INFO 2310

Topics in Web Programming

Ruby on Rails

Last week on INFO 2310

we...

- Built our first static pages
- Made them slightly more dynamic, using test driven development
- Wrote our first helper
- Played with Ruby!

Week 3

Ruby &&
ActiveRecord &&
Our First Models

Lets log in to Amazon and start our EC2 instances

Ruby basics

Hopefully, you went through the material from the end of the last lecture on your own, and started to get a feel for the (very) basics of the Ruby language.

I won't go over the nitty gritty syntax again, but I will touch on the things I would think are most new to Rubyists coming from other languages.

Blocks

Lots (and lots) of objects in Ruby respond to functions which take blocks as arguments.

Blocks are delimited either with "{" and "}" or "do" and "end"

You can think of the code inside of the block as a mini-method, and the variables between the pipes "|" are the arguments to the method.

```
sum = 0
[0,1,2,3].each { |i| sum += i }
```

So this block could be thought of as a method which takes a single argument "i", and adds it to the sum variable.

More Blocks...

```
# print hello 3 times
3.times { puts "hello, world" }

# find all the even numbers from 0 to 99
(0..99).select { |i| i.even? }

# sum all the numbers from 0 to 99 that aren't divisible by 7
sum = 0
(0..99).reject { |i| i % 7 == 0 }.each { |i| sum += i }

# double each element in an array
[1,2,3].map do |i|
  i * 2
end
```

The convention is to use curly braces "{...}" for blocks that fit on a single line, and "do...end" for blocks that span multiple lines

For Loops...

are valid ruby

```
for i in [0, 1, 2, 3, 4]
  puts "hello, world #{i}"
end
```

But they are not the ruby way.
instead....

```
5.times do |i|
  puts "hello, world #{i}"
end
```

or

```
[0,1,2,3,4].each { |i| puts "hello, world #{i}" }
```

Classes

```
class User
  def initialize(first, last)
    @first = first
    @last = last
  end

  def full_name
    "#{@first} #{@last}"
  end
end


u = User.new("Matt", "Goggin")            # Create a new object
u.full_name                                      # Output "Matt Goggin"
```

Inheritance

```
class Student < User
  def initialize(first, last, year)
    super(first, last)
    @year = year
  end

  def name_with_class
    "#{full_name} - #{@year}"
  end
end

s = Student.new("Matt", "Goggin", 2013)       # Create a new object
s.name_with_class                             # Output "Matt Goggin -  2013"
```

attr_accessor

attr_accessor adds three things to a Ruby class

- a private field
- a public getter
- a public setter

E.G.

```
class User
  attr_accessor :username
end

u = User.new
puts u.username                # => nil
u.username = "goggin13"
puts u.username                # => "goggin13"
```

Take that, Java style setters and getters!!

Back to Rails!

We'll start today with one of the most powerful features of Rails, which is ActiveRecord.

ActiveRecord is an Object Relational Mapper (ORM), which gives us a Ruby API for manipulating records in our database.

It has functions for inserting, updating, destroying, validating, finding... all without us typing any SQL ourselves.

....sweet

Why do we want an ORM?

An ORM allows us to connect records in our database with objects that we have defined in Ruby.

Each DB table (e.g. ‘posts’) has a corresponding Ruby class (e.g. ‘Post’).

Each column of the table becomes attribute of the class.
(e.g. post.content, post.title)

Each row of the table can be fetched/saved as an instance of the class.

So if we had this record in the "users" table in our database..

id	username	email
1	goggin13	mg343@cornell.edu

We can do stuff like this...

user = User.find_by_id(1)
puts user.username   # => "goggin13"

user.email = "goggin@example.com"
user.save!

Instead of writing out the SQL queries ourselves.

Login to PuTTY

- open PuTTY
- On the left panel, navigate to Connection->SSH->Auth
  - Next to "Private key file for authentication:", click "Browse", and select the *.ppk file you created on the previous step.
- Then, navigate to Connection->Data
  - For "Auto-login username", type "ec2-user"
- Navigate to "Session" (the very top)
  - Copy the "Public Domain Name" of your EC2 instance; you can see this on the instances page of the Amazon console, when an instance is selected
  - Paste it into the "Host Name (or IP address)" field
- Click "Open"

Login to WinSCP

- Open WinSCP
    - Paste in your domain to "Host name", as you did in PuTTY
    - Type "ec2-user" for the "User name"
    - Click "..." to select your private key file
    - Click "Login"
- Set NotePad++ as the default editor.
    - Click Options->Preferences
    - Select "Editors" from the left tab
    - Click Add
    - Select "External Editor"
    - Find NotePad++ (C:\Program Files (x86)\Notepad++)
    - Click "Open", then "Okay"
    - Drag it to the top of the editor list

config/database.yml

```
development:
  adapter: sqlite3
  database: db/development.sqlite3
  pool: 5
  timeout: 5000

test:
  adapter: sqlite3
  database: db/test.sqlite3
  pool: 5
  timeout: 5000
```

examples for other databases (MySQL, PostgreSQL) here->
https://gist.github.com/961978

First things first

Since we are working on a new feature today, let's start on a feature branch

```
git status                          # should display nothing to commit
git checkout -b user_model               # checkout a new branch
git branch                          # view branches
```

Back to the generator

Not surprisingly, Rails will generate the necessary files so we can start modifying them.

```
rails generate scaffold User name:string email:string --skip-helper
```

We generate everything we need to support a new model, of class User, and tell Rails it will have 2 fields of type string.

Remember other generator commands we used?

-> `rails generate controller StaticPages home new --no-test-framework`

-> `rails generate integration_test StaticPages`

Lots (and lots) of output

What did we get?

- a database migration
- a User class
    - a spec for the User class
- a UsersController
    - a spec for the UsersController
- routes for the UsersController
- views for viewing, editing, creating users
    - specs for these views

Migrations

Aside from giving us a SQL'less API for interacting with our relational database, ActiveRecord gives us tools for creating and modifying the schema as well.

These are called migrations.

db/migrate/[timestamp]_create_users.rb

```
class CreateUsers < ActiveRecord::Migration
  def change
    create_table :users do |t|
      t.string :name
      t.string :email

      t.timestamps
    end
  end
end
```

* create_table is a method which takes a symbol and a block
  * other possible function calls inside the block
    * t.integer :age
    * t.text   :bio   (see guides.rubyonrails.org/migrations.html#supported-types)
* t.timestamps creates two time fields, *created_at* and *updated_at* which Rails will maintain for us automagically.
* what's up with the timestamp?

To execute the migration, we run the following two commands:

```
bundle exec rake db:migrate
bundle exec rake db:test:prepare
```

which creates the following table in our sqlite databases (test and development):

users	
id:	integer
name:	string
email:	string
created_at:	datetime
updated_at:	datetime

What did we get?

Now that our database is ready to accept users, we can take a look around.

lets start our server

```
rails s
```

and check out

your_domain.com:3000/users

app/models/user.rb

```
class User < ActiveRecord::Base
  attr_accessible :email, :name
end
```

The class User inherits from ActiveRecord::Base (this is what will give us all the ActiveRecord goodness we're about to see).

attr_accessible dictates which properties can be modified via "mass assignment". More on that soon.

Let's test drive it

Remember "rails console" from last time?

Let's use it to explore the ActiveRecord API

rails console

Creating a user

```
User.new
=> #<User id: nil, name: nil, email: nil, created_at: nil, updated_at: nil>

user = User.new(name: "Matt", email: "mg343@cornell.edu")
=> #<User id: nil, name: "Matt", email: "mg343@cornell.edu", created_at: nil, updated_at:
nil>

user.save
=> true

user
=> #<User id: 1, name: "Matt", email: "mg343@cornell.edu", created_at: "2013-01-30 23:
10:53", updated_at: "2013-01-30 23:10:53">

user.name
=> "Matt"
user.email
=> "mg343@cornell.edu"
user.updated_at
=> Wed, 30 Jan 2013 23:10:53 UTC +00:00
```

Creating...

```
User.create(name: "A Nother", email: "another@example.org")
=> #<User id: 2, name: "A Nother", email: "another@example.org", created_at: "2013-01-
30 23:17:07", updated_at: "2013-01-30 23:17:07">

foo = User.create(name: "Foo", email: "foo@bar.com")
=> #<User id: 3, name: "Foo", email: "foo@bar.com", created_at: "2013-01-30 23:19:34",
updated_at: "2013-01-30 23:19:34"

User.count
=> 3
```

... and destroying

```
foo.destroy
=> #<User id: 3, name: "Foo", email: "foo@bar.com", created_at: "2013-01-30 23:19:34",
updated_at: "2013-01-30 23:19:34">

foo
=> #<User id: 3, name: "Foo", email: "foo@bar.com", created_at: "2013-01-30 23:19:34",
updated_at: "2013-01-30 23:19:34">

User.count
=> 2
```

Finding

```
User.find(1)
=> #<User id: 1, name: "Matt", email: "mg343@cornell.edu", created_at: "2013-01-30 23:
10:53", updated_at: "2013-01-30 23:10:53">

User.find(3)
=> ActiveRecord::RecordNotFound: Couldn't find User with id=3

User.find_by_id(3)
=> nil

User.find_by_email('mg343@cornell.edu')
=> #<User id: 1, name: "Matt", email: "mg343@cornell.edu", created_at: "2013-01-30 23:
10:53", updated_at: "2013-01-30 23:10:53">

User.find_by_name('Matt')
=> #<User id: 1, name: "Matt", email: "mg343@cornell.edu", created_at: "2013-01-30 23:
10:53", updated_at: "2013-01-30 23:10:53">
```

Finding...

```
User.all
User.first
User.last

User.all.each { |u| puts u.name }
```

Updating

```
user.email
=> "mg343@cornell.edu"

user.email = "goggin@example.com"
=> "goggin@example.com"

user.save
=> true

user.email = "williamson@example.com"
=> "williamson@example.com"

user.reload.email
=> "goggin@example.com"
```

Updating...

```
user.update_attributes(name: "The Dude", email: "dude@abides.org")
=> true
```

The ActiveRecord functions which accept a hash as an argument are the functions which are affected by the "attr_accessible" line in the User model.

Only attributes specified there can be modified in this manner. This allows us to pass incoming web (POST, PUT) requests directly into these function calls without worrying about malicious requests modifying data we do not want modified.

```
user.update_attributes(created_at: "2003-01-30 23:54:42 +0000")
=> ActiveModel::MassAssignmentSecurity::Error: Can't mass-assign protected
attributes: created_at
```

Validations

```
User.create(name: "", email: "")
=> #<User id: 2, name: "", email: "", created_at: "2013-01-31 03:09:01",
updated_at: "2013-01-31 03:09:01">
```

Does that seem okay? Do we want records in our database with blank names, and blank emails?

Probably not.

Let's fix this.

Brief TDD break

Since we are just learning lets play with the validations, then we will test them and continue.

edit ***app/models/user.rb*** to

```
class User < ActiveRecord::Base
  attr_accessible :email, :name
  validates :name, presence: true
  validates :email, presence: true
end
```

And back to the console

```
rails console

user = User.new(name: "", email: "mg343@cornell.edu")
 => #<User id: nil, name: "", email: "mg343@cornell.edu", created_at: nil,
updated_at: nil>

user.save
=> false

user.valid?
=> false

user.errors.full_messages
=> ["Name can't be blank"]
```

And now the test

Now that we know a little bit about validations, let's write some tests.

If you open up **spec/models/user_spec.rb** now, you will see

```
require 'spec_helper'

describe User do
  pending "add some examples to (or delete) #{__FILE__}"
end
```

When RSpec generates specs, it gives us the file, but marks the tests as "pending".

If you run the user_spec you will see output denoting the pending test.

```
bundle exec rspec spec/models/user_spec.rb
```

This is a big snippet, so let's just take it from the lecture_3.txt file and go through it.

If we run them

```
bundle exec rspec spec/models/user_spec.rb
```

we should be all passing

What else should we validate?

```
class User < ActiveRecord::Base
  attr_accessible :email, :name

  validates :name, presence: true,
                          length: { minimum: 4, maximum: 50 }

  VALID_EMAIL_REGEX = /\A[\w+\-.]+@[a-z\d\-.]+\.[a-z]+\z/i
  validates :email, presence: true,
                          format: { with: VALID_EMAIL_REGEX },
                          uniqueness: { case_sensitive: false }
end
```

Check out all the validators here:
http://guides.rubyonrails.org/active_record_validations_callbacks.html

In the interest of time, we will discuss but not test these. But we still know in our heart of hearts that testing is the path to robust, maintainable code and we feel appropriately guilty for skipping it.

Running all the tests

```
bundle exec rspec
```

Oh no! What happened?

When Rails generated the specs for the UsersController, it's tests did not include an email. We have since modified the User class such that every User must have an email.

Lets open up

spec/controllers/users_controller_spec.rb

```
# This line no longer meets the requirements for a valid user
def valid_attributes
  { "name" => "MyString"}
end

# so lets make it!
def valid_attributes
  { "name" => "MyString", "email" => "mystring@example.com" }
end
```

This change should get us back to green

Back to the browser

Now that our tests are passing again, let's start our server and see what the interface looks like when we input invalid data.

```
rails s
```

take a brief moment to bask in all the code we didn't have to write to achieve these results

Understanding

Let's briefly talk through all the files that made this magic happen.

user scaffolding - routes

config/routes.rb

```
resources :users
```

HTTP Verb	Path	Action	used for
GET	/users	index	lists all the users
GET	/users/new	new	displays an HTML form for creating a new user
POST	/users	create	creates a new user
GET	/users/:id	show	display a specific user
GET	/users/:id/edit	edit	displays an HTML form for editing a user
PUT	/users/:id	update	updates a specific user
DELETE	/users/:id	destroy	delete a specific user

http://guides.rubyonrails.org/routing.html

user scaffolding - controller

app/controllers/users_controller.rb

```
    def index
    def show
    def new
    def edit
    def create
    def update
    def destroy
```

user scaffolding - views

app/views/users/edit.html.erb

app/views/users/new.html.erb

app/views/users/index.html.erb

app/views/users/show.html.erb

app/views/users/_form.html.erb

users scaffold - users_controller#index

```
def index
  @users = User.all

  respond_to do |format|
    format.html # index.html.erb
    format.json { render json: @users }
  end
end
```

users scaffold - index.html.erb

```
<h1>Listing users</h1>

<table>
<% @users.each do |user| %>
  <tr>
    <td><%= user.name %></td>
    <td><%= user.email %></td>
    <td><%= link_to 'Show', user %></td>
    <td><%= link_to 'Edit', edit_user_path(user) %></td>
    <td><%= link_to 'Destroy', user, method: :delete, data: { confirm: 'Are you sure?' } %></td>
  </tr>
<% end %>
</table>

<br />

<%= link_to 'New User', new_user_path %>
```

users scaffold - users_controller#update

```
def update
  @user = User.find(params[:id])

  respond_to do |format|
    if @user.update_attributes(params[:user])
      format.html { redirect_to @user, notice: 'User was successfully updated.' }
      format.json { head :no_content }
    else
      format.html { render action: "edit" }
      format.json { render json: @user.errors, status: :unprocessable_entity }
    end
  end
end
```

users scaffold - edit.html.erb

```
<h1>Editing user</h1>

<%= render 'form' %>

<%= link_to 'Show', @user %> |
<%= link_to 'Back', users_path %>
```

users scaffold - _form.html.erb

```
<%= form_for(@user) do |f| %>
  <% if @user.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@user.errors.count, "error") %> prohibited this user from being saved:</h2>
      <ul>
      <% @user.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
      </ul>
    </div>
  <% end %>

  <div class="field">
    <%= f.label :name %><br />
    <%= f.text_field :name %>
  </div>
  <div class="field">
    <%= f.label :email %><br />
    <%= f.text_field :email %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

Time to commit

```
git status      # see what we modified
git add -A     # add all the changes
git commit -m "User model"

# merge it back into master
git checkout master
git merge user_model

git push origin master          # github
git push heroku master    # heroku


# we also need to run our migrations on Heroku
heroku run rake db:migrate
```

Assignment 1

Due Tuesday night at 11:59pm

You try!

Use the "rails scaffold" command to Create a **MicroPost** class that has two attributes

user_id:integer

content:string

-> both attributes are required

-> content should between 5 and 140 characters

After you write the validations, your MicroPost class should pass this spec

https://gist.github.com/goggin13/4717890

which you can download and use to replace your generated file at

spec/models/micro_post_spec.rb

When you're done, push it to GitHub and Heroku

Today we...

- Learned about ActiveRecord
- Learned about validations
- Saw some ERB, and learned what a partial is
- Wrote the User Model

OH

Now until 4pm

Monday from 7-8pm