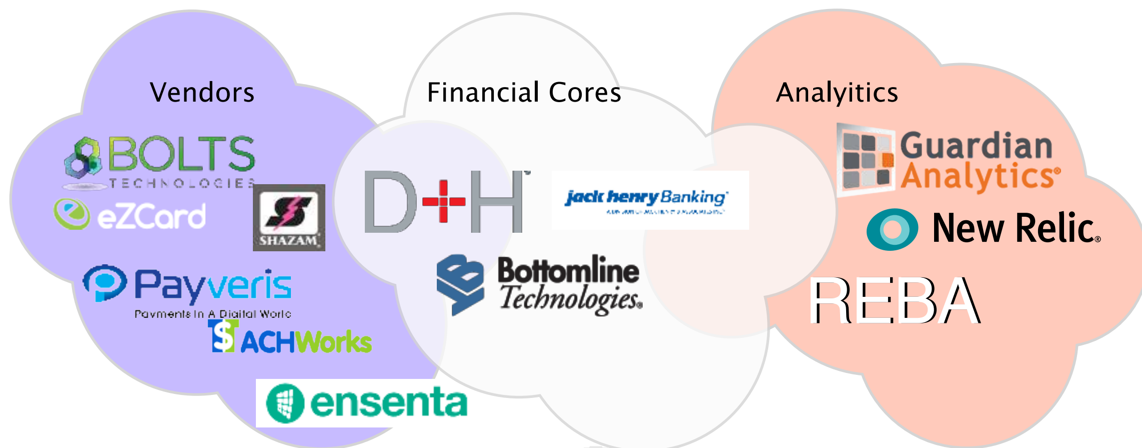
michael graf

239-822-8752
me@mgr.af

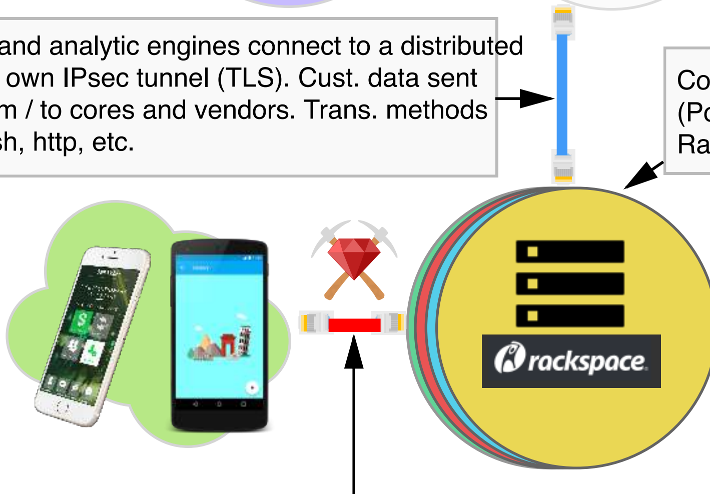
Backend Core Services

Featuring: Rails 3, Ruby 2.4, PostgreSQL, Redis



Vendors, cores, and analytic engines connect to a distributed network via their own IPsec tunnel (TLS). Cust. data sent and received from / to cores and vendors. Trans. methods include telnet, ssh, http, etc.

Core / vendor data hits Rackspace, which holds DB (Postgres), Memcache (Redis), Content (Images, UX & UI). Rails creates a session.



The meat is here. Goal being to create a unified experience regarless of core, vendor, or customer. The bulk of my work is here; this is where my Rails' core classes and methods create and parse data from and to cores, save customer data, transactions, balances, validation, method audits (compliance), etc.

Backend (core, vendor, and analytics) are all written in Ruby. Here is an example of a vendor (Check Image) Interface I wrote.

```
def initialize(parameters)

    @checkimage = Hash.new()

    begin

      organization = parameters['organization']
      raise CommonInterfaceError, "Required parameter, organization, not provided" if !organization

      urlformethods = parameters['urlformethods'].to_s.strip
      raise CommonInterfaceError, 'Required parameter, urlformethods, not provided' if urlformethods.blank?

(1)   businessrules                = Commonfunctions.buildBusinessRules(parameters)
      useSequence                  = businessrules['USESEQUENCENUMBER'].to_s.upcase == 'TRUE'  ? true  : false
      checkNumberZero              = businessrules['CHECKNUMBERZERO'].to_s.upcase == 'TRUE'    ? true  : false
      useDocType                   = businessrules['USEDOCTYPE'].to_s.upcase == 'FALSE'        ? false : true
      useDatePosted                = businessrules['USEDATEPOSTED'].to_s.upcase == 'FALSE'     ? false : true
      replaceAccountNumberWithMICR = businessrules['REPLACEACCOUNTNUMBERWITHMICR'] == 'TRUE'   ? true  : false

(2)   account_number, amount, check_number, date_posted, sequenceNumber, mobileUser = parameters['check_image'].split('|')

      MalauzaiLogger(organization.external_id, 'NotACustomerName', 'parameters', parameters.inspect) if organization.is_debug_on

      #Using Malauzai::Account instead of translatekey to get full account number without dashes or any other value
(3)   account_number  = Malauzai::Account.where(organization_id: organization.id, crossreference_id: account_number).first
      account_number  = replaceAccountNumberWithMICR ? account_number.micr : account_number.account_number_decrypted
      check_number    = 0 if check_number.blank?

      urlParams = Hash.new
      urlParams['docType']      = (useSequence && check_number == 0) ? '37' : '29' #29 => check, 37 => deposits -- PM-4973
      urlParams['acct']         = account_number
      urlParams['amount']       = amount
      urlParams['seq']          = sequenceNumber if useSequence
      urlParams['cknum']        = check_number
      urlParams['date_posted']  = date_posted
(4)   urlParams['pdate']        = date_posted

      urlParams.delete('cknum')   if useSequence || (check_number == 0 && !checkNumberZero)
      urlParams.delete('docType') if !useDocType
      urlParams.delete('date_posted') if !useDatePosted

(5)   myhttpclient = HTTPClient.new()
      myhttpclient.debug_dev = Logger.new(Commonfunctions.obtain_org_log_path(organization)) if organization.is_debug_on

      ft = Thread.new do
        local_urlParams = urlParams.clone
        local_urlParams['F_B'] = local_urlParams['fb'] = 'F'
(6)     response = myhttpclient.get(urlformethods, local_urlParams)
        MalauzaiLogger(organization.external_id, "NotACustomerName", 'Front Image', response.inspect) if organization.is_debug_on
        raise CommonInterfaceError, 'Unable to load front of check image' unless response.ok? && response.body.image?
        Base64.encode64(response.body)
      end

      bt = Thread.new do
        local_urlParams = urlParams.clone
        local_urlParams['F_B'] = local_urlParams['fb'] = 'B'
        response = myhttpclient.get(urlformethods, local_urlParams)

     . . . . .
```

(1) Business Rules are built in a common method, business rules are Redis key / values pairs. Benefit being: I can change these values without a production push / code changes via a CRUD web interface.

(2) check image parameters (listed) are received in a parameter which is set by a transactions interface

(3) Users' account numbers are retrieved via a cross reference table. Account numbers can also be replaced with other info depending on core / preference of the customer.

(4) In this case, the vendor uses HTTP to retrieve check images. Default params are built, some parms are added or removed depended on the customer.

(5) The HTTPClient class (gem) is instantiated and a logger is set to log to a customer's log directory and file.

(6) Threads are used to retrieve images in parallel and then send to the device. Error handling is in place (which will return the errors to the user if there are problems).

Full Stack Personal Projects

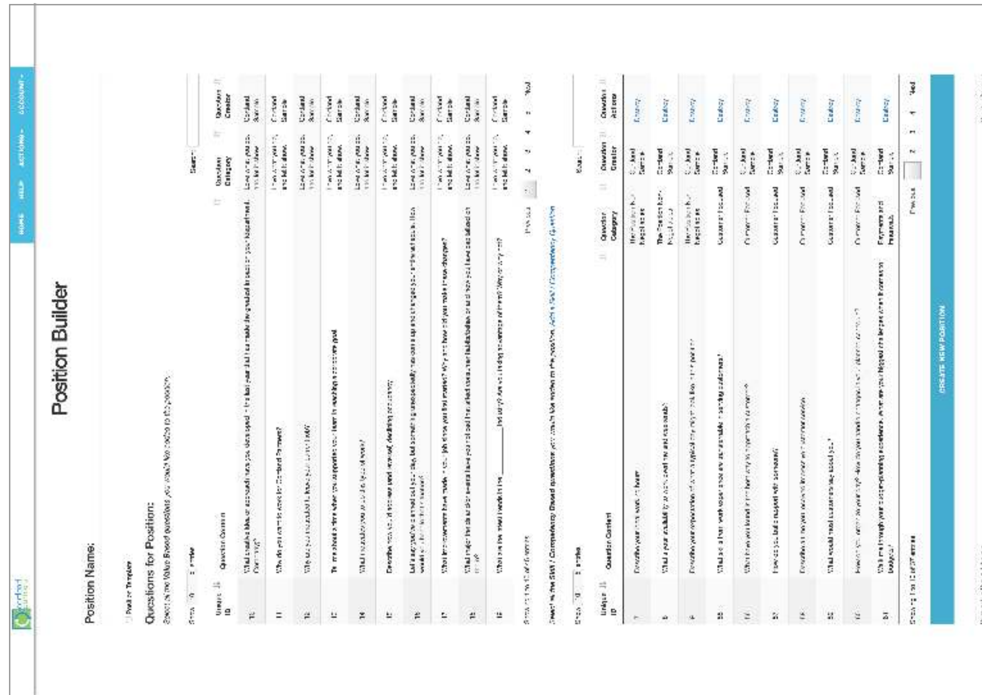
Featuring: Rails 5, Ruby 2.4, PostgreSQL, go, Sinatra, JS, SASS, gulp

Interviewing guide WebApp:

This project grew from a simple static site to a Rails project with polymorphic associations and mailers very quickly. The customer (now a good friend of mine!) wanted a way to streamline the creation of interview guides for his company.

Employees are able to login to the web app, and build interview guides based on predefined questions (mandated) questions, and their own custom questions. Questions, users, positions, question categories, etc. are all database driven objects with attributes and associations.

The interview guide above was generated with dynamic content, SASS, and database driven flows. Let me know if you want to see a complete interview application (source included), or like a visual demo. I can demo this in person.




```
module PositionsHelper

  def print_layout

    allQuestions = Array.new

(1) valueBasedQuestions = Array.new
    @position.questions.joins(:category).where(:categories => {:value_based => [true]}).where(:categories => {:is_global_use => [false]}).each do |question|
      h = Hash.new
      h[:category] = question.category.content
      h[:question] = question.content
      valueBasedQuestions << h
    end

    userQuestions = Array.new
    @position.questions.joins(:category).where(:categories => {:value_based => [false]}).where(:categories => {:is_global_use => [false]}).each do |question|
      h = Hash.new
      h[:category] = question.category.content
      h[:question] = question.content
      userQuestions << h
    end

    globalUseQuestions = Array.new
    @position.questions.joins(:category).where(:categories => {:is_global_use => [true]}).each do |question|
      h = Hash.new
      h[:category] = question.category.content
      h[:question] = question.content
      globalUseQuestions << h
    end

(2) allQuestions << valueBasedQuestions.group_by{|h| h[:category]}.each{|_, v| v.replace(v.map{|h| h[:question]})}
    allQuestions << userQuestions.group_by{|h| h[:category]}.each{|_, v| v.replace(v.map{|h| h[:question]})}
    allQuestions << globalUseQuestions.group_by{|h| h[:category]}.each{|_, v| v.replace(v.map{|h| h[:question]})}

  end
. . . .
```

(1) This is a helper method that I use to build the finished interview guide, between (1) and (2) I'm gathering question types from Active Record, putting them in order and pushing them to a questions array. I was the only dev with this one.

(2) Each question array is grouped based on their category, and then each question is added to an array which is in a array of hashes, where each hash key is the category.

Recreational ~~Projects~~ Hacks

Featuring: Ruby 2.4, PostgreSQL, Node, Siri, Asterisk, spammers.

```ruby
def orig
  stat = '0600'
  gen = (5001..8001).to_a.sample
  return "#{stat}#{gen}"
end

def gen_key(n)
  orig = n.dup
  gs = n.gsub!(/0+/,'')
  num = gs.to_s.chars.map(&:to_i).reduce(:+)
  f = (num * 854) % 1000
  if f <= 1000
    f = "0#{f}"
  end
  return "#{orig}#{f}"
end

code = gen_key(orig)
url = "realURLHere”
p url

exit
```

Background: I just moved to Austin and started my Rails job the very week I wrote this script. I was staying with my uncle at an apartment complex down-town and didn't have the money for $33.00 daily parking. I found out that residents are issued 24 hr parking passes and decided to take some time to correctly replicate them by reverse engineering the 'check' digits in the barcode. It worked, I parked for free and didn't tell a soul. I moved into my apartment a few days later.

After purchasing a new domain I started to receive 20+ robo-calls a month to my cell phone. It had to end. I used an open source telephony project (Asterisk) to build a simple mitigation technique to end the calls.

The system:

incoming call

Deny call. Call Forwarding is enabled; forwarded to Asterisk inbound trunk.

Virtual extension answers

IVR (Virtual Assistant) asks challenge question. ie: “Please dial 9 in order to complete this call”

if success: route call back to phone, change Caller ID to Asterisk server (noting validation).

if challenge fails or times out, call is disconnected. Or, a more sinister approach can be taken and the spammer can be placed on hold with an ever increasing volume.

More hacks available upon request.