-page
              :post login-handler}]
   ["/logout" {:get logout-handler}]
   ["/reset" {:get reset-page
              :post reset-handler}]
   ["/settings" {:get settings-page
                 :middleware [middleware/wrap-restricted]}]
   ["/api/verify-password" {
                            :get new-pass-page
                            :post reset-password-handler}]])
```

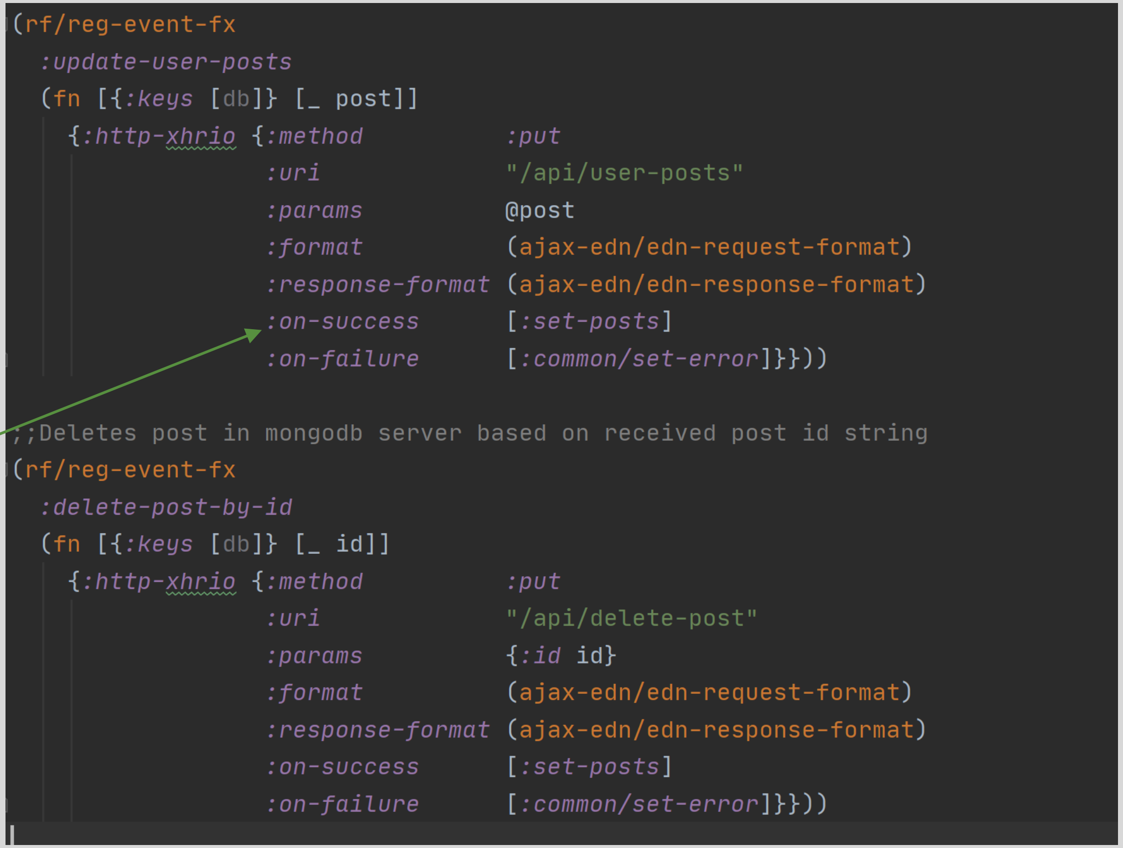


```
(defn get-posts-handler [{:keys [session]}]
  (let [email (:identity session)]
    (-> (response/ok (into [] (db/get-all-posts)))))
        ;(response/header "Content-Type" "application/edn"))))


(defn put-posts-handler [{:keys [params session]}]
  (let [email (:identity session)
        title (:title params)
        post (:id params)]
    (try
      (db/create-post (merge params {:username email :upvotes 0 :downvotes 0}))
      (-> (response/ok (pr-str (db/get-all-posts))))
          ;(response/header "Content-Type" "application/edn"))
      (catch Exception e (response/bad-request
                           (str "Error: " (.getMessage e)))))))

(defn delete-post-by-id [{:keys [params session]}]
  (let [email (:identity session)
        id (:id params)]
    (try
      (db/delete-post-by-id id)
      (-> (response/ok (pr-str (db/get-all-posts))))
      (catch Exception e (response/bad-request
                           (str "Error: " (.getMessage e)))))))
```

# Code

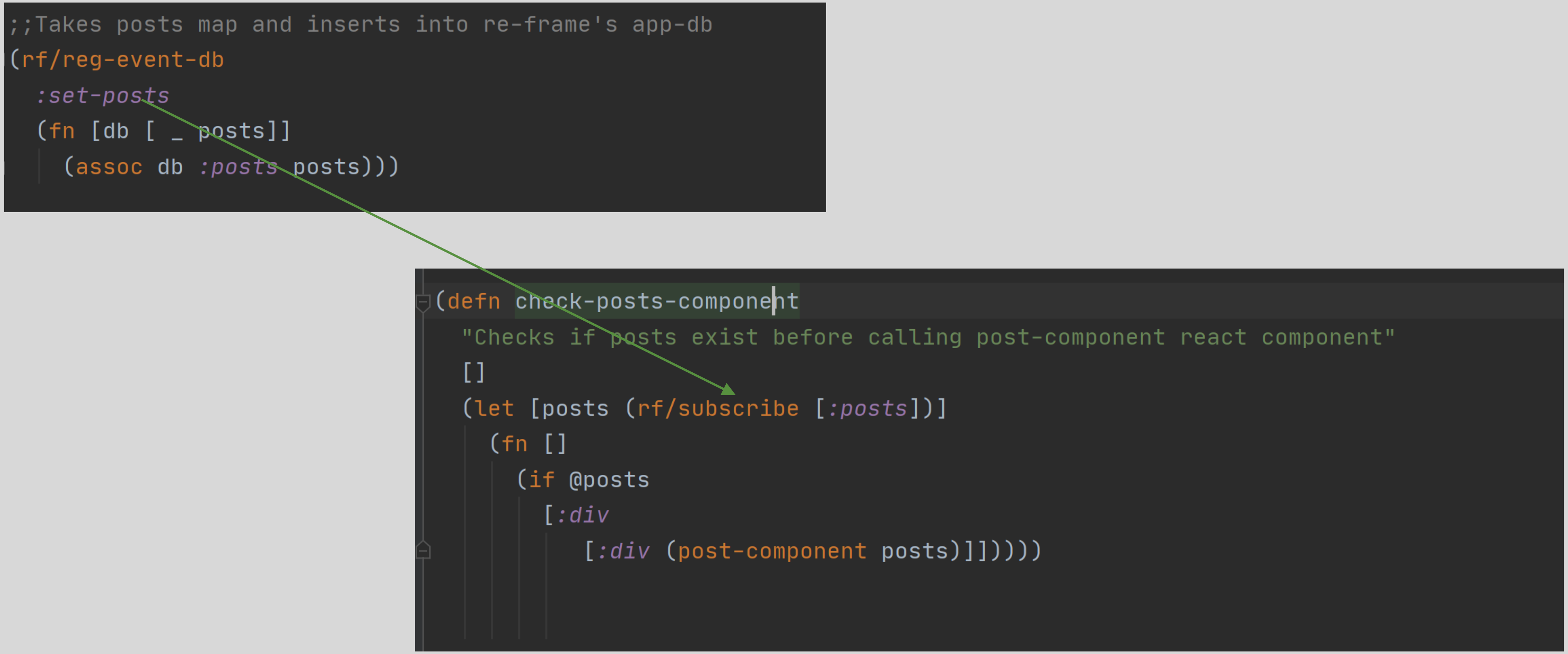


```clojure
(defn delete-post-by-id
  "Deletes post in mongodb based on object ID string"
  [post_id]
  (let [conn (mg/connect)
        db (mg/get-db conn "users")]
    (mc/remove-by-id db "posts" (ObjectId. post_id))))


(defn transform-to-str
  "Converts Object ID field to string format"
  [document]
  (if-let [id (:_id document)]
    (assoc document :_id (.toString id))))

(defn get-all-posts
  "Gets all posts in mongodb"
  []
  (let [conn (mg/connect)
        db (mg/get-db conn "users")]
    (map transform-to-str (mc/find-maps db "posts"))))
```



```clojure
(rf/reg-event-fx
  :update-user-posts
  (fn [{:keys [db]} [_ post]]
    {:http-xhrio {:method          :put
                  :uri             "/api/user-posts"
                  :params          @post
                  :format          (ajax-edn/edn-request-format)
                  :response-format (ajax-edn/edn-response-format)
                  :on-success      [:set-posts]
                  :on-failure      [:common/set-error]}}))

;;Deletes post in mongodb server based on received post id string
(rf/reg-event-fx
  :delete-post-by-id
  (fn [{:keys [db]} [_ id]]
    {:http-xhrio {:method          :put
                  :uri             "/api/delete-post"
                  :params          {:id id}
                  :format          (ajax-edn/edn-request-format)
                  :response-format (ajax-edn/edn-response-format)
                  :on-success      [:set-posts]
                  :on-failure      [:common/set-error]}}))
```

# Code



```clojure
;;Takes posts map and inserts into re-frame's app-db
(rf/reg-event-db
  :set-posts
  (fn [db [ _ posts]]
    (assoc db :posts posts)))
```

```clojure
(defn check-posts-component
  "Checks if posts exist before calling post-component react component"
  []
  (let [posts (rf/subscribe [:posts])]
    (fn []
      (if @posts
        [:div
          [:div (post-component posts)]]))))
```