Rocking with Racket

Marc Burns
Beatlight Inc

What am I doing here?

My first encounter with Racket was in 2010

I wanted to use Racket in industry

The opportunity arose in June 2014: **Loft**

What am I doing here?

My first encounter with Racket was in 2010

I wanted to use Racket in industry

The opportunity arose in June 2014: **Loft**

Eerie Chime

Marc
Upload Home
Log Out

Stems | Samples | Kits

bright swee

ADD TAG

bright sweep | bright | bright spark | too bright | wide bright | bright n brittle | bright but thick

INSTRUMENTS SHOW

Select an instrument category

COMPATIBILITY OFF ON

✔ Melodic ✔ Rhythmic

PRICE (USD)

Min. 0 - none Max.

dreamlike fantasy — Sea Foam

haunting dreamy eerie lonesome — Ice Dream

by Mark — lonely pensive thoughtful outer space — X Feels

by Mark — intriguing mysterious dreamy

by Mark
CHAT FOLLOW PROFILE

Eerie Chime

LOWER +2 HIGHER

SLOWER 99 FASTER

FREE

FILE EDIT CREATE UPLOAD COMMENT

- 99BPM +

Loop Off

Markers

Big Brass Section

Smooth Distant Felix Leblanc 1 a

DI Figz Session 1

1 3 5 7 9 11 13 15 17 19 21

DROP

Cool Tune 6 (Marc Remix) (Loaded in Studio)

Context

We connect artists and producers around the world

We're a 4-person team (2 engineers)

Technical challenges:

- Searching ~1e6 *stems* (short clips of notated music)
- Music editing and digital FX in the browser
- Search tools (query-by-hum, compatibility)
- Handling money
- Concurrent interactive editing

Why Racket?

We evaluated a few languages and frameworks before writing any code

- *JavaScript* with *node.js* is too full of explicit CPS and weird/unexpected automatic conversions + behaviour of builtins
- *C++* is too low-level; even simple functionality requires a lot of work to achieve (*Boost* compounds the problem)
- *Python* is great, but libraries for *PostgreSQL* interaction aren't well-maintained

Why Racket?

- *Racket* comes with an excellent *PostgreSQL* library and a fairly mature typed variant
- Macros are also very attractive (a double-edged sword, as we'll see…)
- Fringe benefit: It might make the job of finding new engineers a bit easier
- Most importantly, simple functionality is easy to implement and the resulting code is clear

June 2014

```
(define (loft.user id)
   (match-define
     (vector name email)
     (query-value SELECT name |,| email
                  FROM loft.user WHERE id = ,id))
  (lambda (action . args)
    (case action
      [(get-name) name]
      [(set-name)
       (set! name (first args))
       (save!)]
      …)))
```

- This started to turn into a full-blown ORM
- The approach was too general
- Ad-hoc object system
- No introspection in code using the data model

```
> (define alice (loft.user 34))
> alice
#<procedure>
> (alice 'set-shoutout mp3-bytes)
```

[an exotic exception is thrown by an unrelated module]

- Bad use of macros: Made the defining code clear but produced code that was opaque and full of opportunities for bugs!

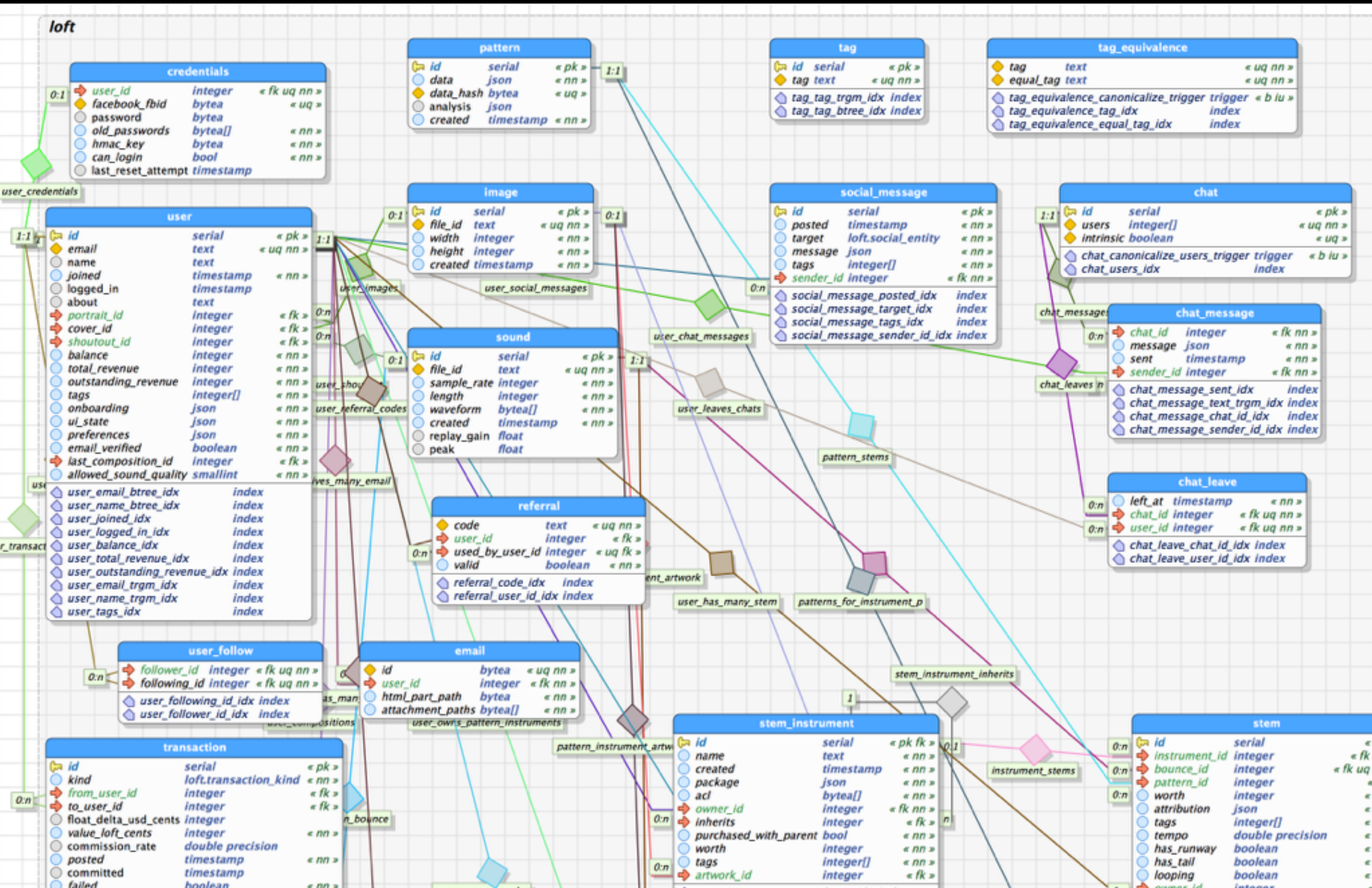
What about the rest?

- We used the Racket web framework with in-memory continuations (the server had state for auth and pagination)
- Requests could also be performed over WebSocket
- Had nginx sitting in front of Racket for SSL termination and static content
- Many problems…

- Messy: Parsing HTTP twice (nginx, Racket), separate code for HTTP and WebSocket sessions
- State makes scaling up difficult
- Many concurrent connections could overwhelm db connection pool
- Lots of bugs resulting from values passed in the wrong place or wrong types
- Adding contracts only made the bugs more clear
- nginx and Racket don’t play well like this: too many files, TCP connect overhead x6

Maybe we should move to Typed Racket
and overhaul our architecture.

April 2015


loft
credentials
user_id integer « fk uq nn »
facebook_fbid bytea « uq »
password bytea
old_passwords bytea[] « nn »
hmac_key bytea « nn »
can_login bool « nn »
last_reset_attempt timestamp
user_credentials
user
id serial « pk »
email text « uq nn »
name text
joined timestamp « nn »
logged_in timestamp
about text
portrait_id integer « fk »
cover_id integer « fk »
shoutout_id integer « fk »
balance integer « nn »
total_revenue integer « nn »
outstanding_revenue integer « nn »
tags integer[] « nn »
onboarding json « nn »
ui_state json « nn »
preferences json « nn »
email_verified boolean « nn »
last_composition_id integer « fk »
allowed_sound_quality smallint « nn »
user_email_btree_idx index
user_name_btree_idx index
user_joined_idx index
user_logged_in_idx index
user_balance_idx index
user_total_revenue_idx index
user_outstanding_revenue_idx index
user_email_trgm_idx index
user_name_trgm_idx index
user_tags_idx index
pattern
id serial « pk »
data json « nn »
data_hash bytea « uq »
analysis json
created timestamp « nn »
tag
id serial « pk »
tag text « uq nn »
tag_tag_trgm_idx index
tag_tag_btree_idx index
tag_equivalence
tag text « uq nn »
equal_tag text « uq nn »
tag_equivalence_canonicalize_trigger trigger « b iu »
tag_equivalence_tag_idx index
tag_equivalence_equal_tag_idx index
image
id serial « pk »
file_id text « uq nn »
width integer « nn »
height integer « nn »
created timestamp « nn »
user_images
user_social_messages
social_message
id serial « pk »
posted timestamp « nn »
target loft.social_entity « nn »
message json « nn »
tags integer[] « nn »
sender_id integer « fk nn »
social_message_posted_idx index
social_message_target_idx index
social_message_tags_idx index
social_message_sender_id_idx index
chat
id serial « pk »
users integer[] « uq nn »
intrinsic boolean « uq »
chat_canonicalize_users_trigger trigger « b iu »
chat_users_idx index
chat_message
chat_id integer « fk nn »
message json « nn »
sent timestamp « nn »
sender_id integer « fk nn »
chat_message_sent_idx index
chat_message_text_trgm_idx index
chat_message_chat_id_idx index
chat_message_sender_id_idx index
sound
id serial « pk »
file_id text « uq nn »
sample_rate integer « nn »
length integer « nn »
waveform bytea[] « nn »
created timestamp « nn »
replay_gain float
peak float
user_chat_messages
user_leaves_chats
pattern_stems
chat_leave
left_at timestamp « nn »
chat_id integer « fk uq nn »
user_id integer « fk uq nn »
chat_leave_chat_id_idx index
chat_leave_user_id_idx index
referral
code text « uq nn »
user_id integer « fk »
used_by_user_id integer « uq fk »
valid boolean « nn »
referral_code_idx index
referral_user_id_idx index
user_has_many_stem
patterns_for_instrument_p
user_follow
follower_id integer « fk uq nn »
following_id integer « fk uq nn »
user_following_id_idx index
user_follower_id_idx index
email
id bytea « uq nn »
user_id integer « fk nn »
html_part_path bytea « nn »
attachment_paths bytea[] « nn »
user_owns_pattern_instruments
stem_instrument_inherits
transaction
id serial « pk »
kind loft.transaction_kind « nn »
from_user_id integer « fk »
to_user_id integer « fk »
float_delta_usd_cents integer
value_loft_cents integer « nn »
commission_rate double precision
posted timestamp « nn »
committed timestamp
failed boolean
stem_instrument
id serial « pk fk »
name text « nn »
created timestamp « nn »
package json « nn »
acl bytea[] « nn »
owner_id integer « fk nn »
inherits integer « fk »
purchased_with_parent bool « nn »
worth integer « nn »
tags integer[] « nn »
artwork_id integer « fk »
instrument_stems
stem
id serial « pk »
instrument_id integer « fk »
bounce_id integer « fk uq »
pattern_id integer
worth integer
attribution json
tags integer[]
tempo double precision
has_runway boolean
has_tail boolean
looping boolean
user_referral_codes
pattern_instrument_artwork

April 2015

```
(define-type Social<%>
  (Class
   [get-tags (-> (Sequenceof String))]
   [add-tag (String . -> . Void)]
   [check-access (Actor Access . -> . Boolean)]
   …))
```

```
(define-type User%
  (Class
   #:implements Social<%>
   [get-id (-> Natural)]
   [get-email (-> String)]
   [get-name (-> String)]
   …))
```

```
(: social-mixin
 (All (r #:row)
   ((Class #:row-var r
      #:implements Social-Obligations<%>)
    . -> .
    (Class #:row-var r
      #:implements Social<%>))))

(define (social-mixin %)
  (class %
    (super-new)
    (define/public (get-tags)
      …)))
```

```
(: user% User%)
(define user%
  (social-mixin
    (class object%
      (super-new)

      (define-from-row
        (query-row
        SELECT ,@user-row-sql
        FROM loft.user WHERE id = ,id)
       [name : String]
       [email : String]
       [joined : Timestamp]
       …)

     (define/public (get-name) name)
     …))))
```

```
> (define user (get-user-by-id 42))
> (define-values (data-port file-name)
    (send
      (send user get-last-composition)
      export))
> (process/ports #f data-port #f "play -")
```

Very nice to work with!

Why does it take 3 minutes to `raco make`?

Why is it so sluggish to run?
(about twice as slow on common queries)

```
#lang typed/racket

(provide (all-defined-out))

(define-type C%
  (Class [id : (-> (Listof Byte) (Listof Byte))]))

(: c% C%)
(define c%
  (class object%
    (super-new)
    (define (id xs) xs)))

(define (test [c : C%])
  (time (void (send c id big-byte-list))))
```

```
(module* test/typed typed/racket
  (require (submod ".."))
  (test (new c%)))

(module* test/untyped racket
  (require (submod ".."))
  (test (new c%)))
```

```
> (require (submod "." test/typed))
cpu time: 0 real time: 0 gc time: 0

> (require (submod "." test/untyped))
cpu time: 3831 real time: 3829 gc time: 3101
```

Why?

- Classes and objects that pass the typed/untyped boundary are wrapped in contracts

- This is necessary for soundness

- Contracts in a complex system of objects are large and slow (typed methods that accept objects will be augmented to wrap the arguments in contracts; this is recursive)

- Solution (for now): No untyped code!

```
(define-type Media (U (Instance Image%)
                      (Instance Sound%)))
(define (media->response [media : Media])
  : Response
  (response
    200 "Good" (current-seconds)
    (send media get-mime-type)
    (if (is-a? media sound%)
        (list
          (header #"X-Content-Duration"
                  (send media get-duration)))
        empty)
    (send media get-port))
```

```
send: method not understood by object
```

Reality:

```
> is-a?
- : (-> Any ClassTop Boolean)
```

Desire:

```
> make-is-a?
- : (All (A)
     (A . -> .
      (Any . -> . Boolean
       : #:+ (Instance A))))
```

Just using is-a? isn’t sound

Solution (bad): Cast to (Instance Class)!

```
(if (is-a? media sound%)
    (list
       (header …))
    empty)
```

```
(with-handlers
 ([exn:fail? (lambda _ empty)])
 (list
   (header #"X-Content-Duration"
           (send (cast media (Instance Sound%))
                 get-duration))))
```

- This wraps media in a contract
- It’s also asking the wrong question:

  “Does media act like an instance of type Sound%?”
- What I really want to know:

  “Is media an instance of class sound%?”

Solution (less bad): Implement make-is-a?

```
(require/typed/unsafe
 "is-a-maker.rkt"
 [make-is-a?
   (All (A)
     (A . -> .
       (Any . -> . Boolean
          : #:+ (Instance A))))])
```

- Works; no contracts!
- It’s unsound
- Needs typed-racket PR#126

What about the rest?

- No more server state
- WebSocket and HTTP are handled by some nasty C++ code (nginx/SCGI for HTTP)
- API requests are delivered to Racket in a high-level form via redis
- One Racket-level thread per Racket process to handle requests
- Average request opens no new TCP connections

Did it matter?

BEFORE:

- 2 unique exceptions / week in production
- 50 requests / process second

AFTER:

- 0.3 unique exceptions / week in production
- 200 requests / process second

Future Directions

- Some Racket code doing musical analysis in the browser with Whalesong (in the works)
- Make debugging memory errors in racket3m easier?
- Memory allocation / GC traffic visualization
- TR bindings for db that fix the sql-null problem (stretch goal: integrate types with queries?)

Large-scale projects in Racket are fun and good!

We've recently changed our name to **Outro** for trademark reasons. You can find us out outro.io .