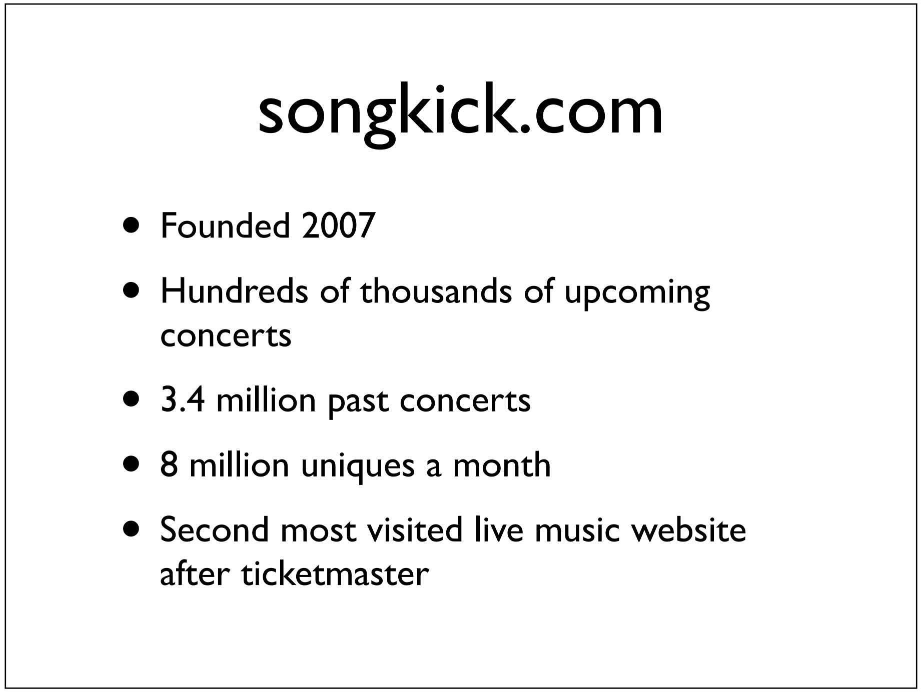
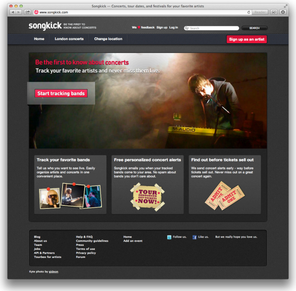
# How we scaled Songkick



# songkick.com

- Founded 2007
- Hundreds of thousands of upcoming concerts
- 3.4 million past concerts
- 8 million uniques a month
- Second most visited live music website after ticketmaster

Songkick aggregates live music performances, so music fans will never miss a gig again.
The user tells us which artists they like and which cities they would be interested in seeing those artist in.
When the artist are in those cities we contact the user. In the case of the website we send an email.



Be the first to know about concerts

Track your favorite artists and never miss them live.

**Track your favorite bands**

Tell us who you want to see live. Easily organize artists and concerts in one convenient place.

**Free personalized concert alerts**

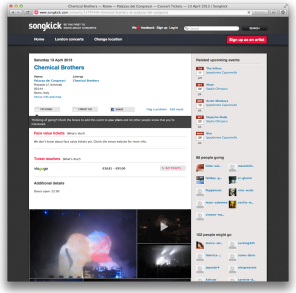
Songkick emails you when your tracked bands come to your area. No spam about bands you don't care about.

**Find out before tickets sell out**

We send concert alerts early - way before tickets sell out. Never miss out on a great concert again.

But we really hope you love us.

Kyte photo by gideon


Chemical Brothers — Rome — Palazzo dei Congressi — Concert Tickets — 13 April 2013 | Songkick
www.songkick.com/concerts/14797444-chemical-brothers-at-palazzo-dei-congressi
songkick BE THE FIRST TO KNOW ABOUT CONCERTS
We ♥ feedback Sign up Log in
Search
SEARCH
Home
London concerts
Change location
Sign up as an artist
□ I'M GOING
□ I MIGHT GO
SHARE
Flag a problem
Edit event
BUY TICKETS

Saturday 13 April 2013

**Chemical Brothers**

Venue
Palazzo dei Congressi
Piazzale J.F. Kennedy
00144
Rome, Italy
Venue info and map

Line-up
Chemical Brothers

Thinking of going? Check the boxes to add this event to your plans and let other people know that you're interested.

Face value tickets (What's this?)

We don't know about face value tickets yet. Check the venue website for more info.

Ticket resellers (What's this?)

viagogo €38.81 - €95.00

Additional details

Doors open: 22:00

Related upcoming events

TUE 11 JUN The Killers Ippodromo Capannelle

SAT 06 JUL Muse Stadio Olimpico

WED 10 JUL Arctic Monkeys Ippodromo Capannelle

SAT 20 JUL Depeche Mode Stadio Olimpico

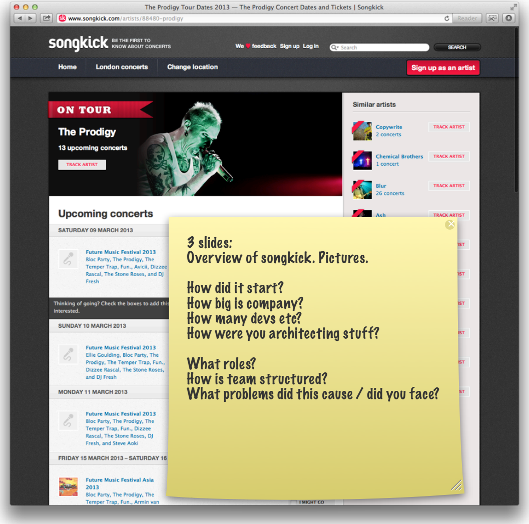
MON 29 JUL Blur Ippodromo Capannelle

88 people going

linda-sal... massimili... fattboy-g... el-glacial Peppemust nico-waits mary-salemme cecilia-m... simone-ma...

102 people might go

marco-sal... sucking345 federica-... stone-dario Japanair4 alexgrosseto Ashraw carmine-d...


The Prodigy Tour Dates 2013 — The Prodigy Concert Dates and Tickets | Songkick
www.songkick.com/artists/88480-prodigy
Reader
songkick BE THE FIRST TO KNOW ABOUT CONCERTS
We ♥ feedback Sign up Log in
Search
SEARCH
Home
London concerts
Change location
Sign up as an artist
ON TOUR
TRACK ARTIST
I MIGHT GO

**The Prodigy**

13 upcoming concerts

Similar artists

Copywrite 2 concerts TRACK ARTIST

Chemical Brothers 1 concert TRACK ARTIST

Blur 26 concerts TRACK ARTIST

Ash TRACK ARTIST

Upcoming concerts

SATURDAY 09 MARCH 2013

Future Music Festival 2013
Bloc Party, The Prodigy, The Temper Trap, Fun., Avicii, Dizzee Rascal, The Stone Roses, and DJ Fresh

Thinking of going? Check the boxes to add this interested.

SUNDAY 10 MARCH 2013

Future Music Festival 2013
Ellie Goulding, Bloc Party, The Prodigy, The Temper Trap, Fun., Dizzee Rascal, The Stone Roses, and DJ Fresh

MONDAY 11 MARCH 2013

Future Music Festival 2013
Bloc Party, The Prodigy, The Temper Trap, Fun., Dizzee Rascal, The Stone Roses, DJ Fresh, and Steve Aoki

FRIDAY 15 MARCH 2013 – SATURDAY 16

Future Music Festival Asia 2013
Bloc Party, The Prodigy, The Temper Trap, Fun., Armin van

3 slides:
Overview of songkick. Pictures.

How did it start?
How big is company?
How many devs etc?
How were you architecting stuff?

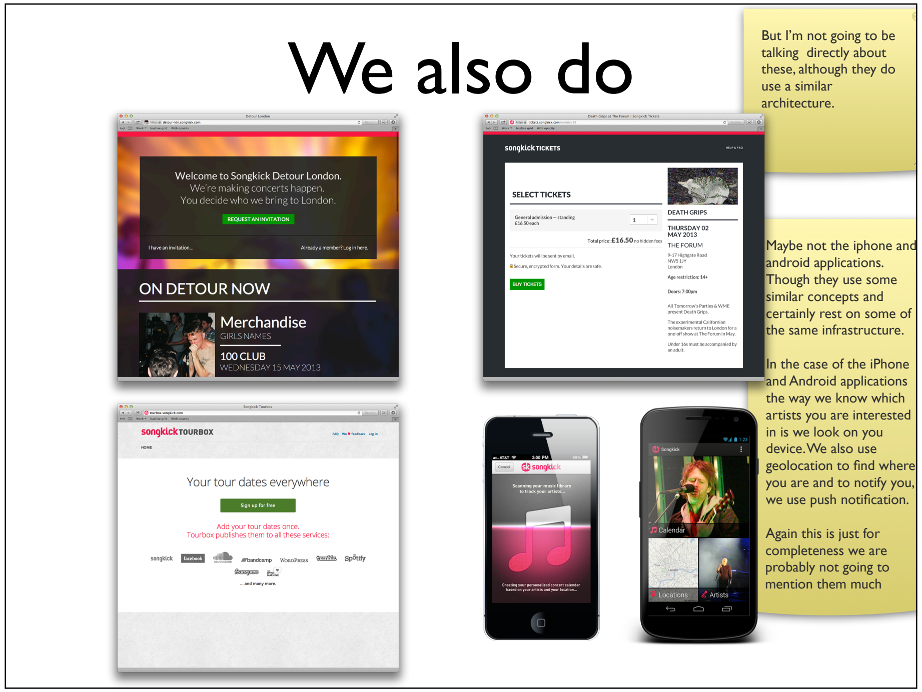
What roles?
How is team structured?
What problems did this cause / did you face?

# We Started small

- Four people in a flat in Spitalfields
- And grew

# We are still small

- 30 People in an office in Hoxton
- We are divided into cross functional teams the number and size of which change as we need



# We also do

But I'm not going to be talking  directly about these, although they do use a similar architecture.

Maybe not the iphone and android applications. Though they use some similar concepts and certainly rest on some of the same infrastructure.

In the case of the iPhone and Android applications the way we know which artists you are interested in is we look on you device. We also use geolocation to find where you are and to notify you, we use push notification.

Again this is just for completeness we are probably not going to mention them much

Welcome to Songkick Detour London.
We're making concerts happen.
You decide who we bring to London.

REQUEST AN INVITATION

ON DETOUR NOW

Merchandise
GIRLS NAMES
100 CLUB
WEDNESDAY 15 MAY 2013

songkick TICKETS

SELECT TICKETS

DEATH GRIPS
THURSDAY 02 MAY 2013
THE FORUM

BUY TICKETS

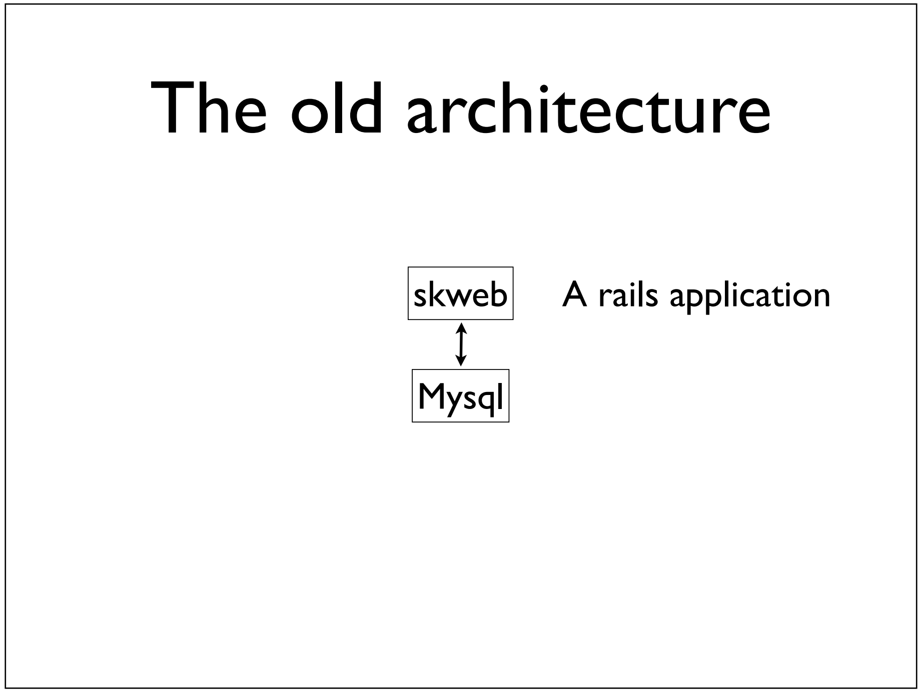
songkick TOURBOX

Your tour dates everywhere

Sign up for free

Add your tour dates once.
Tourbox publishes them to all these services:

But I'm not going to be talking  directly about these, although they do use a similar architecture.
Maybe not the iphone and android applications. Though they use some similar concepts and certainly rest on some of the same infrastructure.
In the case of the iPhone and Android applications the way we know which artists you are interested in is we look on you device. We also use geolocation to find where you are and to notify you, we use push notification.
Again this is just for completeness we are probably not going to mention them much meor



# The old architecture

skweb

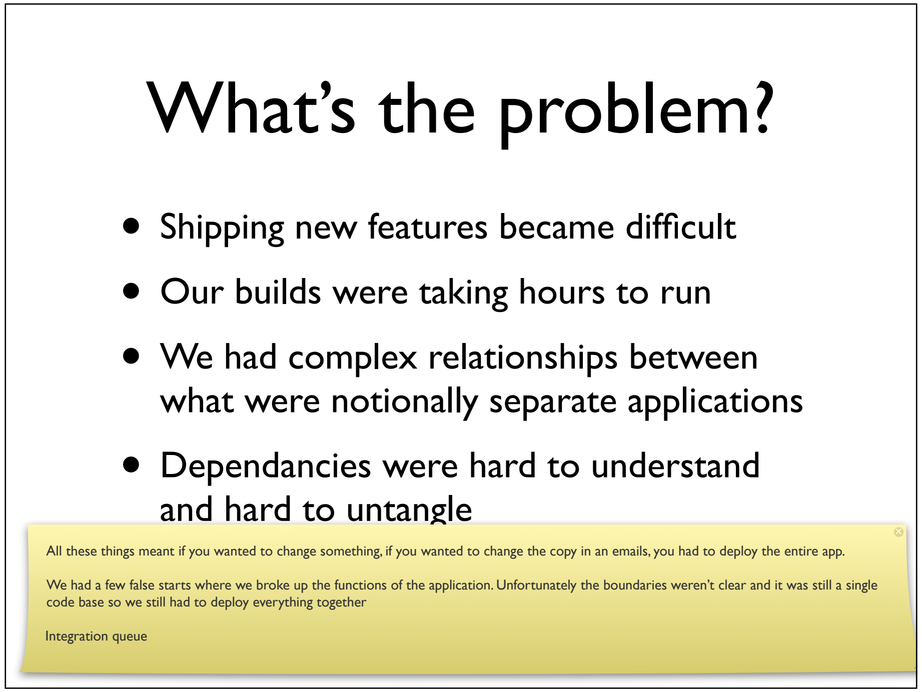
Mysql

A rails application

```
app/
   controllers/
   models/
   views/
features/
lib/
vendor/
   api/
   batch_processing/
   daemons/
   file_store/
   notifications/
   web_scrapers/
```

# What was the problem

- Initially features were over-engineered
- To develop and ship quickly it was easier to stick it all in one place
- But site was up, traffic growing. Trouble brewing …



# What’s the problem?

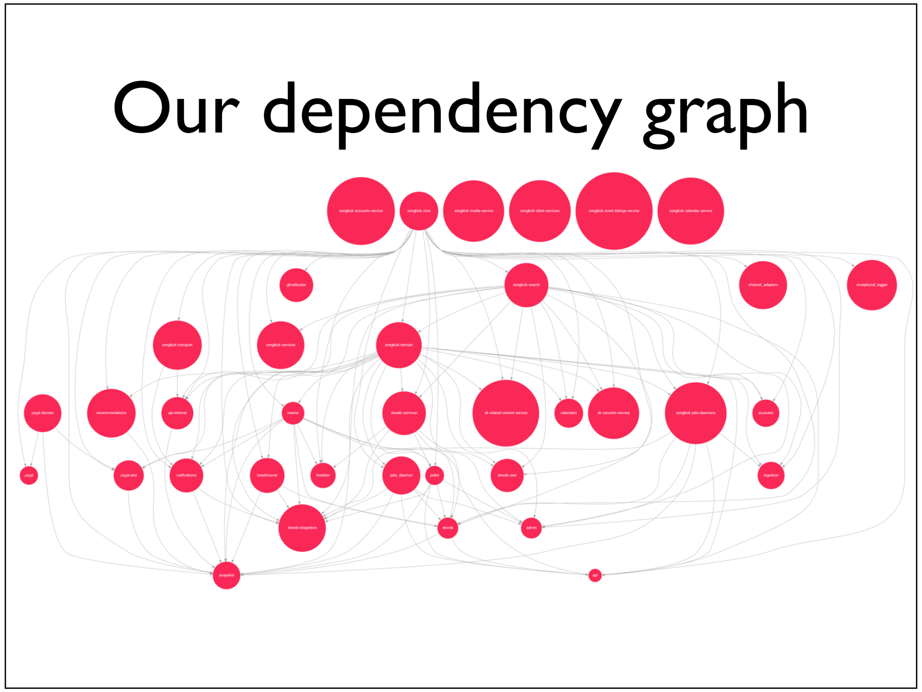
- Shipping new features became difficult
- Our builds were taking hours to run
- We had complex relationships between what were notionally separate applications
- Dependancies were hard to understand and hard to untangle

All these things meant if you wanted to change something, if you wanted to change the copy in an emails, you had to deploy the entire app.

We had a few false starts where we broke up the functions of the application. Unfortunately the boundaries weren’t clear and it was still a single code base so we still had to deploy everything together

Integration queue

All these things meant if you wanted to change something, if you wanted to change the copy in an emails, you had to deploy the entire app.
We had a few false starts where we broke up the functions of the application. Unfortunately the boundaries weren’t clear and it was still a single code base so we still had to deploy everything together
Integration queue
if I changed email and you change scrapers we had to wait, it was serial. This was a productivity sink hole



# Our dependency graph

Um, as you can see this is quite complex, and this isn't the original, because I couldn't find it, so we already had some services.
I've not brought the after graph, because it might take away form the horror.

# Why re-architect?

- Scale (more users doing more things)
- Developer productivity (more features, fewer bugs)
- Agility (more frequent releases, shorter time between releases)

For us increasing productivity was one of our most important goals.

We can respond to competitors and changes in the market more readily.

Yes performance was import, and, yes we had a lot of code in the app to make it more performant (caching etc) Which did make it reasonably performant.

# Why not re-architect?

These were all real concerns.

- You might never finish
- You might not achieve the benefits
- It might be easier to rewrite
- The new architecture might not be better than the old one

How do you persuade the other people in the company that spending six months doing this is worth doing?

This is a start-up you really can't spend six month navel gazing. The company could go bust.

You need to have a compelling reason and a plan.

# Collaboration!

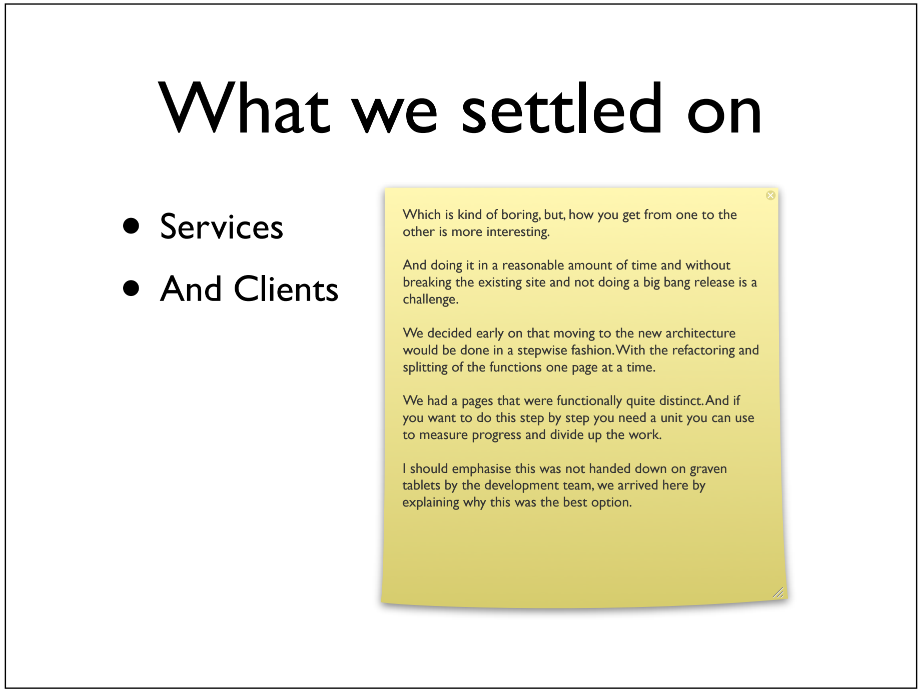
- Software is built by a team
- Not just a team of programmers
- You need to agree on what can be cut
- What is the minimum feature set needed
- What things are called

How did we know what cut and what to keep.
iPhone.

Why what things are called?
Shared vocabulary, every one in the company calls the same thing by the same name.

This will become important later on, in the development of the application.

And actually this process of identifying, cutting or adding features, choosing names and prioritising work is iterative. Each step of the way this process is repeated.



# What we settled on

- Services
- And Clients

Which is kind of boring, but, how you get from one to the other is more interesting.

And doing it in a reasonable amount of time and without breaking the existing site and not doing a big bang release is a challenge.

We decided early on that moving to the new architecture would be done in a stepwise fashion. With the refactoring and splitting of the functions one page at a time.

We had a pages that were functionally quite distinct. And if you want to do this step by step you need a unit you can use to measure progress and divide up the work.

I should emphasise this was not handed down on graven tablets by the development team, we arrived here by explaining why this was the best option.

Why? How did you make the decision?
How did you sell it?

# So far, so conventional

# What a service looks like

- Sinatra application
- Emits JSON over HTTP
- Accepts form encoded or JSON data over HTTP
- Completely internally encapsulated

We actually started with dummy services where we implemented the interface to the service inside the application.

Active record leaks up the

Worth noting our services don't have versioning, access control or XML. And that we do not need to maintain backwards compatibility, since we control all the clients. (at least for now)

And they are kind of REST. Not real, phd level REST, but certainly popular, Rails-developer style REST.

# What a client application looks like

- Rails application (so far any way)
- Has a traditional 'MVC' structure
- Gets all its data from services
- We added 'pages', 'components' and 'elements'

Pages are ruby classes that model what a pages behaviour should be

# How it fits together

- A Page
- Is made of components
- Some components are composed of elements

What is a page?

Why add pages?

What are components?

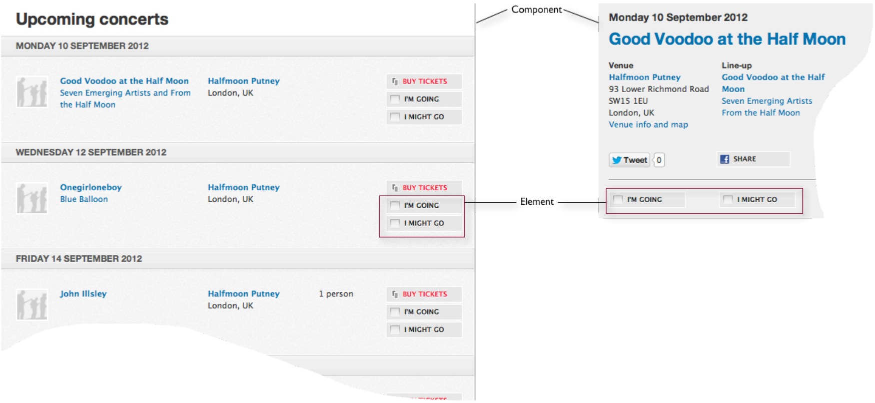
What are elements?

Benefits?

What makes a component?
A self contained unit on the page normally you can draw a box around it and give it a name.

Arbitrarily components cannot be nested.

What makes an element? Are common functionality shared between components.



## Upcoming concerts

MONDAY 10 SEPTEMBER 2012

Good Voodoo at the Half Moon
Seven Emerging Artists and From the Half Moon
Halfmoon Putney
London, UK

WEDNESDAY 12 SEPTEMBER 2012

Onegirloneboy
Blue Balloon
Halfmoon Putney
London, UK

FRIDAY 14 SEPTEMBER 2012

John Illsley
Halfmoon Putney
London, UK
1 person

Monday 10 September 2012

Good Voodoo at the Half Moon

Venue
Halfmoon Putney
93 Lower Richmond Road
SW15 1EU
London, UK
Venue info and map

Line-up
Good Voodoo at the Half Moon
Seven Emerging Artists
From the Half Moon

Event pages need a venue and an artist and an event

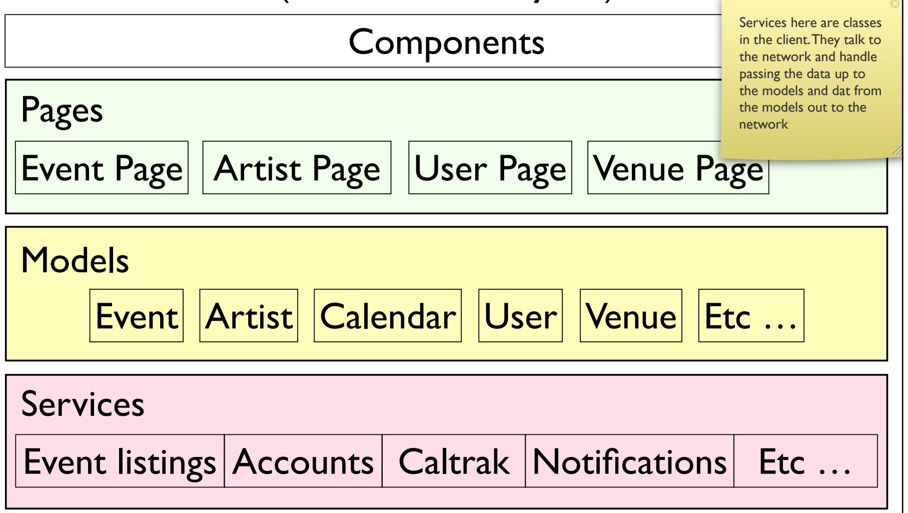
# Layers

(it’s all about layers)

Artist pages have an artist + calendar and media (see etc)

Users are User + calendar

Services here are classes in the client. They talk to the network and handle passing the data up to the models and dat from the models out to the network



Components

Pages

Event Page | Artist Page | User Page | Venue Page

Models

Event | Artist | Calendar | User | Venue | Etc …

Services

Event listings | Accounts | Caltrak | Notifications | Etc …

# So how does all this work in practice?

- The client is still a rails app with the familiar rails layout
- Anywhere a rails app might talk to a data store, the app talks to a service instead
- And we have added some conventions

The css file imports smaller files with shared styling

# The conventions

- Every Page has a type
- Every page has one CSS file
- The CSS file has the same name as the page type
- Every component has a corresponding CSS file
- If it needs it the component also has a javascript file

Components are self contained. Each component has a name and takes an object.

The object contains the data the component needs and any decision making is provided by methods on that object.

The name of the component is also the name of the template file on disc, the html class name and the name of its corresponding css and javascript files.

This tight convention around names makes understanding the dependancy between a

Every page having a css file does mean you get some repetition, but, the confidence it gives you about where changes will appear makes it well worth it.

# A little bit of code

```
skweb/
    app/
        controllers/
            venues_controller.rb
        models/
            page_models/
                venue.rb
            skweb/
                models/
                    venue.rb
        views/
            venues/
                _brief.html.erb
                show.html.erb
    public/
        javascripts/
            songkick/
                component/
                    tickets.js
        stylesheets/
            components/
                venue-brief.css
            shared/
                components/
                    brief.css
            venue.css
```

```
class VenuesController < ApplicationController
  def show
    @page = PageModels::Venue.new(venue,
logged_in_user)
  end
end
```

```
module PageModels
  class Venue < PageModels::Base
    def brief
      Brief.new(@venue, upcoming_events.total_entries,
@logged_in_user)
    end
 end
end
```

```
module PageModels
  class Venue
    class Brief
      def geolocation
        @venue.geolocation
      end
    end
  end
end
```

This is of cause in ruby but that hardly matters.

# Moving to the view

component() and shared_component() are defined in ApplicationHelper and look like this:

```
<div class="primary col">
  <%= component('brief', @page.brief) %>
  <%= component('map', @page.brief.geolocation) %>
  <%= shared_component('calendar_summary',   @page.calendar_s
  <%= shared_component('media_summary',      @page.media_summ
  <%= shared_component('media_links',        @page.media_link
  <%= shared_component('gigography_summary', @page.gigography
</div>
```

```
def component(component_name, object)
  return '' if object.nil?
  render :partial => component_name, :object => object
end

def shared_component(component_name, object)
  component("shared/components/#{component_name}", object)
end
```

```
@import 'shared/components/brief.css';
@import 'components/venue-brief.css';
@import 'components/venue-map.css';
@import 'shared/components/media-summary.css';
@import 'shared/components/event-listings.css';
```

# What did this give us

- Developer productivity was radically improved
- Application performance was much better

I'd hoped to have a graph showing improved page response times, but unfortunately we didn't keep them

Many of our services can be scaled horizontally mean at lest in the medium term we can increase capacity by adding nodes

The compartmentalisation of the application.

The independence of the services means parallelising development is simpler.

Knowing where to add functionality is easier.

# A leaner code base

- Before After
- 3.5MB 1.4MB ./app
- 1.8MB 744KB ./features
- 1.2MB 724KB ./spec

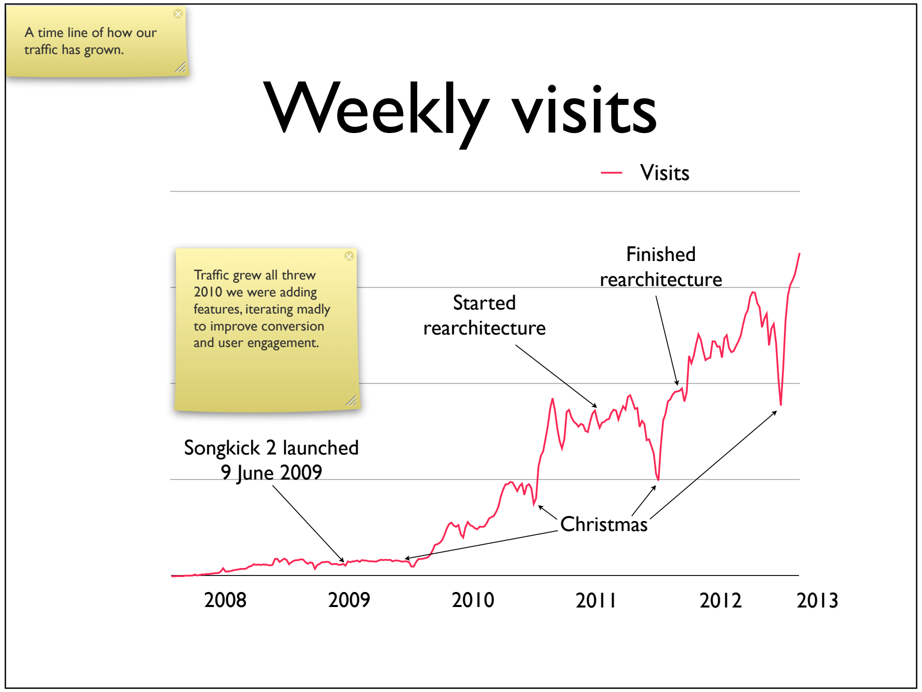
# Faster Builds

**Before**

- Over an hour
- Parallelized with 1 local and 10 ec2 instances

**After**

- 10 minutes
- 1 local machine



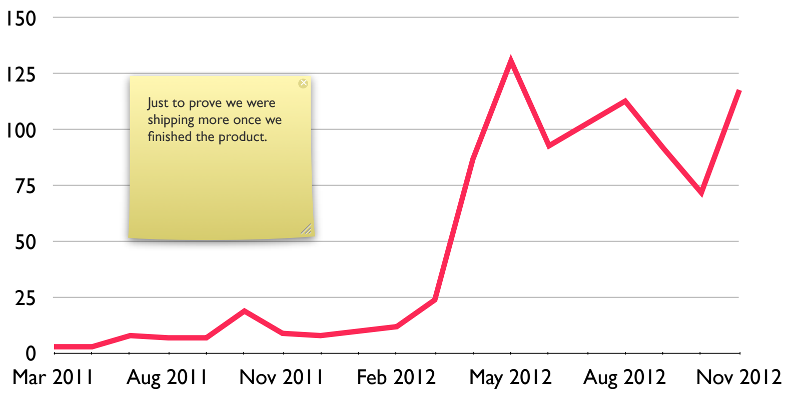
A time line of how our traffic has grown.

# Weekly visits

Traffic grew all threw 2010 we were adding features, iterating madly to improve conversion and user engagement.

In January 2011 we decided out current application was creaking under its own weight
We weren't releasing features fast enough, we had a lot of features users weren't using and we couldn't make changes with enough confidence.
We needed a plan!
This is mostly about what the plan was, how we implemented it and what the result was.

# Releases per month



Just to prove we were shipping more once we finished the product.