Sinatra is Awesome!

A presentation by Ted Price.

My Language Background

- - Java (DB)
- - PHP (Bitscribe, the Heroku guys!)
- - Ruby on Rails (GorillaNation Media)
- - Ruby (AT&T)
- - Ruby, Rails, Sinatra (theSlingshot)

Why Sinatra?

- - Ruby on Rails was too much
- - Rack was too little
- - Sinatra was just right

Why Sinatra? (cont.)

- Light weight => fast
- Straight forward
- You can understand the source code in less than a day!
- Mature community
- Lots of plugins and extensions
- Rack compliant
- Wanted to try something new

I hope to accomplish:

[not in this order]

- Simple routes, blocks
- Route patterns and matching
- Route conditions
- Config options
- Sinatra outside the box
- Templates
- Sessions, Helpers, Filters, YEEHAW!!
- Basic Testing

Lets get started!

```
$ gem install sinatra
```

```
require ”sinatra”
get ”/”
  ”hello world”
end
```

Doing stuffs and such As...

[verb] [path] [block]

Acceptable verbs:
[head/get/post/put/delete/patch/options]

Taking parameters

```
get "/hello/:name" do
   "Yo #{params[:name]}"
end


Goto:  "http://localhost:4567/hello/"
...OOOPS!
```

Conditional Parameters

Sinatra knows REGEX. In fact, they are having an affair!

```
get ”/hello/:name?” do
  ”Hello #{params[:name] || ”my lady” }”
end

get ”/list.?:format? Do
# GET /list  GET /list.html  GET /list.json...
end
```

Other fun with pattern matching

Splats!

```
get "hello/*" do params[:splat].inspect end
```

Block arguments:

```
Get "/file/*.*" do |filename,ext|
"You requested #{filename} with ext #{ext}
end
```

Route Conditions

```
get ”/”, :host_name => /^admin./ do
  ”restricted”
end


get ”/”, :agent => /.*MSIE.*/ do
  ”This website must be viewed in a browser..
  NARF!!!”
end
```

Sinatra is just Ruby, baby!

```
%w(these are all routes).each do |word|
  get "/#{word}" do word end
end
```

No, its seriously totes Ruby!

```
def get_time; Time.now.to_s end
get ”/time_now” do
  ”IT'S <b>#{get_time}</b>!!”
end
```

Handle them errors

Ways to handle different errors:

- not_found for routes
- ”error” for exceptions in the route handling block
- Create, raise and handle your own exceptions

```
get "/" do
  "Thats Boring"
end

error 403 do
  "You gots a 403 error!"
end

get "/good_route" do
  403
end

not_found do
  %{Ooops!  Bad route <a href="/">click here</a>}
end
```

Basic Configs

Static assets are served from ”public” directory or:

```
set :public_folder, ”path/to/static/assets”
```

Views are served form the ”views” directory or:

```
set :views, ”path/to/views”
```

...or specify with the ”views” rendering option (more later)

Views for DAYZIES!

”Tilt is the engine that pays may!”

Sinatra ~1.0 introduced the Tilt rendering engine.

Templates galore:

- Erb
- Haml
- Builder
- Nokogiri
- Creole
- Coffee

….

- Define your own!! (see tilt docs)

I use ERB, because of daddy issues.

[template engine] [view], [option hash]

```
get "/foo" do
  erb :bar, :layout => nil
end
```

Allow me to reintroduce, myself...

...its all just Ruby and strings

```
get "/time_now_more" do
  "This is a string: #{erb :time_now}"
end

get ”/random” do
  res =”Random number: ” ”<%= rand(1_000_000) %>”
  erb res
end

get "/mark(down)_haml" do
  res = <<-EOL
%h1 I am your.....
%p= markdown(:father)
  EOL
  haml res
end
```

Islands in the Streaming

- Introduced recently
- YMMV on certain servers and their capabilities (Thin vs Unicorns)

```
get '/' do
  stream do |out|
    out << ”Stressed out?  Lets count to 10.”
    sleep 1
    (1..10).each { |i| out << ”<br/>#{i}”; sleep 1 }
    out << "<br/>Feel better?”
  end
end
```

Set response values

- Its good to use the return vals from the called proc
- When you need to manual set values use: status, head, body
- You can also set content_type, mime_type, etc.

```
get ”/jersey_shore” do

  status 1_000_000

  headers ”Allow” => ”GET, GTL, T_SHIRT_TIME”

  body ”Time to beat the beat up!”

end
```

Passing the buck!

Pass can be used to move between route handlers

It will move execution to next pattern matched

```
def ”/hello/:name” go
  name = params[:name]

  pass if name.eql?”Ted”

  ”Hello #{name}”
end

def ”/hello” do
  ”hello world”
end

def ”/hello/*” do
  ”Shout outs to:<ul><li>#{params[:splat][0].split(”,”).join(”<li>”)}</ul>”
end
```

Halting the execution

halt [status code] [body]

halt [string]

```
get ”/good” do
  ”hello world”
end

get ”/bad” do
  halt 410, ”STOP!!”
end

get ”/awesomely_bad” do
   halt erb(:awesomely_bad)
end
```

Sessions, State and Utopia

```
enable :sessions

get ”/counter” do

  session[:counter] = session[:counter] ? session[:counter]+1 : 0

  %{Times you've been here: #{session[:counter]}<br/>

    <a href=”/counter”>reload!</a>}

end
```

Sessions, State and Utopia (cont)

Sessions use a secret:

- Randomly generated per app instance
- Multiple instances mean no shared session
- Set session secret using:

```
set :session_secret, ”i'm a pretty big kitty”
```

Sessions, State and Utopia (cont)

- Session data is stored in a cookie
- Single cookies store ~4k of info
- Depending on usage, this can get way out of hand
- Sessions are just Rack middleware, lots of options and controls

Filters

Two types: before and after

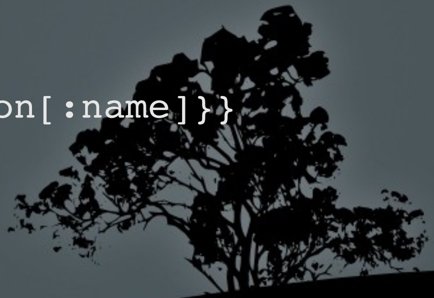
[before or after] [option hash] [block]

Filters act as a call chain!

```
before "/:name" do
  puts "Before filter /:name"
  session[:name] = params[:name]
  redirect "/"
end

before do
  puts "Before filter"
  session[:name] ||= "no name"
end

get "/" do
  %{Session[:name] was set in the before filter to: #{session[:name]}}
end
```



Helpers

[helpers] [block of method definitions]

Helper methods can be called from route handlers

```
helpers do
  def upcase_and_reverse str
    str.upcase.reverse
  end
end

get ”/:word” do
  upcase_and_reverse params[:word]
end
```

Getting Ready for Prime Time

Configuration block:

```
configure [env, ] do

  set :happy_thoughts, ”ImGonnaBeTheNextZuck”
  enable :logging   # set :logging, true
  disable :sessions # set :sessions, false

end

get ”/starbucks_pitch_to_developer” do
  10.times { ”To self: #{settings.happy_thoughts}” }

  logger.info ”Its Facebook meets 4sq meets Angry Birds.  Wanna develop the entire
   thing for 1% sweat equity? [ Yn ]”

end
```

Getting Ready for Prime Time (cont).

Top-level app to modulur in two steps:

- Require ”sinatra/base”
- Wrap methods in class that inherits from ”Sinatra::Base”

```
require "sinatra"
  get '/' do
    'Hello world!'
  end
end
```

becomes...

```
require 'sinatra/base'

class MyApp < Sinatra::Base

  get '/' do
    'Hello world!'
  end

  run! if app_file == $0  # or config.ru
end
```

Testing is Awesome Sauce!

We all love TDD, BDD, SDD, TAFTA, TDT and RiSC.

- Lets take a ganders at the basic app and test!

Info and Stuff

Edward "Ted" Price

eprice@theslingshot.com, ted.price@gmail.com

theslingshot.com when you need a break from coding!

Sinatra Resources:

- http://sinatrarb.com/
- http://recipes.sinatrarb.com/
- https://github.com/sinatra/sinatra-contrib