Real Applications in Ruby on Rails: An Introduction

Mike Gorski
mike@namedpipe.net

What I'm not covering...

- Installing and connecting to MySQL
- Mailer models
- Testing
- Routes
- Migrations/rake tasks
- MVC fundamentals

Installing Rails

```
$ sudo gem install rails
```

Create rails application

```
$ rails mytestapp

$ script/server
```

Now visit http://localhost:3000

Edit database config

```
$ vi config/database.yml
```

- Change mysql to sqlite3
- Remove other mysql stuff (user, password, host)
- Change database to dbfile

Create DB and table

```
$ sqlite3 db/development.db
Enter ".help" for instructions

sqlite> create table posts (id
integer primary key
autoincrement not null, title
varchar(100), body text);

sqlite> .exit
```

Create scaffolding

```
$ script/generate scaffold Post
```

Start the application again

```
$ script/server
```

Now visit http://localhost:3000/posts

Real World Rails

Real World Rails

- Directory structure
- Structure of URL
- Use of scaffolding
- Layouts
- Reusing views

Real World Rails (cont)

- How many controllers?
- Using subdirectories
- Other libraries
- Plugins
- Non-database models

Directory Structure

- `app/` – Logic of application in here
  - `controllers/` – these handle URL requests
  - `helpers/` – good place to put HTML snippets
  - `models/` – business objects that access the DB
  - `views/` – HTML templates to present things
- `components/` – reusable versions of app/
- `config/` – database and application config values
- `db/` – database file and database migrations
- `doc/` – RTFM
- `lib/` – other code libraries used, usually Ruby

Directory Structure

- `log/` – log files
- `public/` – publicly accessible HTML, images, JS, CSS
- `script/` – bin files to automate things
- `test/` – unit, integration, functional
- `tmp/` – session files, temp sockets, caches
- `vendor/` – 3rd party code and plugins

Structure of URL

$$\texttt{http://localhost/}\overbrace{\texttt{posts}}^{\text{Controller - posts\_controller.rb}}\texttt{/}\underbrace{\texttt{edit}}_{\text{Action or method in the controller}}\texttt{/}\texttt{1}\left.\vphantom{\Big|}\right\}\text{ID of model instance}$$

Use of scaffolding

Rather than...

```
$ script/generate scaffold Post
```

use...

```
$ script/generate controller posts
```

then edit app/controllers/posts_controller.rb...

```
class PostsController < ApplicationController
  scaffold :posts
end
```

Layouts

- HTML wrapper around dynamic portion of page
- RoR will look for layouts of "<controller>.rhtml", "application.rhtml" by default
- Another reason to use scaffold macro instead of script/generate scaffold

Layouts

app/controllers/posts_controller.rb

```
class PostsController < ApplicationController

  layout “special”

end
```

app/views/layouts/special.rhtml

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
  <title>Posts: <%= controller.action_name %></title>
  <%= stylesheet_link_tag 'scaffold' %>
</head>
<body>
<p style="color: green"><%= flash[:notice] %></p>
   <%= yield  %>
</body>
</html>
```

Layouts

app/controllers/posts_controller.rb

```
class PostsController < ApplicationController

  layout “special”

  def show
    render :template => “show”, :layout => “nonstandard”
  end

end
```

Reusing views

```
render :template => variable
```

How many controllers?

- It’s easy to end up with 1-to-1 relationship of models to controllers
- Doesn’t scale

Using subdirectories

- Using subdirectories creates modules
- These modules act like namespaces

Using subdirectories

```
$ script/generate controller admin/users
$ script/generate controller admin/posts
```

app/controllers/admin/users_controller.rb

```
class Admin::UsersController < ApplicationController
end
```

app/controllers/admin/posts_controller.rb

```
class Admin::PostsController < ApplicationController
end
```

Using subdirectories

Edit app/controllers/admin/posts_controller.rb

```
class Admin::PostsController < ApplicationController
  scaffold :post

end
```

And you’re ready to hit
http://localhost:3000/admin/posts

Other libraries

- All in lib/
- We’ve put web services integration here
- Encryption libraries

Plugins

```
$ script/plugin list

$ script/plugin install auto_complete
```

Non-database models

```
class Cart
 attr_reader :items
 attr_reader :total_price
 attr_reader :subtotal_price
 attr_reader :total_tax

end
```

Non-database models

```
class Cart
 attr_reader :items
 attr_reader :total_price
 attr_reader :subtotal_price
 attr_reader :total_tax

end
```

Non-database models

```
class ShopController < ApplicationController


  def add_to_cart
    session[:cart] = Cart.new
  end

  def cart
    @cart = session[:cart]
  end
.
.
.
```

Questions?

Resources

- http://wiki.rubyonrails.org
- http://api.rubyonrails.com
- http://api.rubyonrails.com/fr_method_index.html
- http://www.rubyonrails.org/community