LISP

(COMMON) LISP

WHY?

WHY?

WHY NOT?

WHY?

PRODUCTIVITY

WHY?

PRODUCTIVITY
EXPRESSIVITY

WHY?

PRODUCTIVITY
EXPRESSIVITY
CREATIVITY

WHY?

PRODUCTIVITY
EXPRESSIVITY
CREATIVITY
(ALL THE -IVITIES, REALLY.)

PROBLEMS WITH LISP

PROBLEMS WITH LISP

ALL THOSE DAMN PARENTHESES

PROBLEMS WITH LISP

ALL THOSE DAMN PARENTHESES

DISTRIBUTION

PROBLEMS WITH LISP

ALL THOSE DAMN PARENTHESES

DISTRIBUTION

TOO POWERFUL MAYBE?

PROBLEMS WITH LISP

ALL THOSE DAMN PARENTHESES
DISTRIBUTION
TOO POWERFUL MAYBE?
DUNNO, DISTRIBUTION AGAIN?

NO MORE!

NO MORE!

WE HAVE BOSH

NO MORE!

WE HAVE BOSH
WE HAVE DOCKER

NO MORE!

WE HAVE BOSH
WE HAVE DOCKER
WE HAVE CLOUD FOUNDRY

SHOUT!

BETTER NOTIFICATIONS

WHY?

WHY?

BETTER NOTIFICATIONS

TIME FOR A CODE DIVE

OWN THE LIBS

```
(load "~/quicklisp/setup.lisp")
(ql:quickload :hunchentoot)
(ql:quickload :drakma)
(ql:quickload :cl-json)
```

YOU COULD BE A MODEL, BABY!

```
(defclass event ()
  ((message
    :initarg :message
    :accessor message)
   (ok
    :initarg :ok
    :accessor event-ok?)))

(defclass state ()
  ((name
    :initarg :name
    :accessor state-name)
   (status
    :initarg :status
    :initform "unknown"
    :accessor status)
   (last-event
    :initarg :last-event
    :accessor last-event)))

(defvar *states* ())
```

I'M OKAY

```
(defun still-ok? (e1 e2)
  (and (event-ok? e1)
       (event-ok? e2)))
```

YOU'RE OKAY

```
(defun transition (e1 e2)
  (cond ((still-ok? e1 e2) "working")
        ((event-ok? e2)    "fixed")
        (t                 "broken")))
```

```lisp
(defun update-state (state event)
  (let ((prev (last-event state)))
    (setf (status     state) (transition prev event)
          (last-event state) event)
    (when (not (still-ok? prev event))
      (notify-about state))
    state))

(defun create-state (key topic event)
  (let ((state (make-instance 'state
                              :name topic
                              :last-event event
                              :status (if (event-ok? event)
                                          "working" "broken"))))
    (setf *states* (acons key state *states*))
    state))

(defun ingest (topic event)
  (let* ((key (intern topic))
         (state (cdr (assoc key *states*))))
    (if state
      (update-state state event)
      (create-state key topic event))))
```

HELLO, JSON,
MY OLD FRIEND...

```lisp
(defun event-json (event)
  (when event
    `((message . ,(message   event))
      (ok      . ,(event-ok? event)))))

(defun state-json (state)
  (when state
    `((name   . ,(state-name state))
      (status . ,(status     state))
      (last   . ,(event-json (last-event state))))))
```

```lisp
(defun api (port)
  (defun handle-get-states ()
    (setf (content-type* *reply*) "application/json")
    (format nil (encode-json-to-string *states*)))
  (push (create-prefix-dispatcher "/states" 'handle-get-states)
        *dispatch-table*)
  (start (make-instance 'easy-acceptor :port port)))
```

WEB 2.1

```lisp
(defun json-body ()
  (decode-json-from-string
    (raw-post-data :force-text t)))
```

A HELPFUL INTERLUDE

```lisp
(defun attr (object field)
  (cdr (assoc field object)))
```

```
(defun api (port)
  (defun handle-get-states ()
    (setf (content-type* *reply*) "application/json")
    (format nil (encode-json-to-string *states*)))
  (push (create-prefix-dispatcher "/states" 'handle-get-states)
        *dispatch-table*)

  (defun handle-post-events ()
    (setf (content-type* *reply*) "application/json")
    (let ((b (json-body)))
      (format nil "~A~%"
        (encode-json-to-string
          (state-json
            (ingest
              (attr b :topic)
              (make-instance 'event :ok      (attr b :ok)
                                    :message (attr b :message))))))))))
(push (create-prefix-dispatcher "/events" 'handle-post-events)
      *dispatch-table*)

(start (make-instance 'easy-acceptor :port port)))
```

WEB 2.1.7

WAIT WAIT WAIT

```
(defun api (port)
  (defun handle-get-states ()
    (setf (content-type* *reply*) "application/json")
    (format nil (encode-json-to-string *states*)))
  (push (create-prefix-dispatcher "/states" 'handle-get-states)
        *dispatch-table*)

  (defun handle-post-events ()
    (setf (content-type* *reply*) "application/json")
    (let ((b (json-body)))
      (format nil "~A~%"
        (encode-json-to-string
          (state-json
            (ingest
              (attr b :topic)
              (make-instance 'event :ok      (attr b :ok)
                                    :message (attr b :message))))))))))
(push (create-prefix-dispatcher "/events" 'handle-post-events)
      *dispatch-table*)

(start (make-instance 'easy-acceptor :port port)))
```

UG-LY

WE (LISP) CAN DO BETTER

```lisp
(defun api (port)
  (defun handle-get-states ()
    (setf (content-type* *reply*) "application/json")
    (format nil (encode-json-to-string *states*)))
  (push (create-prefix-dispatcher "/states" 'handle-get-states)
        *dispatch-table*)

  (defun handle-post-events ()
    (setf (content-type* *reply*) "application/json")
    (let ((b (json-body)))
      (format nil "~A~%"
        (encode-json-to-string
          (state-json
            (ingest
              (attr b :topic)
              (make-instance 'event :ok      (attr b :ok)
                                    :message (attr b :message))))))))
  (push (create-prefix-dispatcher "/events" 'handle-post-events)
        *dispatch-table*)
```

WE NEED A

MACRO

```lisp
(defmacro handle (url &body body)
  (let ((fn (gensym "fn")))
    `(progn
       (defun ,fn ()
         (setf (content-type* *reply*) "application/json")
         (format nil "~A~%" (encode-json-to-string (progn ,@body))))
       (push (create-prefix-dispatcher ,url ',fn) *dispatch-table*))))
```

WEB 3.0

```lisp
(defun api (port)
  (handle "/states" *states*)
  (handle "/events"
    (let ((b (json-body)))
      (state-json
        (ingest
          (attr b :topic)
          (make-instance 'event :ok      (attr b :ok)
                                :message (attr b :message)))))))
  (start (make-instance 'easy-acceptor :port port)))
```

WHAT HAVE WE DONE?

122 LINES OF CODE

A COMPLETE STATE ENGINE

SLACK INTEGRATION

somebot 1.0 APP 3:17 AM

something else is now broken!

> this is my message https://starkandwayne.com

something else is now fixed!

> this is my message https://starkandwayne.com

CODE

https://github.com/jhunt/shout
https://github.com/jhunt/shout-boshrelease
https://github.com/jhunt/shout-resource