U0C3SLTHP : i can't see
U051SS2EU : but I could see it for eg. something that repeatedly accesses a resource and eventually returns a result
U0C3SLTHP : yeah
U0C3SLTHP : clojure is not lazy by default, like haskell right, how the lazy works on clojure , `(lazy-seq (cons h (flat t)))`
U051SS2EU : right - but many functions are lazy
U5Z4ECHCM : Lazy is only there when you don't want it to be &gt;.&gt;
U051SS2EU : including concat
U5Z4ECHCM : &gt; trying to print something&gt; LAZY SEQ HELLO

U051SS2EU : that's only if you call str - just printing won't do that
U051SS2EU : ```+user=&gt; (str (map inc (range 10)))"clojure.lang.LazySeq@c5d38b66"
+user=&gt; (println (map inc (range 10)))
(1 2 3 4 5 6 7 8 9 10)
nil
```

U051SS2EU : and if you need to build up a string, `pr-str` will help ```user=&gt; (str "fixed: " (pr-str (map inc (range 10))))"fixed: (1 2 3 4 5 6 7 8 9 10)"
```

U5Z4ECHCM : Well how-about-that
U5Z4ECHCM : learn something every day
U5YHX0TQV : yada seem to have implemented something themselves <https://github.com/juxt/yada/blob/master/ext/oauth2/src/yada/oauth.clj>. Maybe its time we see a new repository appearing under your github account :wink:
U2PGHFU5U : Does anyone know how to store state in one instance of a simulation in `clj-gatling`?
/edit Answer from the docs:

<http://i.imgur.com/SISAxzT.png>

U2PGHFU5U : Hmm looks like the `context` is passed along in every step. Hopefully I can just assoc.
U2PGHFU5U : That's not it. `assoc`ing to that context is not persistent
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 (&gt;! 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
```