# 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         # checkout a new branch
git branch                        # view branches
```

## Lecture 8 specs

There are 3 specs to download for today;

https://raw.github.com/goggin13/curails-mg343/master/spec/requests/relationships_spec.rb
=> **spec/requests/relationships_spec.rb**

https://raw.github.com/goggin13/curails-mg343/master/spec/models/user_spec.rb
=> **spec/models/user_spec.rb**

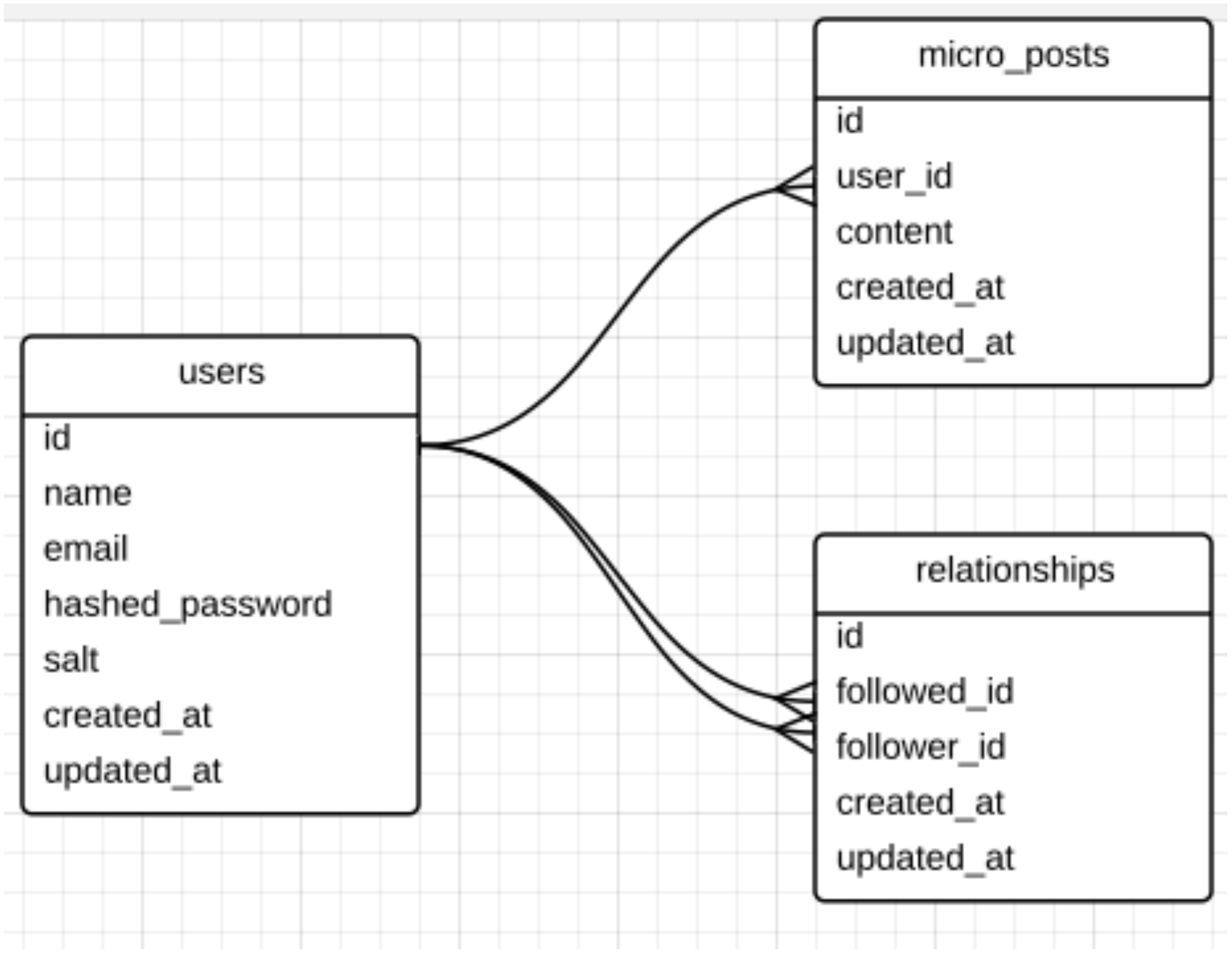
https://raw.github.com/goggin13/curails-mg343/master/spec/models/relationship_spec.rb
=> **spec/models/relationship_spec.rb**

## Following Users

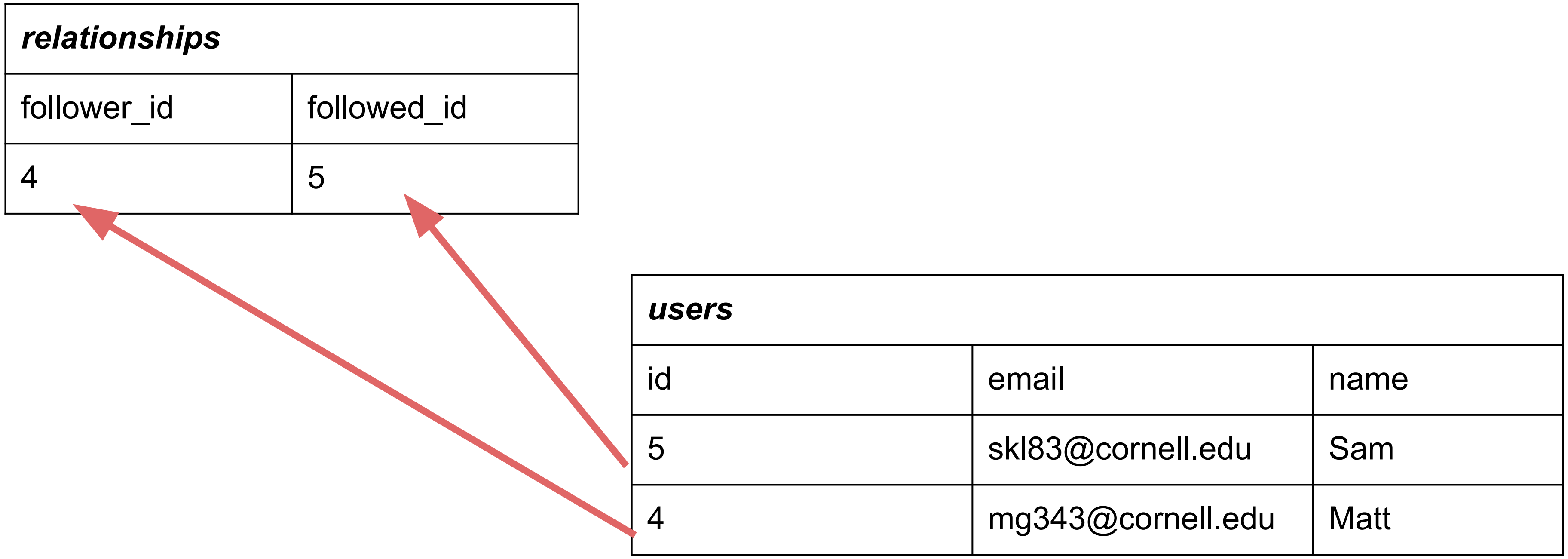
Time to add some social to our app; users should be able to follow/unfollow one another.

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

# ER Diagram



micro_posts
id
user_id
content
created_at
updated_at
users
id
name
email
hashed_password
salt
created_at
updated_at
relationships
id
followed_id
follower_id
created_at
updated_at



| ***relationships*** | |
|---|---|
| follower_id | followed_id |
| 4 | 5 |

| ***users*** | | |
|---|---|---|
| id | email | name |
| 5 | skl83@cornell.edu | Sam |
| 4 | mg343@cornell.edu | Matt |

"Matt is following Sam"

# Generate Model

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

=> type **n** when prompted if you wish to overwrite the relationship_spec file

**db/migrate/<timestamp>_create_relationships.rb**

```
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
```

# Uniqueness

Consider this simple User class, and two web requests wishing to create a new user with the email 'goggin13@gmail.com'

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

```
Web Request 1
                                                                          Web Request 2

user = User.new(email: "goggin13@gmail.com")

                                                  user = User.new(email: "goggin13@gmail.com")
user.save!
select count(*) from users where email='goggin13@gmail.com';
=> 0
                                                                                 user.save!
                                       select count(*) from users where email='goggin13@gmail.com';
                                                                                      0 <=
insert into users....

                                                                           insert into users....
```

## 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

  validates :followed_id, presence: true
  validates :follower_id, presence: true,
                                   uniqueness: { scope: :followed_id }

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

## belongs_to

Inside our MicroPost class, we see

**belongs_to :user**

This line works with no configuration, because the both the following are true:

1. The *micro_posts* table has a *user_id* field on it.
2. We want the *user_id* field to be used as a foreign key for a class named "User".

Now look at our Relationship model

We wanted to add follower, and followed methods.

1. There ARE fields on the *relationships* table named *follower_id*, and *followed_id*
2. But there is no "Follower" class or "Followed" class; instead we want Rails to use our "User" class.  So we just need to give it a little configuration

**belongs_to :follower, class_name: "User"**

**belongs_to :followed, class_name: "User"**

## Add relationships to our User model

```
class User < ActiveRecord::Base
  has_many :micro_posts

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

end
```

# has_many

In our User class we see

**has_many :micro_posts**

With no additional configuration, this works with the following assumptions

1. There is a table named *micro_posts,* and that table has a column named *user_id*.
2. There is a ruby class named **MicroPost**

We just added

**has_many :relationships, foreign_key: "follower_id"**

1. There is a table named *relationships,* but that table does not have a *user_id* field; so we override that option with the foreign key we wish to use.
2. There is a ruby class named **Relationship**

## 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
```

# has_many :through

We also added this line to our User class

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

We will want a method on User to retrieve all the users that they are following. This information is available ***through*** the relationships relation we defined already.  A user has_many relationships; and each of those relationships has a followed method on it.

Here we are passing this information to Rails to get our followed users method.

***User.rb***

has_many :**relationships**, foreign_key: "follower_id"

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

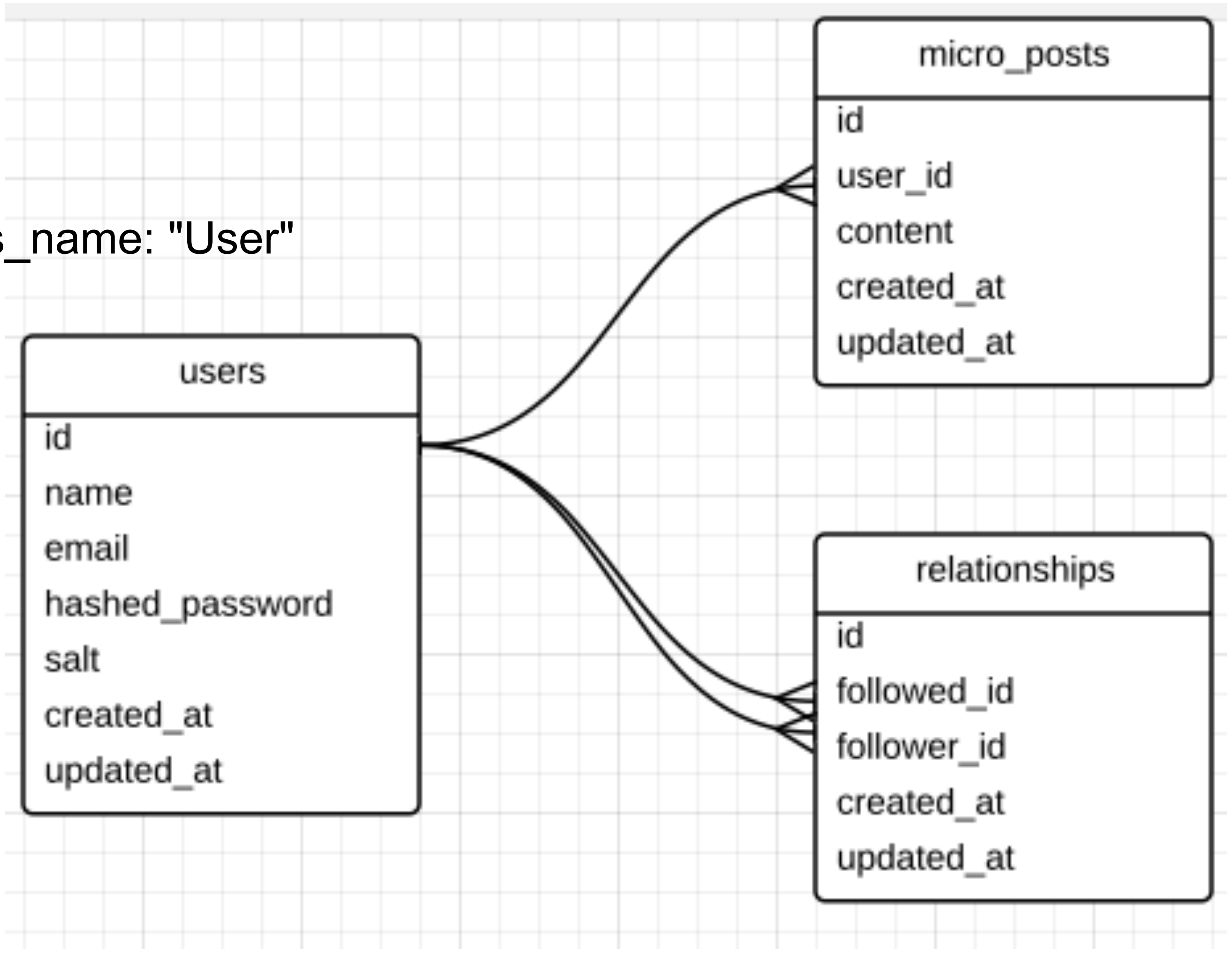
***Relationship.rb***

belongs_to :**followed**, class_name: "User"

**User**
has_many :micro_posts
has_many :relationships, foreign_key: "follower_id"
has_many :followed_users, through: :relationships, source: :followed

**MicroPost**
belongs_to :user

**Relationship**
belongs_to :followed, class_name: "User"



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

This clause simply tells Rails to destroy all of the child relationships when the parent is destroyed

So when we call

**user.destroy**

all of that user's relationships records will also be removed from the database.

## Phew

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

should be passing

# 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
```

Note that if you are writing code *inside of* the user class (as you will be in the next slide, you can omit *user* object from these examples; you could replace it with *self*, but that is implied so not necessary).

## 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/users/_follow_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/users/_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/users/_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
```

**app/controllers/relationships_controller.rb**

```
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
        # your code here; populate the @user variable and follow them
        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
        # your code here, populate the @user variable and unfollow them
        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 user

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

```
<% if signed_in? %>
  <%= render 'follow_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

## 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| %>
```

## Script partial for create

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

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

## Script partial for destroy

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

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

## Try it out

Now we should be able to follow/unfollow via AJAX

# ActiveRecord Querying

```
# A collection of all the users who have the username 'goggin13'
User.where('username = ?', 'goggin13')
-> select *
   from users
   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')
-> select *
   from users
   where email != NULL and id in (1,2,3,4)
   order by email desc;
```

# ActiveRecord Querying

```
MicroPost.where('content like ? and user_id in (?)', '%hello%', [1,2,3,4])
            .paginate(page: 2, per_page: 10)
-> select *
   from micro_posts
   where content like '%hello%'
            and user_id in (1,2,3,4)
   limit 10
   offset 10;
```

# 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
```

**Hints**

- Don't feel like you need to start from the current implementation; you may rather start from the previous ActiveRecord examples.
- It could help write out the raw SQL for what you want to accomplish first; then translate it into the ActiveRecord API calls
- This may be useful..
    - followed_user_ids = followed_users.map { |u| u.id }

      => an array containing all the ids of the users this user is following

**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
```

## Our JavaScript erb file

**app/views/micro_posts/refresh.js.erb**

```
var element = $("<%= j( render(@new_micro_posts) ) %>").hide();
$('#user_feed').prepend(element);
element.fadeIn();
```

## 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
```

* you can't copy paste this version with the comments inline; the lecture_8.txt file has a copy/pasteable version

## 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
```

## Maintaining Tests

Is work.

The first time we write tests, we inevitably make some assumptions about our application. E.G. that a "valid user" is comprised of just an email and a name (we later revised this assumption to include password).

Or we assumed that any user can create a MicroPost (later we required that the user be authenticated).

As we revise our applications, we may violate our prior assumptions, resulting in failing tests.

In the interest of time I have not been keeping us up to date with maintaining our full test suite; if you run your full test suite you will see failures have crept in. A good exercise for those of you who wish to start getting your hands dirtier would be to go through the failing specs and repair your test suite to all green.

## Today we...

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