

# **INFO 2310**

Topics in Web Programming  
Ruby on Rails

# Survey

What should we do with our last 2 sessions together?

(link also available on the course website)

<https://docs.google.com/forms/d/1ucc5yN7KrY4dPLgq1qYTTTud4yCMOxSGpQYoG3TE070/viewform>

I have to leave right at 2:40 today to catch a flight\*, so I unfortunately will **not be available after class** as I normally am; shoot me an email if you have any questions or need help getting caught up!

\* flying to an interview, NOT leaving early for Cancun! #stayInSkool

# Week 8

Following  
&& Unfollowing

Go ahead and start your EC2 instances

# 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 following-users

# checkout a new branch

git branch

# view branches

# Lecture 8 specs

There are 3 specs to download for today;

[https://raw.githubusercontent.com/goggin13/curails-mg343/master/spec/requests/relationship\\_spec.rb](https://raw.githubusercontent.com/goggin13/curails-mg343/master/spec/requests/relationship_spec.rb)

=> **spec/requests/relationship\_spec.rb**

[https://raw.githubusercontent.com/goggin13/curails-mg343/master/spec/models/user\\_spec.rb](https://raw.githubusercontent.com/goggin13/curails-mg343/master/spec/models/user_spec.rb)

=> **spec/models/user\_spec.rb**

[https://raw.githubusercontent.com/goggin13/curails-mg343/master/spec/models/relationship\\_spec.rb](https://raw.githubusercontent.com/goggin13/curails-mg343/master/spec/models/relationship_spec.rb)

=> **spec/models/relationship\_spec.rb**

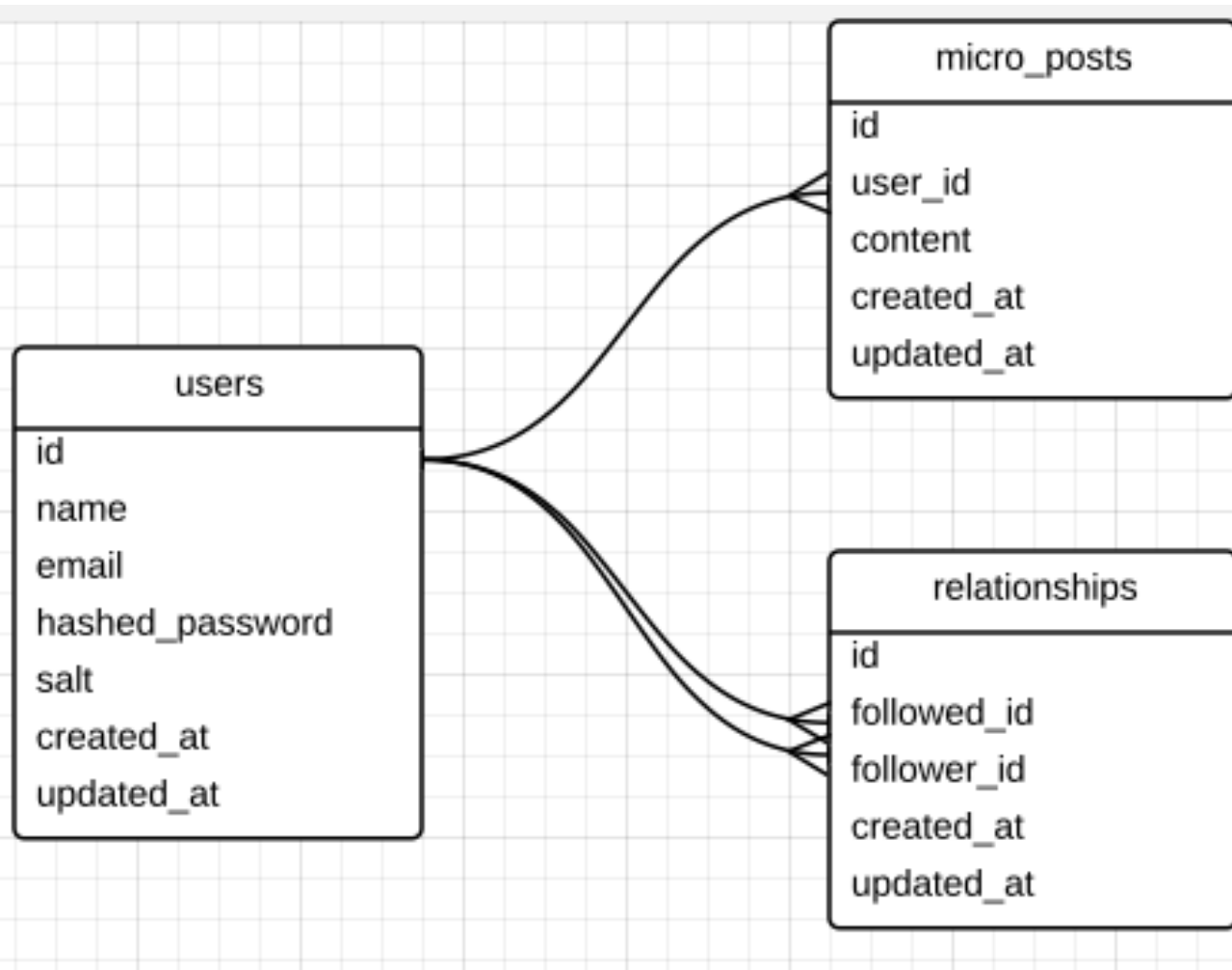
# Following Users

Time to add some social to our app; users should be able to follow/unfollow one another.

The home page will display the feed of all the users that you are following.



# ER Diagram



# Generate Migration

```
rails generate model Relationship follower_id:integer followed_id:integer
```

```
class CreateRelationships < ActiveRecord::Migration
  def change
    create_table :relationships do |t|
      t.integer :follower_id
      t.integer :followed_id

      t.timestamps
    end

    add_index :relationships, :follower_id
    add_index :relationships, :followed_id
    add_index :relationships, [:followed_id, follower_id], unique: true
  end
end
```

# Run the migrations

```
bundle exec rake db:migrate
```

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

# Our Relationship model

```
class Relationship < ActiveRecord::Base
  attr_accessible :followed_id, :follower_id

  belongs_to :follower, class_name: "User"
  belongs_to :followed, class_name: "User"

  validates :follower_id, presence: true
  validates :followed_id, presence: true
  validates_uniqueness_of :follower_id, scope: :followed_id
end
```

**bundle exec rspec spec/models/relationship\_spec.rb**  
should be passing

Notice we are having to start to do some "configuration" in the "convention over configuration". More in the next slide...

# Add relationships to our User model

```
class User < ActiveRecord::Base
```

```
  has_many :micro_posts
```

```
  has_many :relationships, foreign_key: "follower_id", dependent: :destroy
```

```
  has_many :followed_users, through: :relationships, source: :followed
```

```
end
```

**bundle exec rspec spec/models/user\_spec.rb -e "relationships"**

should be passing

follower_id	followed_id
4	5

id	email	name
5	skl83@cornell.edu	Sam
4	mg343@cornell.edu	Matt

"Matt is following Sam"

# ActiveRecord relation functions

We'll use the examples of a User and MicroPosts  
a User has\_many MicroPosts

```
user = User.find(1)
```

```
# Return a micro post by this user with the content "Hello, World"  
user.micro_posts.find_by_content("Hello, World")
```

```
# Create a new micro post for this user with the content "Hello, World"  
user.micro_posts.create!(content: "Hello, World")
```

```
# Locate and then destroy a micro post by this user with the content "Hello, World"  
user.micro_posts.find_by_content("Hello, World").destroy
```

# Using the API from the above slide, create these functions on the user object:

```
class User < ActiveRecord::Base
```

```
  # Returns the Relationship object this user has with other_user
```

```
  # or nil if no relationship exists
```

```
  def following?(other_user)
```

```
  end
```

```
  # create a Relationship object where this user is following other_user
```

```
  def follow!(other_user)
```

```
  end
```

```
  # destroy the Relationship object where this user is following other_user
```

```
  def unfollow!(other_user)
```

```
  end
```

```
end
```

**bundle exec rspec spec/models/user\_spec.rb -e "following functions"**

should be passing

**HINT: They are all one line functions!!**



# A follow form

**app/views/relationships/\_form.html.erb**

```
<% unless current_user == @user %>
  <div id="follow_form">
    <% if current_user.following?(@user) %>
      <%= render 'unfollow' %>
    <% else %>
      <%= render 'follow' %>
    <% end %>
  </div>
<% end %>
```

# A follow button

## **app/views/relationships/\_follow.html.erb**

```
<%= form_for(current_user.relationships.build(followed_id: @user.id)) do |f| %>
  <div>
    <%= f.hidden_field :followed_id %>
  </div>
  <%= f.submit "Follow", class: "btn btn-large btn-primary" %>
<% end %>
```

# An unfollow button

## **app/views/relationships/\_unfollow.html.erb**

```
<%= form_for(current_user.relationships.find_by_followed_id(@user),
  html: { method: :delete }) do |f| %>
  <%= f.hidden_field :followed_id %>
  <%= f.submit "Unfollow", class: "btn btn-large" %>
<% end %>
```

# Routes for our forms

```
INFO2310::Application.routes.draw do
```

```
  .  
  .  
  resources :sessions,    only: [:new, :create, :destroy]
```

```
  resources :relationships, only: [:create, :destroy]
```

```
  .
```

```
  .
```

```
end
```

```
class RelationshipsController < ApplicationController
```

```
  # params[:relationship][:followed_id] contains the id of the user to follow;  
  # use our functions from the last exercise to have the current_user follow them
```

```
  def create
```

```
    respond_to do |format|
```

```
      format.html { redirect_to @user }
```

```
      format.js
```

```
    end
```

```
  end
```

```
  # params[:relationship][:followed_id] contains the id of the user to follow;  
  # use our functions from the last exercise to have the current_user UNfollow them
```

```
  def destroy
```

```
    respond_to do |format|
```

```
      format.html { redirect_to @user }
```

```
      format.js
```

```
    end
```

```
  end
```

```
end
```

# Finally, add the form to a profile page

Render the follow form on a profile page, if there is an authenticated use

**app/views/users/show.html.erb**

```
<% if signed_in? %>
```

```
  <%= render 'relationships/form' %>
```

```
<% end %>
```

# Before we AJAXify

Let's test we got it right without AJAX

```
bundle exec rspec spec/requests/relationships_spec.rb
```

And we can try it out in our browser as well

# Script partial for create

**app/views/relationships/create.js.erb**

```
$("#follow_form").html("<%= j(render('relationships/unfollow')) %>")
```

# Script partial for destroy

**app/views/relationships/destroy.js.erb**

```
$("#follow_form").html("<%= j(render('relationships/follow')) %>")
```

# And set the 'remote' flag on our forms

## **app/views/relationships/\_follow.html.erb**

```
<%= form_for(current_user.relationships.build(followed_id: @user.id)) do |f| %>
```

```
=>
```

```
<%= form_for(current_user.relationships.build(followed_id: @user.id), remote: true) do |f| %>
```

## **app/views/relationships/\_unfollow.html.erb**

```
<%= form_for(current_user.relationships.find_by_followed_id(@user),  
             html: { method: :delete }) do |f| %>
```

```
=>
```

```
<%= form_for(current_user.relationships.find_by_followed_id(@user),  
             remote: true,  
             html: { method: :delete }) do |f| %>
```



# Try it out

Now we should be able to follow/unfollow via  
AJAX

# Where statements

# A collection of all the users who have the username 'goggin13'

```
User.where('username = ?', 'goggin13')
```

# A collection of users who do not have a nil email and whose ids are in the  
# array [1,2,3,4], ordered by email descending.

```
ids = [1,2,3,4]
```

```
User.where('email != ? and id in (?)', nil, ids)  
  .order('email DESC')
```

# You can always add the 'paginate' call to the end of these functions

```
User.where('username = ?', 'goggin13').paginate(paginate_options
```

# And update our existing feed function

```
class User < ActiveRecord::Base
```

```
  # Update our feed function so that it returns all of the MicroPosts  
  # that were created by this user OR users this user is following,  
  # ordered by created_at descending.
```

```
  def feed(paginate_options)  
    micro_posts.paginate(paginate_options)  
  end
```

```
end
```

```
# This may be useful...
```

```
followed_user_ids = followed_users.map { |u| u.id }
```

```
bundle exec rspec spec/models/user_spec.rb -e "feed"
```

will pass when you're done

# **Our feed is done!**

Let's try it out.

We should be able to follow and unfollow other users and see their MicroPosts displayed on our home page.

# Live updates

It would be cool if we could automatically refresh a user's feed whenever a user they are following creates a new post.

We can achieve this by "polling" our server for updates.

Here's our strategy:

- Every X seconds, via JavaScript, we look at the home page and retrieve the ID of every MicroPost that is currently being displayed
- We make an AJAX call with those IDs to a new controller function
- The controller function will query for the user's feed, and if it finds any MicroPosts whose ID is NOT in the list we just sent, it will execute the JavaScript necessary to render them out onto the page.

# A new route

First, let's add a route for our new controller function; note in this case ORDER MATTERS. The new route must go above 'resources :micro\_posts'

## **config/routes.rb**

```
get '/micro_posts/refresh'  
resources :micro_posts
```

*Why does the order matter?*

**resources :micro\_posts** generates a URL pattern that looks like this:

**/micro\_posts/:id**

The path to a single MicroPost.

This pattern matches **/micro\_posts/refresh** as well. So if the **/micro\_posts/:id** pattern was listed first in the routes file then **/micro\_posts/refresh** would be unreachable.

# The controller function

**app/controllers/micro\_posts\_controller.rb**

```
# GET /micro_posts/refresh?ids=[1,2,3,4,5]
```

```
def refresh
```

```
  feed = current_user.feed(page: 1)
```

```
  @new_micro_posts = feed.reject { |p| params[:ids].include?(p.id.to_s) }
```

```
  respond_to do |format|
```

```
    format.js
```

```
  end
```

```
end
```

# The JavaScript

Is a little more involved;

Let's grab the code from the lecture\_8.txt file and walk through it.



# Try it out!

In order to see it in action, we can use the Rails console to create some new posts from someone we are following. We'll also take this opportunity to demonstrate how powerful (convenient) chaining together ActiveRecord commands can be.

```
User.find_by_email('goggin13@gmail.com') # retrieve my user's by their email
    .relationships                        # collection of my relationships
    .first                             # get the first relationship
    .followed                          # get the user being followed
    .micro_posts                       # get their collection of micro_posts
    .create! content: "Do you love chaining too?" # and create a new one
```

# Commit time

git status # see what we modified

git add -A # add all the changes

git commit -m "authentication"

git checkout master # merge it back into master

git merge following-users

git push origin master # github

git push heroku master # heroku

# execute this command and follow the instructions closely  
for i in {1..10}; do echo 'have a great Spring break!!!'; done

# Today we...

- Created a new Relationship model
- allowed users to follow/unfollow each other via AJAX
- implemented an auto-refresh feature for our feed with AJAX polling