INFO 2310

Topics in Web Programming
Ruby on Rails

Last week on INFO2310

- Learned about ActiveRecord
- Learned about validations
- Saw some ERB, and learned what a partial is
- Wrote the User Model together
- You added the MicroPost model

Week 4

Relations
&& Authenticating Users

Go ahead and login to Amazon and start your EC2 instances

Survey Feedback

* Less copying and pasting code
* Understanding the things we will be doing before we do them.
* Have more people on hand to walk around
* Project your voice
* Still haven't really LEARNED the syntax of Ruby
* Slow down
* Go faster

Unfortunately today's lecture was pretty solidified before I could act on these suggestions:

But next week, I will try to incorporate more exercises, and we can play with introducing a concept, seeing an example, and then having you code it in your app.

I think that we could get perhaps find a good flow if I introduce a concept, give you thorough specs, and then you can work on making them pass. So next week we can try to do more of that.

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

Today's branch

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

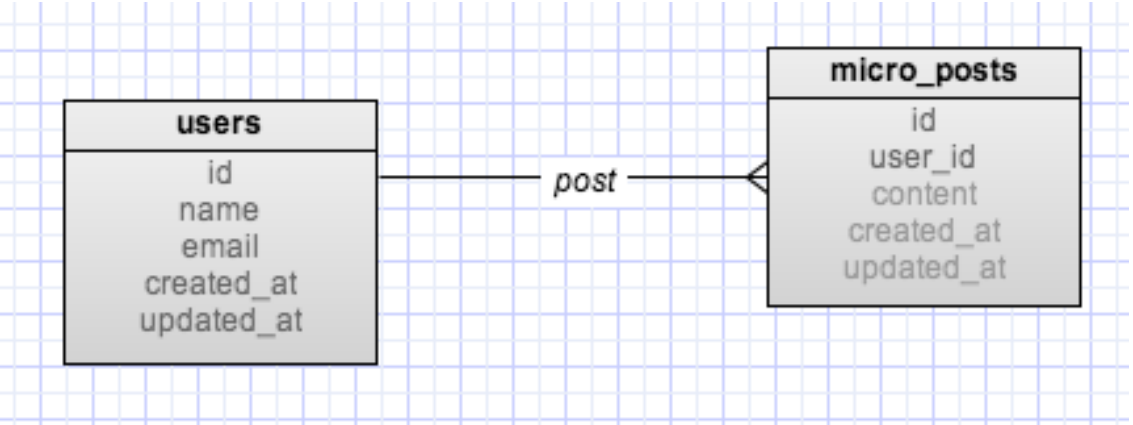
```
git status                  # should display nothing to commit
git checkout -b bootstrap   # checkout a new branch
git branch                  # view branches
```

ActiveRecord relations

We would like to achieve the following relationships:

A User has many MicroPosts
A MicroPost belongs to a user

The micro_posts table is already setup to handle this, with the user_id field ready to point to the owning user.



ActiveRecord relations

We can accomplish the first by adding the following line to our users model:

has_many :micro_posts

This line tells ActiveRecord that it should look for a "user_id" column on the "microposts" data, and it can load THIS user's microposts by querying that table for this user's id.

ActiveRecord relations

And the converse of that relationship is added with the following line in app/models/micro_post.rb

belongs_to :user

This line tells ActiveRecord that it can find the User that this MicroPost belongs to by looking at the "id" column of the "users" table that matches the "user_id" column of this MicroPost.

What do we get?

Ruby (ActiveRecord) methods to query, update, destroy these related models.

```
# querying
user.micro_posts
user.micro_posts.count
user.micro_posts.each do |mp|
  puts mp.content
end

micro_post.user

# Which can be chained...
# the username of the user who posted the first comment
post.comments.first.user.username
```

```
# creating

# user_id is automatically set to the id of the user object
user.micro_posts.create content: "hello world"

# building
# (makes a new unsaved object)
user.micro_posts.build content: "hello world"
```

User Profiles

It would be nice to see all of a user's micro posts on their profile page.

Let's do that.

First, the test

spec/requests/user_spec.rb

```
describe "GET /users/id" do

  before do
    @user = User.create! name: "Matt", email: "goggin13@gmail.com"
    3.times { |i| @user.micro_posts.create! content: "hello world - #{i}" }
  end

  it "should display the number of posts the user has" do
    visit user_path(@user)
    page.should have_content("3 MicroPosts")
  end

  it "should list the content for each micro post " do
    visit user_path(@user)
    3.times do |i|
      page.should have_content "hello world - #{i}"
    end
  end
end
```

Partials

We first saw partials last week, with the **app/views/user/_form.html.erb**

Partials are pieces of a view that you can reuse in other views (allowing you to keep everything nice and DRY).

Since we will be displaying micro_posts in a few places, let's make a partial for that.

app/views/micro_posts/_micro_post.html.erb

```
<p>
  <span class='micro_post_timestamp'>
     <%= micro_post.created_at.strftime("%m/%d %l-%M") %>
   </span>
   <%= micro_post.content %>
</p>
```

Rendering collections with partials

app/views/users/show.html.erb

```
<p>
  <%= pluralize(@user.micro_posts.length, "MicroPost") %>
</p>

<%= render @user.micro_posts %>
```

That last line seems pretty magical... We are passing a collection of objects to a render function (we usually pass the name of a partial). What's happening?

Rails can infer that **@user.micro_posts** is a collection of objects of type MicroPost. From there, it iterates the collection, and for each item, it renders the template named for that class

(in our case **app/views/micro_posts/_micro_post.html.erb**)

and passes along the current item to render in the **micro_post** variable.

Back to green

bundle exec rspec spec/requests/user_spec.rb
bundle exec rspec

Let's see it in the browser

Let's start our server, create a user, and create some microposts for that user.

rails s

Time for some stylin'

I... cannot design my way out of a hole in the ground.

TwitterBootstrap to the rescue!

TwitterBootstrap is a front-end framework with attractive base styles and layouts we can use to start.

There's a gem for that

open up your Gemfile (at the root of your app).

below the line for the rails gem, add the bootstrap gem:

gem 'rails', '3.2.11'
gem 'bootstrap-sass', '~> 2.2.2.0'

Then, run
bundle install
in your PuTTY terminal to install the bootstrap gem

Now... lots of HTML

In order to reap the benefits of our new stylin' framework, we need to make some markup changes.

This isn't conceptually interesting, but it will be nice going forward to have our site laid out so we that can start to fill in functionality.

Copy/Paste party

There is a lot of noisy HTML, not worth reproducing on the slides.

Let's talk through each file as we transfer it over from the **lecture_4.txt** file.

app/views/layouts/application.html.erb
app/views/layouts/_footer.html.erb
app/views/layouts/_header.html.erb
app/views/static_pages/about.html.erb
app/views/static_pages/help.html.erb
app/views/static_pages/home.html.erb

app/assets/stylesheets/custom.css.scss

application.css && application.js

Before we admire our handiwork, let's talk briefly about what's happening with our CSS and JS files here.

Let's look at **application.css**; application.js operates in an identical manner.

app/assets/stylesheets/application.css

```
/*
 *= require_self
 *= require_tree .
 */
```

=require_self includes any css in this file

=require_tree . includes all the files in this directory and all the subdirectories.

instead you could require specific files

=require 'matts_styles'

or different specific directories

= require_tree 'other_dir'

This line from app/views/layouts/application.html.erb uses the **application.css** to decide which files to output

```
<%= stylesheet_link_tag    "application", :media => "all" %>
```

.css.scss

What's with the funny extensions here?

SASS = "Syntactically Awesome StyleSheets"
http://sass-lang.com/

You can opt out of using SASS by removing the ".scss" extension.

But SASS is a superset of CSS, so you can also just write CSS in this file and ignore the ".scss"

.js.coffee

CoffeeScript; "a little language that compiles into javascript"

http://coffeescript.org/

CoffeeScript is not (a la SASS and CSS) a superset of JavaScript; if you wish to use vanilla JS you must remove the ".coffee" extension.

The root path

Currently, the root path of our application is public/index.html

the default Rails welcome page.

Time to graduate to our own home page!

root

Adding this line to our **config/routes.rb** file

root :to => 'static_pages#home'

tells Rails how to route incoming requests for the root domain.

Recall that Rails will always give priority to the files in the **public** directory. So we also need to delete the file at **public/index.html**

And now finally....

rails s

Looks great!

or good enough, at least.

With that many changes, I'd bet we would all feel better if we ran the tests again before moving forward.

bundle exec rspec

So what's next?

Currently our app has no concept of authentication, or of whom the current user is.

We'll need to fix that before we can push forward on the other micro posting features.

Users probably need passwords

But... passwords are different.

We want a to allow setting a *password* field on our model, but we don't want to actually save it in plain text. We want to save an encrypted version.

How should we go about this?

First, columns for the encrypted password and salt

```
rails generate migration add_hashed_password_and_salt_to_users
hashed_password:string salt:string
```

creates this file

```
class AddHashedPasswordToUsers < ActiveRecord::Migration
  def change
    add_column :users, :hashed_password, :string
    add_column :users, :salt, :string
  end
end
```

```
bundle exec rake db:migrate          # applies the changes
bundle exec rake db:test:prepare    # and to the test database as well
```

password tests - spec/models/user_spec.rb

```
describe "without a password" do
  before do
    @user.password = ""
  end

  it "should not be valid" do
    @user.should_not be_valid
  end
end

describe "hashed_password" do
  it "should be populated after the user has been saved" do
    @user.save
    @user.hashed_password.should_not be_blank
  end
end

describe "salt" do
  it "should be populated after the user has been saved" do
    @user.save
    @user.salt.should_not be_blank
  end
end
```

Recall "attr_accessor :password" adds a field to the user class which we can then populate.

Adding :password to attr_accessible allows us to set that field via the hash style function calls we played with earlier.

e.g. **user.update_attributes! password: "new_password"**

app/models/user.rb

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

This allows us to store a password field on a user object in memory, but it will not be persisted to the database

You can copy the full text of **app/models/user.rb** from the lecture_4.txt file, which includes the changes from the next two slides as well.

Callbacks

Now we have a password field in memory. Next, we need to use that field to generate and store the salt and hashed password to the database.

To achieve the desired functionality, we'll use something called callbacks.

Callbacks provide us places to hook into the process of saving, updating, destroying an object.

e.g. before_save, after_destroy, before_validation, many more

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

before_save

This one seems like it could serve our purpose
lets add this line of code to the user class:

before_save :encrypt_password

as well as these two functions

```
def encrypt_password
      self.salt ||= Digest::SHA256.hexdigest("--#{Time.now.to_s}- -#{email}--")
      self.hashed_password = encrypt(password)
end

def encrypt(raw_password)
     Digest::SHA256.hexdigest("--#{salt}--#{raw_password}--")
end
```

This should get us to green. The full code for the user class is in lecture_4.txt

||=

A common Ruby idiom for setting a variable if it's not already set.

You are familiar with "+="

x += y

~>

x = x + y

x ||= some_expensive_function()

~>

x = x || some_expensive_function()

some_expensive_function() is only evaluated if x is false

Now we update the front-end

app/views/users/_form.html.erb

add a field for the password

```
<div class="field">
  <%= f.label :password %><br />
  <%= f.password_field :password %>
</div>
```

Full Test Suite

Let's run all the tests and clean up any failures we see.

bundle exec rspec

Recall we had a similar issue before; often when we change what it means to be a valid object, we will have to update tests that create those objects

```
# spec/controllers/user_controller_spec.rb has a valid_attributes function that
# needs to be updated:
def valid_attributes
  { "name" => "MyString", "email" => "matt@hotmail.com", "password" => "foobar" }
end

# And similarly in spec/requests/user_spec.rb:
@user = User.create! name: "Matt",
                     email: "goggin13@gmail.com",
                     password: "foobar"
```

Now we should be okay again

bundle exec rspec

All our tests are passing

So let's try it out

Start your server

rails s

head to your_domain.com:3000/users

We are in a little bit of a funny state, since we just added a validation for passwords, but none of our current users have passwords (so they are all invalid).

Let's edit a user, and give him a password. And then try to change his email.

what gives?

Oh No A Bug!

But all our tests are passing...

This is bad.

Bug fixing in TDD:

1) Write a Failing Test
2) Fix Bug
3) Tests Pass
4) high fives

spec/models/user_spec.rb

```
describe "with valid attributes" do

  it "should be valid" do
    @user.should be_valid
  end

  it "should be valid if it has an encrypted_password but no password" do
    @user.save
    @user.password = nil
    @user.should be_valid
  end
end
```

Remember to run **bundle exec spec/models/user_spec.rb**
to be sure this test is failing

Conditional Validations

In **app/models/user.rb**

We can change

validates :password, presence: true

to

validates :password, presence: true, if: "hashed_password.blank?"

The if clause says to only run the validation IF the string evaluates to true. Note it is evaluated in the context of the user object being validated.

There are more advanced types of conditional validation, including passing blocks or methods names; check the rails guide for details.

http://guides.rubyonrails.org/active_record_validations_callbacks.html#conditional-validation

Let's try again

Now let's ensure we can edit a user's email without re-entering their password every time.

Authenticate

Last thing today; looking ahead, we know we are going to want to authenticate users from a login page.

We will be given an email, and a password, and we want to return a user that matches those, or nil.

Authenticate specs

```
describe "authenticate" do

  before do
    @user.save
  end

  it "should return the user with correct credentials" do
    User.authenticate(@user.email, @user.password).should == @user
  end

  it "should return nil if the given email does not exist" do
    User.authenticate("noone@example.com", @user.password).should be_nil
  end

  it "should return nil if the wrong password is provided" do
    User.authenticate(@user.email, "wrong_password").should be_nil
  end
end
```

again, full code for the user spec is in lecture_4.txt

Authenticating

Your turn!! Take 5 minutes, and then we'll go over together.

```
# If there is a user in the database with the given email, and
# the password matches theirs, returns the user.
# Otherwise, returns nil
def self.authenticate(email, plain_text_password)


end
```

self.method_name denotes a class method in Ruby; so we can call this method as **User.authenticate**

Commit time

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


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

git push origin master              # github
git push heroku master              # heroku
heroku run rake db:migrate          # apply database migrations to production
```

Today we...

- Added our first ActiveRecord relation
- Used ActiveRecord callbacks and a conditional validator
- Wrote the User.authenticate function

OH
Now until 4pm
Monday from 7-8pm