Introduction to Web Development

using Sinatra

Konstantinos Karasavvas

CITY College

October 30, 2013

Table of contents

About

- Sinatra
  - *Lightweight* web application framework or micro framework
  - In fact: DSL and web application library
- Minimalistic approach to development
  - handle HTTP requests
  - deliver responses
  - no ORM, no pre-fab configuration files, no project structure
- Flexibility
  - exactly as large as they should be
  - extras need to be added `manually`
  - good for understanding and learning
- Web application size
  - Good for small to medium
  - Perfect for small

Example

```
$ gem install sinatra
```

```
require 'sinatra'

get '/' do
  "Hi stranger!"
end
```

```
$ ruby my_app1.rb
[2013-06-20 16:19:39] INFO  WEBrick 1.3.1
[..] INFO  ruby 1.9.3 (2012-12-25) [x86_64-linux]
== Sinatra/1.4.3 has taken the stage on 4567 for development with↩
     backup from WEBrick
[..] INFO  WEBrick::HTTPServer#start: pid=4209 port=4567
```

- Goto: http://localhost:4567

Sinatra::Application

```
# my_app2.rb
require 'rubygems' if RUBY_VERSION < "1.9"
require 'sinatra'

class MyApp < Sinatra::Application
  get '/' do
    "Hi stranger!"
  end
end

MyApp.run!
```

```
$ ruby myapp2.rb
...
```

- Goto: http://localhost:4567

Separate Files

```
# my_app2.rb
require 'rubygems' if RUBY_VERSION < "1.9"
require 'sinatra'

class MyApp < Sinatra::Application
  get '/' do
    "Hi stranger!"
  end
end
```

```
# run_my_app.rb
require 'my_app2'

MyApp.run!
```

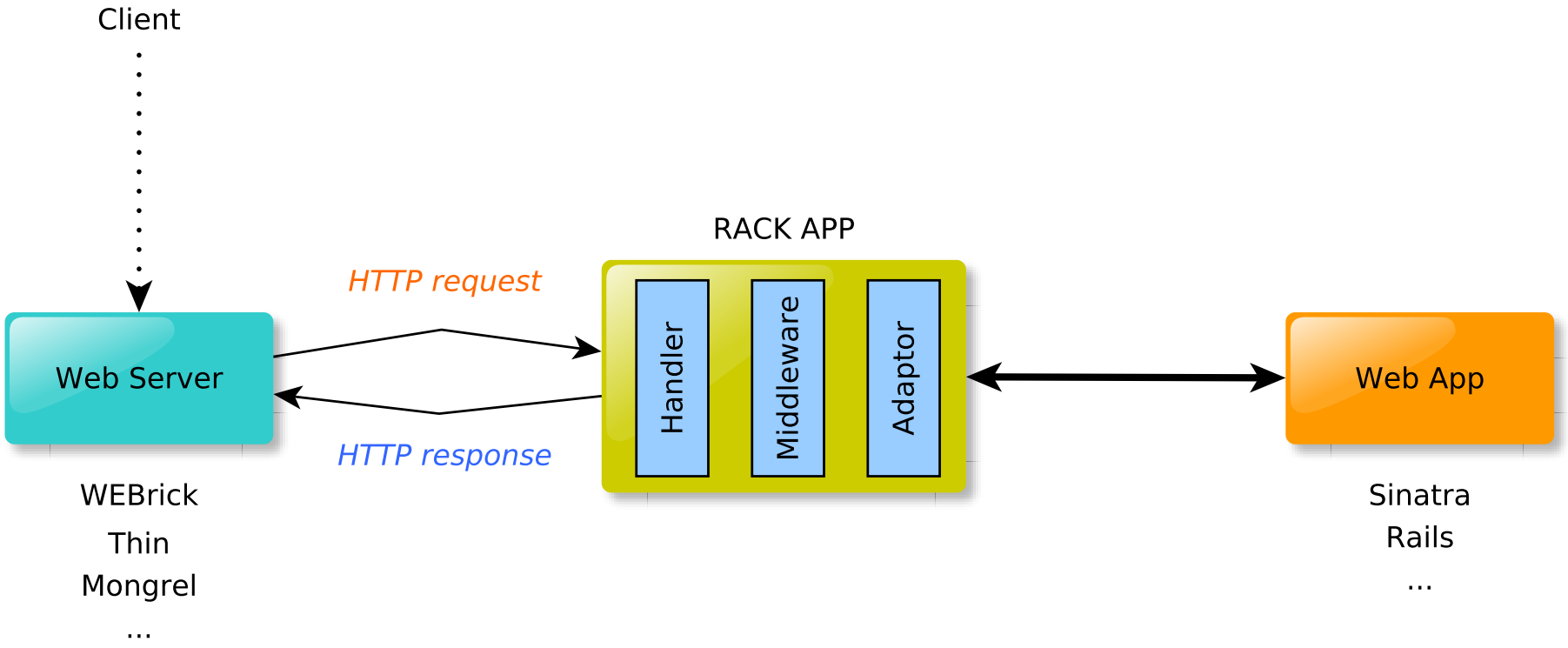
```
$ ruby run_my_app.rb
```

Rack Basics

Rack provides a minimal, modular and adaptable interface for developing web applications in Ruby. By wrapping HTTP requests and responses in the simplest way possible, it unifies and distils the API for web servers, web frameworks, and software in between (the so-called middleware) into a single method call.

- Specification
  - specifies how a Rack application and a web server should communicate
  - a Rack application is a Ruby object that responds to `call`
  - takes one argument: the **environment**
  - returns an Array with three values: **status**, **headers**, and the **body**.
- Implementation (rack gem)
  - provides basic implementations of request, response, cookies and sessions
  - includes a collection of utilities and facilitating classes to make web application development easier
  - eases middleware development

Rack Basics, cont.



Rack Example

```
require 'rack'

class MyApp
  def call(env)
    [ 200,
      {'Content-Type' => 'text/plain'},
      ['Hello from Rack!']
    ]
  end
end

Rack::Handler::WEBrick.run(MyApp.new, {:Port => 9292})
```

```
$ ruby my_rack_app.rb
```

Rackup

- Tool to run Rack applications
- Automatically figures out the environment it is run in
  - standalone with WEBrick or Thin, etc. or via FastCGI, CGI
  - can be used to configure (port, environment, etc.)
- Typically the web application is written as an independent class
- Then, `rackup` is used to run it

Rack Example with rackup

```
# my_rack_app2.rb
class MyApp
  def call(env)
    [ 200,
      {'Content-Type' => 'text/plain'},
      ['Hello from Rack!']
    ]
  end
end
```

```
# rack_config.ru
require './my_rack_app2'

run MyApp.new
```

```
$ rackup rack_config.rb
```

Sinatra Example with rackup

```
# my_app2.rb
require 'rubygems' if RUBY_VERSION < "1.9"
require 'sinatra'

class MyApp < Sinatra::Application
  get '/' do
    "Hi stranger!"
  end
end
```

```
# rack_sinatra_config.ru
require './my_app2'

run MyApp.new
```

```
$ rackup rack_sinatra_config.rb
```

Rackup options

```
$ rackup -p 8888 rack_sinatra_config.rb
```

```
$ rackup -s thin rack_sinatra_config.rb
```

```
$ rackup -D rack_sinatra_config.rb
```

```
$ rackup -P rack.pid rack_sinatra_config.rb
```

```
$ rackup -E production rack_sinatra_config.rb
```

Routes

```
1   get "/" do                          # localhost:4567/
2     redirect to('/hello')
3   end
4
5   get "/hello" do                     # localhost:4567/hello
6     "Hi stranger!"
7   end
8
9   get "/hello/:name" do |name|        # localhost:4567/hello/Kostas
10    "Hello #{name}"
11  end
12
13  get "/download/*.*" do |path, ext|   # ..4567/download/file.xml
14    [path, ext]  # => ["file", "xml"]
15  end
16
17  get %r{/hi/([\w]+)} do |c|
18    "Hi #{c}"
19  end
```

Testing Routes

- Using the browser

```
http://localhost:4567/hello/Kostas
```

- Using the `curl`
  - tool to transfer data from/to server
  - many protocols: HTTP, HTTPS, FTP, IMAP, POP, SCP, GOPHER, ...
  - very powerful: proxy, user athentication, HTTP POST, cookies, ...

```
$ curl http://localhost:4567/hello/Kostas
```

Testing Routes, cont.

```
$ curl -v http://localhost:4567/hello/Kostas
```

```
* About to connect() to localhost port 4567 (#0)
*   Trying 127.0.0.1... connected
> GET /hello/Kostas HTTP/1.1
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib↩
    /1.2.3.4 libidn/1.23 librtmp/2.3
> Host: localhost:4567
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html;charset=utf-8
< Content-Length: 12
< X-Xss-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Server: WEBrick/1.3.1 (Ruby/1.9.3/2012-12-25)
< Date: Sun, 14 Jul 2013 17:17:12 GMT
< Connection: Keep-Alive
<
* Connection #0 to host localhost left intact
* Closing connection #0
Hello Kostas
```

Routes, cont.

```
get "/hello" do
  "Hi stranger!"
end

post "/hello" do
  "#{params[:name]}, your age is #{params[:age]}"
end
```

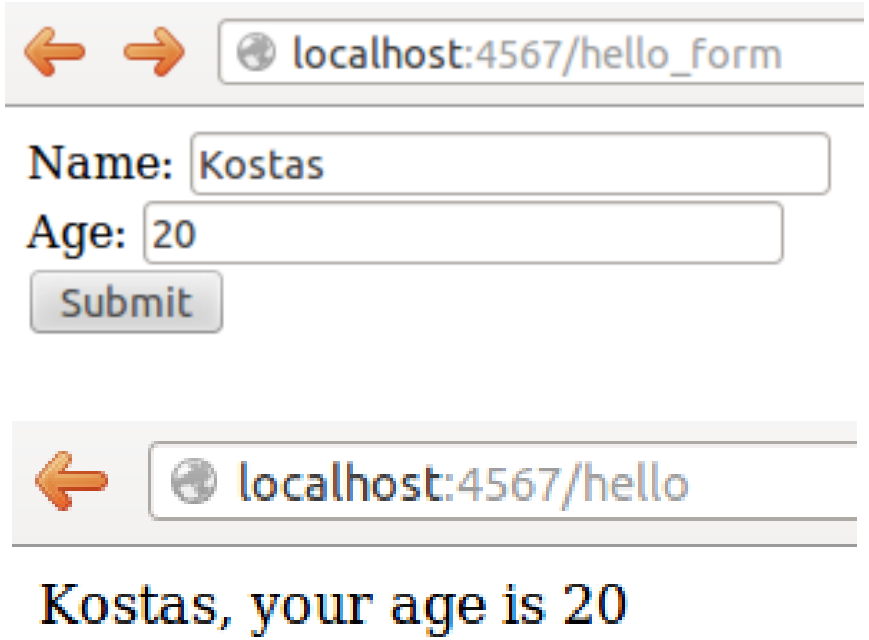
```
$ curl http://localhost:4567/hello
Hi stranger!
```

```
$ curl -X POST -d "name=Kostas&age=20" http://localhost:4567/↩
    hello
Kostas, your age is 20
```

Hello Form – HTML

```
post "/hello" do
  "#{params[:name]}, your age is #{params[:age]}"
end

get "/hello_form" do
  "<HTML>" +
    "<HEAD><TITLE>Hello Form</TITLE></HEAD>" +
    "<BODY>" +
      "<FORM ACTION=\"/hello\" METHOD=\"post\">" +
        "Name: <INPUT TYPE=\"text\" name=\"name\"><br>" +
        "Age: <INPUT TYPE=\"text\" name=\"age\"><br>" +
        "<INPUT type=\"submit\" value=\"Submit\">" +
      "</FORM>" +
    "</BODY>" +
  "</HTML>"
end
```

Hello Form – HTML, cont.



Hello Form – HTML, cont.

```
get "/hello_form" do
  "<HTML>" +
    "<HEAD><TITLE>Hello Form</TITLE></HEAD>" +
    "<BODY>" +
      "<FORM ACTION=\"/hello\" METHOD=\"post\">" +
        "Name: <INPUT TYPE=\"text\" name=\"name\"><br>" +
        "Age: <INPUT TYPE=\"text\" name=\"age\"><br>" +
        "<INPUT type=\"submit\" value=\"Submit\">" +
      "</FORM>" +
    "</BODY>" +
  "</HTML>"
end
```

- Some issues:
  - routes provide control logic
  - HTML provide presentation
  - Can quickly get out of hand and messy

Templates Introduction

- Template Languages
  - generate any kind of text from a template
  - combine plain text with variable substitution and control structures
- Common on the Web to generate HTML
- A wide variety of templates
  - ERB
  - Haml (HTML abstraction markup language)

ERB

```
<%# template.erb %>
Time is <%= Time.now %>.
```

```
$ erb -T 1 template.erb > output.txt
```

```
require 'erb'

name = "Kostas"
template = "My name is <%= name %>."

renderer = ERB.new(template)
puts renderer.result()
```

Haml

```
-# template.haml
Time is #{Time.now}.
```

```
$ gem install haml
...
Successfully installed haml-4.0.3
$ haml template.haml output.html
```

```
require 'haml'

name = 'Kostas'
template = 'My name is #{name}.'

engine = Object.new
Haml::Engine.new(template).def_method(engine, :render, :name)
puts engine.render(:name => name)
```

Simple HTML form in ERB

```
<html>
  <head>
    <title>Hello Form</title>
  </head>
  <body>
    <h1>Time now is <%= Time.now %></h1>
    <form action="/hello" method="post">
      <label for="name-id">Name:</label>
      <input id="name-id" name="name" type="text">
      <br>
      <label for="age-id">Age:</label>
      <input id="age-id" name="age" type="text">
      <br>
      <input value="Submit" type="submit">
    </form>
  </body>
</html>
```

- Notice, nothing dynamic in this view...
- Layout information is not recommended...

Simple HTML form in Haml

```
%html
  %head
    %title Hello Form
  %body
    %h1 Time now is #{Time.now}
    %form{:action => "/hello", :method => "post"}
      %label{:for => "name-id"} Name:
      %input#name-id{:type => "text", :name => "name"}
      %br
      %label{:for => "age-id"} Age:
      %input#age-id{:type => "text", :name => "age"}
      %br
      %input{:type => "submit", :value => "Submit"}
```

```
$ haml form.haml
```

- Notice, nothing dynamic in this view...
- Layout information is not recommended...

Simple HTML form in Haml, cont.

```
<html>
  <head>
    <title>Hello Form</title>
  </head>
  <body>
    <h1>Time now is 2013-10-30 17:52:16 +0200</h1>
    <form action='/hello' method='post'>
      <label for='name-id'>Name:</label>
      <input id='name-id' name='name' type='text'>
      <br>
      <label for='age-id'>Age:</label>
      <input id='age-id' name='age' type='text'>
      <br>
      <input type='submit' value='Submit'>
    </form>
  </body>
</html>
```

Tiny Sinatra App: User App

- Tiny app: store user names and respective ages

```
# user_app.rb
class UserApp < Sinatra::Application
  get "/hello" do
    "Hi stranger!"
  end

  post "/hello" do
    "#{params[:name]}, your age is #{params[:age]}"
  end

  get "/hello_form" do
    "<HTML>" +
      "<HEAD><TITLE>Hello Form</TITLE></HEAD>" +
      "<BODY>" +
        "<FORM ACTION=\"/hello\" METHOD=\"post\">" +
          "Name: <INPUT TYPE=\"text\" name=\"name\"><br>" +
          "Age: <INPUT TYPE=\"text\" name=\"age\"><br>" +
          "<INPUT type=\"submit\" value=\"Submit\">" +
        "</FORM>" +
      "</BODY>" +
    "</HTML>"
  end
end
```

Tiny Sinatra App: User App, cont.

```
# config.ru
require 'sinatra'
require './user_app'

run UserApp.new
```

- Using ERB and/or Haml templates

```
$ mkdir views
$ cp form.erb form.haml views/.
```

Tiny Sinatra App: User App, cont. (ERB)

```
# config2.ru
require 'sinatra'
require './user_app2'

run UserApp.new
```

```
# user_app2.rb
class UserApp < Sinatra::Application
  get "/hello" do
    "Hi stranger!"
  end

  post "/hello" do
    "#{params[:name]}, your age is #{params[:age]}"
  end

  get "/hello_form" do
    erb :form
  end
end
```

Tiny Sinatra App: User App, cont. (Haml)

```ruby
# config3.ru
require 'sinatra'
require 'haml'
require './user_app3'

run UserApp.new
```

```ruby
# user_app3.rb
class UserApp < Sinatra::Application
  get "/hello" do
    "Hi stranger!"
  end

  post "/hello" do
    "#{params[:name]}, your age is #{params[:age]}"
  end

  get "/hello_form" do
    haml :form
  end
end
```

VC – View-Controller

- What have we done?
  - We separated the view from the routes (controllers)
  - templates (that represent the view) are in their own files
  - Cleaner code – separation of controllers and views
- MVC – Model-View-Controller architecture
- Sinatra supports *VC* out of the box

Database Frameworks

- Relational Database
  - Postgresql, MySQL, MariaDB, **Sqlite**
  - Sqlite (simple and efficient – for learning and testing)
    - Zero configuration
    - Server-less
    - Single database file
    - Stable cross-platform database file
    - Very fast reads for single user operation
- Database Frameworks
  - Why: differences between database vendors
  - What: Abstract the database (SQL and everything...)
  - How: Implement programming API that translates to SQL and specific database calls
  - ActiveRecord, **Sequel**, DataMapper

Sequel (Just a Ruby API, no SQL)

```
$ gem install sqlite3 sequel
$ mkdir db
$ touch db/users.db
$ mkdir scripts
$ vim scripts/init_db.rb
```

```ruby
# init_db.rb
require 'sequel'

DB = Sequel.sqlite('db/users.db')

DB.create_table :users do
    primary_key :id
    String :name
    Integer :age
end
```

```
$ ruby ./scripts/init_db.rb
```

Model – The *M* of MVC

- Object-Relational Mapping (ORM)
  - Typically programming using *objects*
  - Typically storing data using *relational* tables
  - ORM *translates* between the two
- A model abstracts a *business* entity of the domain
- Programmers interact only with models (objects)

A User model using Sequel

```
$ mkdir models
$ vim models/users.rb
```

```
require 'sequel'

DB = Sequel.sqlite('db/users.db')

class User < Sequel::Model
end
```

Tiny Sinatra App: Reminder

```
# config3.ru
require 'sinatra'
require 'haml'
require './user_app3'

run UserApp.new
```

```
# user_app3.rb
class UserApp < Sinatra::Application
  get "/hello" do
    "Hi stranger!"
  end

  post "/hello" do
    "#{params[:name]}, your age is #{params[:age]}"
  end

  get "/hello_form" do
    haml :form
  end
end
```

Tiny Sinatra App: Storing a User

```ruby
# config4.ru
require 'sinatra'
require 'haml'
require './models/users'
require './user_app4'

run UserApp.new
```

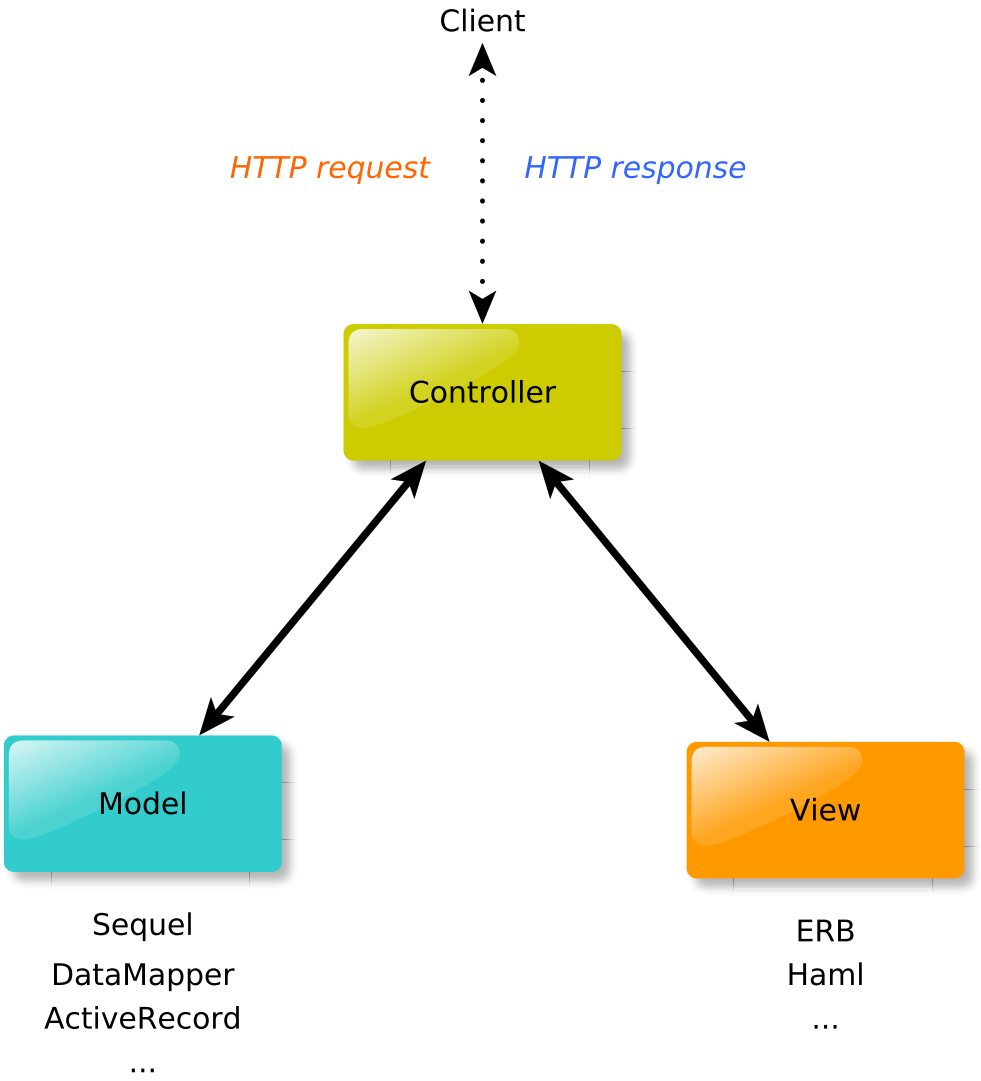
```ruby
# user_app4.rb
class UserApp < Sinatra::Application
  # ...

  post "/hello" do
    user = User.create(:name => params[:name], :age => params[:age])
    "#{user.name}, your age is #{user.age}"
  end

  get "/hello_form" do
    haml :form
  end
end
```

Tiny Sinatra App: Checking database

```
$ sqlite3 db/users.db
SQLite version 3.7.9 2011-11-01 00:52:41
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> select * from users;
1|Kostas|25
sqlite>
```

```
sequel sqlite://db/users.db
Your database is stored in DB...
1.9.3-p362 :001 > users = DB[:users]
 => #<Sequel::SQLite::Dataset: "SELECT * FROM `users`">
1.9.3-p362 :002 > users.all
 => [{:id=>1, :name=>"Kostas", :age=>25}]
1.9.3-p362 :003 >
```

Model-View-Controller (MVC)



- Software Architectural Pattern
- Web Development perspective
  - Model: represents domain (business) objects
  - View: output representation of data
  - Controller: mediates input converting it to commands for the model or view
  - Note that Model and View have no dependency
- Separation of Concerns
- Code Re-usability

Model-View-Controller (MVC), cont.

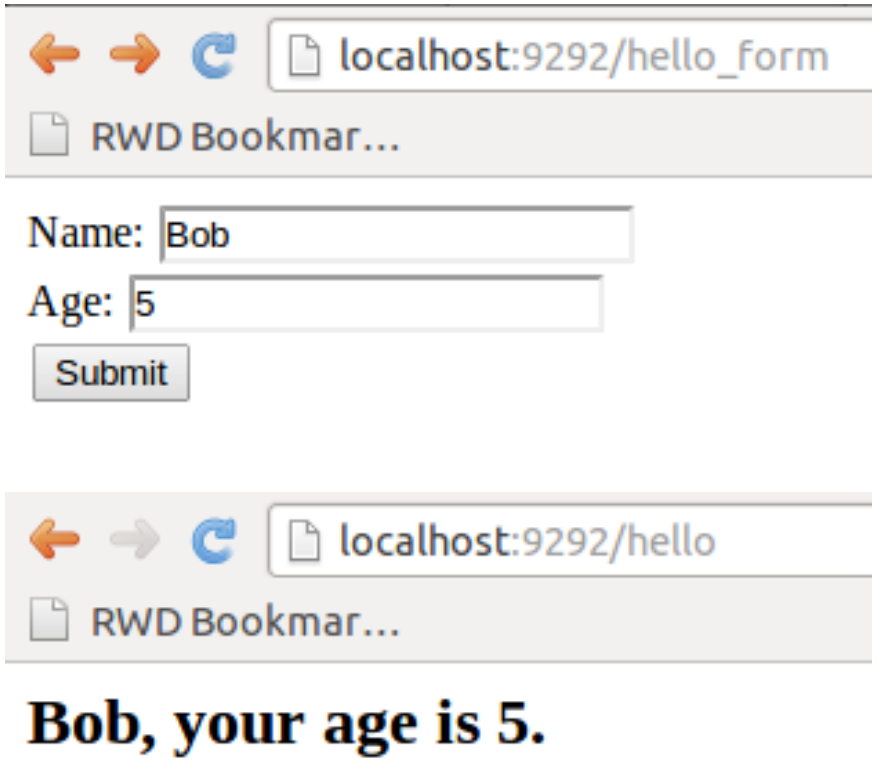
- MVC is about understanding separation of concerns
- Frameworks guide but cannot enforce proper MVC
  - A view has available only data that the controller passed to it
    - A view never uses database/Model objects directly
  - A model is completely independent of both controller and view (passive MVC)
  - The same model can be presented in multiple ways via multiple views
  - A controller (route) coordinates between model(s) and view(s)

Tiny Sinatra App: More Views

```
# user_app5.rb
class UserApp < Sinatra::Application
  # ...

  post "/hello" do
    @user = User.create(:name => params[:name], :age => params[:age])
    haml :hello
  end

  get "/hello_form" do
    haml :form
  end
end
```

```
%html
  %head
    %title Hello Stored User
  %body
    %h2= "#{@user.name}, your age is #{@user.age}."
```

Tiny Sinatra App: More Views, cont.



Tiny Sinatra App: Adding a Layout

```
%html
  %head
    %title Hello Stored User
  %body
    %h2= "#{@user.name}, your age is #{@user.age}."
```

```
%html
  %head
    %title Hello Form
  %body
    %h1 Time now is #{Time.now}
    %form{:action => "/hello", :method => "post"}
      %label{:for => "name-id"} Name:
      %input#name-id{:type => "text", :name => "name"}
      %br
      %label{:for => "age-id"} Age:
      %input#age-id{:type => "text", :name => "age"}
      %br
      %input{:type => "submit", :value => "Submit"}
```

Tiny Sinatra App: Adding a Layout, cont.

```
-# layout.haml
%html
  %head
    %title= @title
  %body
    = yield
```

```
%h2= "#{@user.name}, your age is #{@user.age}."
```

```
%form{:action => "/hello", :method => "post"}
  %label{:for => "name-id"} Name:
  %input#name-id{:type => "text", :name => "name"}
  %br
  %label{:for => "age-id"} Age:
  %input#age-id{:type => "text", :name => "age"}
  %br
  %input{:type => "submit", :value => "Submit"}
```

Tiny Sinatra App: Adding a Layout, cont.

```
# user_app6.rb
class UserApp < Sinatra::Application
  # ...

  post "/hello" do
    @title = "Hello Form"
    @user = User.create(:name => params[:name], :age => params[:age])
    haml :hello2, :layout => true
  end

  get "/hello_form" do
    @title = "Hello Stored User"
    haml :form2, :layout => true
  end
end
```

Partials

- Sub-views that can be added to multiple views
- Core re-usability
- Can complement layouts

```
= haml :partial
```

```
<%= erb :partial %>
```

Helpers

- Methods available in routes and views
- For calculations and/or HTML generation

```
# ...

helpers do
  def em(text)
    "<em>#{text}</em>"
  end
end

get '/hello' do
  @subject = 'World'
  haml :hello
end

# ...
```

```
# hello.haml
%p= "Hello " + em(@subject)
```

Rake

- Build program similar to **make**
- Typically: `Rakefile`

```
$ gem install rake
```

```
# Rakefile
task :default => [:start]

task :start do
  exec "rackup config.ru"
end
```

```
$ rake
```

Tiny Sinatra App: Automate DB creation

```
task :default => [:start]

desc 'Starts app'
task :start do
  # exec replaces current process with command
  exec "rackup config6.ru"
end

# database tasks
task :db => ["db:clean", "db:init"] do
  puts "Cleaned and initialised db schema"
end

namespace :db do

  desc 'Deletes the database'
  task :clean do
    File.delete('db/users.db') if File.exists?('db/users.db')
  end

  desc 'Creates the database schema: db is ready to be populated'
  task :init do
    touch 'db/users.db' unless File.exist?('db/users.db')
    ruby "scripts/init_db.rb"
  end

end
```

Tiny Sinatra App: Automate DB creation, cont.

```
$ rake
[2013-07-24 20:20:02] INFO  WEBrick 1.3.1
[2013-07-24 20:20:02] INFO  ruby 1.9.3 (2012-12-25) [x86_64-linux]
[2013-07-24 20:20:02] INFO  WEBrick::HTTPServer#start: pid=5229 port=9292
```

```
$ rake db
touch db/users.db
/home/karask/.rvm/rubies/ruby-1.9.3-p362/bin/ruby scripts/init_db.rb
Cleaned and initialised db schema
```

```
$ rake -D db
rake db:clean
    Deletes the database

rake db:init
    Creates the database schema: db is ready to be populated
```

Bundler

- Managing application dependencies
  - Maintaining a consistent environment for ruby applications
  - Multiple developers and/or multiple machines
- Specify dependencies: `Gemfile`

```
$ gem install bundler
```

Tiny Sinatra App: Managing Dependencies

```
source 'http://rubygems.org'

gem 'sinatra', '~> 1.3.2'
gem "sequel", "~> 3.45.0"
gem "haml"
gem "sqlite3"
```

```
$ bundle install
$ git add Gemfile Gemfile.lock
```

```
# config7.ru
require 'bundler'

Bundler.require

require './models/users'
require './user_app7'

run UserApp.new
```

```
$ bundle exec rackup config7.ru
```

Directory Structure