U2PGHFU5U : over steps
U2PGHFU5U : One solution is to keep a separate database, but it is not very clean
U2PGHFU5U : Aaah you can send the context forward
U2PGHFU5U : Return `[result context]`
U2PGHFU5U : Next question: in `clj-gatling` my response times are way longer than they actually are. It seems like it is not only counting the time my request takes, but also the time it takes to run on my computer. Is there a way to fix this?
U2PGHFU5U : When I time the request
```
(defn login-request [ctx]
  (go
    (time (let [{:keys [status] :as res} @(http/get "<https://www.google.com>"
                                            {:headers {"Accept" "text/html"}})]
          [(= status 200) (assoc ctx :state "state")])))
```

It prints "Elapsed time: 200ms".

But the end result in the Gatling reports states it lasts longer than 1200ms.

U5YHX0TQV : why is it wrapped in a go block? Is clj-gatling designed around this
U5YHX0TQV : cause you're doing a blocking call with @?
U2PGHFU5U : Well spotted
U2PGHFU5U : If I don't wrap it in a go-block the same problem occurs
U2PGHFU5U : I think if I use ```(defn- http-get [url _]  (let [response (chan)
        check-status (fn [{:keys [status]}]
                       (go (>! response (= 200 status))))]
    (http/get (str base-url url) {} check-status)
response))``` like in the example things will work.

U2PGHFU5U : <https://github.com/mhjort/clj-gatling-example/blob/master/src/clj_gatling_example/simulations.clj>
U2PGHFU5U : Okay not using a go block works!
U2PGHFU5U : This works:```
(defn login-request [ctx]
  (let [check-result (fn [{:keys [status]}] (= status 200))]
    (http/get "<https://www.google.nl>"
              {:headers {"Accept" "text/html"}}
              check-result)))
```

No problem :slightly_smiling_face:

Up to 4000 users

U170T0Y3H : How can I refer to a var inside ns1 when the macro defined in ns1 is called from ns2?```
(ns ns1)

(defn a-fn* [] "hello")

(defmacro a-marco []
  `(defn a-fn [] (a-fn*)))

(ns ns2)

(ns1/a-marco) ;=&gt; Can't refer to qualified var that doesn't exist
```

U060FKQPN : that's going to work
U060FKQPN : I don't believe you see that error message on a fresh repl, you must have some stale state
U060FKQPN : also that macro is slightly wrong, should be ```(defmacro a-macro [] `(defn ~'a-fn [] (a-fn*)))```

U060FKQPN : needless to say macros like that are discouraged in clojure
U0JFCEH9P : I'm playing with an event sourcing/CQRS style system in Clojure. It involves some number of load-balanced app servers. My idea is to have local in-memory caches, and then "catch up" by applying pending events before any read operations.
U0JFCEH9P : I'm trying to avoid having any extra pieces like a message queue
U0JFCEH9P : does that seem sane?
U170T0Y3H : <@U060FKQPN> It worked, with the unquote-quote. But now I'm pretty discouraged.