



#8: Blazing Fast Rails API

or

what to do when ActiveRecord sucks

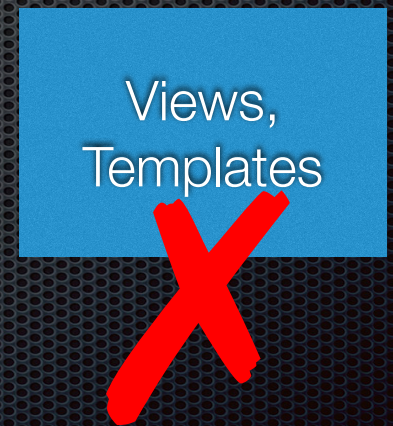
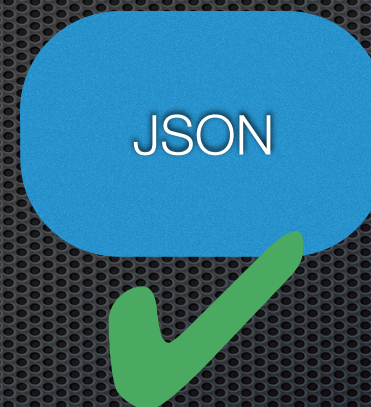
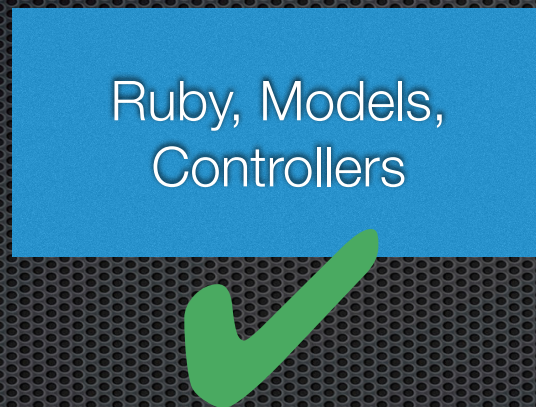
Konza Investment Tech

www.konzainvestech.com

[@vstegman](#)

https://github.com/vstegman/speedy_api_demo

What We Will Cover



“ScoreKeeper” Data Model Overview

Users

1..n

Scores

n..1

Games

- value
- date



API end points

`api/v1/users` # index for Users

`api/v1/games` # index for Games

nested resources for:

`:id/scores` # all scores for User or Game


`:id/stats` # summary stats for User or Game

Gems

- ✦ 'oj' -> Optimized JSON
- ✦ 'benchmark_suite' -> Enhances standard ruby Benchmark
- ✦ 'meta_request' -> Supports RailsPanel in Chrome
- ✦ 'descriptive-statistics' -> Mean, Mode, Standard Deviation

RailsPanel

Chrome Add In
requires 'meta_request' Gem

**RailsPanel**
offered by Dejan Simic
★★★★★ (79) [Developer Tools](#) 19,745 users

ADDED TO CHROME

OVERVIEWREVIEWSUPPORTRELATED

91

Compatible with your device

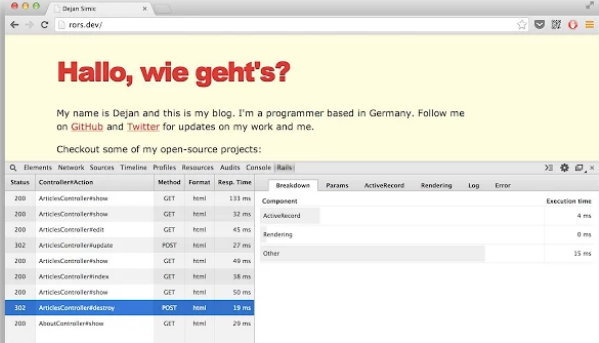
Devtools panel for Rails development

RailsPanel is a Chrome extension for Rails development that will end your tailing of development.log. Have all information about your Rails app requests right there in the Developer Tools panel. Provides insight to db/rendering/total times, parameter list, rendered views, text editor integration and more.

To use this extension you need to add meta_request gem to your app's Gemfile:

```
group :development do
  gem 'meta_request'
end
```

[Website](#)
[Report Abuse](#)
Version: 0.2.4
Updated: December 4, 2014
Size: 154KB
Language: English (United States)



Status	ControllerAction	Method	Format	Resp. Time	Component	Params	ActiveRecord	Rendering	Log	Error	Execution time
200	ArticlesController#show	GET	html	133 ms	Component						4 ms
200	ArticlesController#show	GET	html	32 ms	ActionRecord						0 ms
200	ArticlesController#edit	GET	html	45 ms	Rendering						15 ms
302	ArticlesController#update	POST	html	27 ms							
200	ArticlesController#show	GET	html	49 ms							
200	ArticlesController#index	GET	html	38 ms							
200	ArticlesController#show	GET	html	50 ms							
302	ArticlesController#destroy	POST	html	19 ms							
200	ArticlesController#show	GET	html	29 ms							

Problem #1, Rendering

```
1 class Api::V1::ScoresController < ApplicationController
2   def index
3     scores = Score.where(user_id: params[:user_id])
4     hash = {
5       meta: {user_id: params[:user_id], count: scores.length},
6       scores: scores
7     }
8     render json: hash
9   end
10 end
11
```

~/api/v1/users/2/scores

**Request takes
almost 3 seconds!
Most time used in
Rendering**

```
{
  "meta":
    {"user_id": "5", "count": 10000},
  "scores":
    [
      {"value": 44553, "date": "2002-09-10", "user_id": 5, "game_id": 10}
      etc....
    ]
}
```


Solution #1: Get Specific

```
13
14 class Score < ActiveRecord::Base
15   belongs_to :user
16   belongs_to :game
17
18   SCORE_STRUCT = Struct.new('Score', :value, :date, :user_id, :game_id)
19
20
21   def self.as_struct(struct = nil)
22     struct ||= SCORE_STRUCT
23     arr = pluck(:value, :date, :user_id, :game_id)
24     structs = []
25     arr.each {|cf| structs << struct.new(cf[0], cf[1], cf[2], cf[3])}
26     structs
27   end
28
29   def as_json(args)
30     {value: value, date: date, user_id: user_id, game_id: game_id}
31   end
32
33 end
```

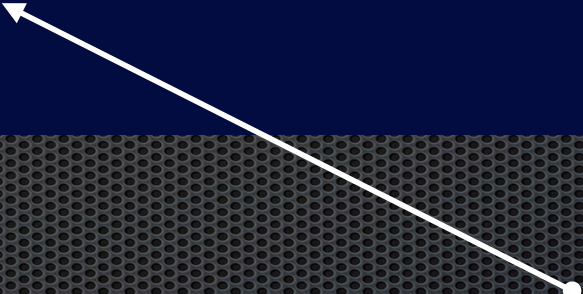
Tell the **render json:**
command exactly what to
do with an **as_json** method



Reduces request time by ~2 seconds

Solution #2: Parse Faster

```
1 class Api::V1::ScoresController < ApplicationController
2   def index
3     scores = Score.where(user_id: params[:user_id])
4     hash = {
5       meta: {user_id: params[:user_id], count: scores.length},
6       scores: scores
7     }
8     #render json: hash
9     render json: Oj.dump(hash, mode: :compat)
10  end
11 end
12
```



Oj is faster than the default
JSON rendering

Reduces request time by another 50%

Solution #3: Bypass ActiveRecord

```
1 class ScoreReader
2   attr_reader :value, :date, :user_id, :game_id
3   def initialize(args)
4     @value = args[0]
5     @date = args[1]
6     @user_id = args[2]
7     @game_id = args[3]
8   end
9
10  def self.build_from_relation(relation)
11    arr = relation.pluck(:value, :date, :user_id, :game_id)
12    obj = []
13    arr.each {|r| obj << ScoreReader.new(r)}
14    obj
15  end
16
17  def as_json
18    {
19      value: @value,
20      date: @date,
21      user_id: @user_id,
22      game_id: @game_id
23    }
24  end
25
26 end
```

Use a PORO to encapsulate your data without the overhead of ActiveRecord

Place logic in specific objects and avoid a 'God class'

Bypass ActiveRecord

```
1 class Api::V2::ScoresController < ApplicationController
2   def index
3     scores = ScoreReader.build_from_relation(
4       Score.where(user_id: params[:user_id])
5     )
6     hash = {
7       meta: {
8         user_id: params[:user_id],
9         count: scores.length
10      },
11      scores: scores
12    }
13    render json: Oj.dump(hash, mode: :compat)
14  end
15 end
```

V2 of our API incorporates our prior lessons plus ScoreReader

And we can still take advantage of ActiveRecord's querying interface

We've reduced response time from 3 seconds to 100~200ms for a 10,000 record request!

Next Challenge: Calculations

[~/api/v1/users/4/stats](#)

- ✦ For Each User / Game calculate the average (mean) score and...
- ✦ Create a “Skill Score” to show how consistently good the User is:

$$\frac{\text{average score}}{\text{standard deviation of scores}}$$

Benchmarking calculation
options with Benchmark#ips

Class Options

- Hash
- PORO
- Struct

Sum Options

- `#each` with a sum variable
- `#inject` (this is how Rails implements `Array#sum`)
- `#reduce`

Benchmarking Conclusions

- When instantiating, Hashes are fast, Structs are faster, simple PORO similar
- Array#each is fast, consider using it instead of #reduce, #inject, or #map

AdvancedStat Class

```
{
  "meta":{"user_id":"1"},
  "stats": [
    {
      "game_id":10,
      "count":1000,
      "mean":57213.383,
      "skill_score":9.840391
    },{
      "game_id":9,
      ...
    }
  ]
}
```

```
1 class AdvancedStat
2
3   def initialize(relation)
4     if(relation.is_a? ActiveRecord::Relation)
5       @stats = DescriptiveStatistics::Stats.new(relation.pluck(:value))
6     else
7       @stats = DescriptiveStatistics::Stats.new(relation)
8     end
9   end
10
11   def to_hash
12     {
13       count: @stats.length,
14       mean: @stats.mean,
15       skill_score: @stats.mean / @stats.standard_deviation
16     }
17   end
18 end
```


Class Method as_struct

Define the Struct in an initializer

```
1 Struct.new('Score', :value, :date, :user_id, :game_id)
```

Loads data into an array of faster, attribute only objects

```
24  
25 def self.as_struct(struct = nil)  
26   arr = pluck(:value, :date, :user_id, :game_id)  
27   structs = []  
28   arr.each {|cf| structs << SCORE_STRUCTURE.new(cf[0],cf[1],cf[2], cf[3])}  
29   structs  
30 end
```


Putting it all together: GroupedScore

```
1 class GroupedScore
2   def initialize(relation)
3     scores = relation.as_struct
4     @scores = scores.group_by {|s| s.game_id}
5     run_stats
6   end
7
8   def run_stats
9     @stats = []
10    @scores.each do |k,v|
11      stat = AdvancedStat.new(v.map(&:value)).to_hash
12      stat[:game_id] = k
13      @stats << stat
14    end
15    @stats
16  end
17
18  def as_json
19    @stats
20  end
21
22 end
```

Now our Controller simply needs to pass an AR::Relation of Scores to a PORO to handle the business logic and presentation

More Resources

- <http://brainspec.com/blog/2012/09/28/lightning-json-in-rails/>
- <http://www.ohler.com/oj/>
- <https://github.com/evanphx/benchmark-ips>