**RoR Week 2 (6/17 & 6/19)**

**The Project App**

**------------------------------------------------------------TUESDAY 6/17-----------------------------------------------------------**

**Start a new project called ewb\_projects**

*>>rails new ewb\_projects*

**Generate a welcome controller**

*>>rails generate controller welcome index*

**Generate a products controller with the 7 RESTful functions & explore routing**

*>>rails generate controller products index create destroy edit update new show*

**Delete the products controller**

*>>rails destroy controller products index create destroy edit update new show*

**Create a “Project” model**

*>>rails generate model Project*

**Take a look at the migration file that was just created and add a title or type string and description of type text. OR you can use the migration command**

*>>rails generate migration AddTitleToProject title:string*

*>>rails generate migration AddDescriptionToProject description:text*

*The above two lines will generate two new migration files*

*Or you can modify the migration file directly with what is below and a new migration file won’t be created*

*t.string :title*

*t.text :description*

*\*\*Note: in order for your migration files to be executed and added to the database you must run your migration (the command is rake db:migrate)*

**Now that we created the model, we can play with it in console. This is how we can see what is in our database**

*>>rails console*

**Now you are in the ruby console inside of Rails. You can tell because of the irb>> prompt.**

**So now lets Create a few new projects and add them to our database.**

*Irb>>Project.all*

*Irb>>a = Project.new*

*Irb>>a.title = “Dominican Republic Sustainable Infrastructure”*

*Irb>>a.description = “The international project team is traveling to the DR this July to ….tbc”*

*Irb>>a.save*

*\*\*don’t forget to “a.save”! The object isn’t pushed to the database until you do so!*

**Exit the console**

*>>exit*

**Now we are back in the Rails console and we have the proejcts we just made in the database. Lets get them rendered on our website! This means we need to create controllers and views for what we want to display**

**Create the controller with no default actions (we will add the action and view associated with it manually this time so you know how. But know that rails will do all this for you if you say “rails g controller projects index”)**

*>>rails g controller projects*

**Modify your routes file such that the projects resource is there now**

*resources :projects*

**Add an index action to your controller such that you can access your model with a variable from your view**

*def index*

*@projects = Project.all*

*end*

**Now go to your view and print the project titles in your database**

*<p>This is a list of EWB Projects</p>*

*<ul>*

*<% @projects.each do |proj| %>*

*<li><%= link\_to proj.title, project\_path(proj)%></li>*

*<% end %>*

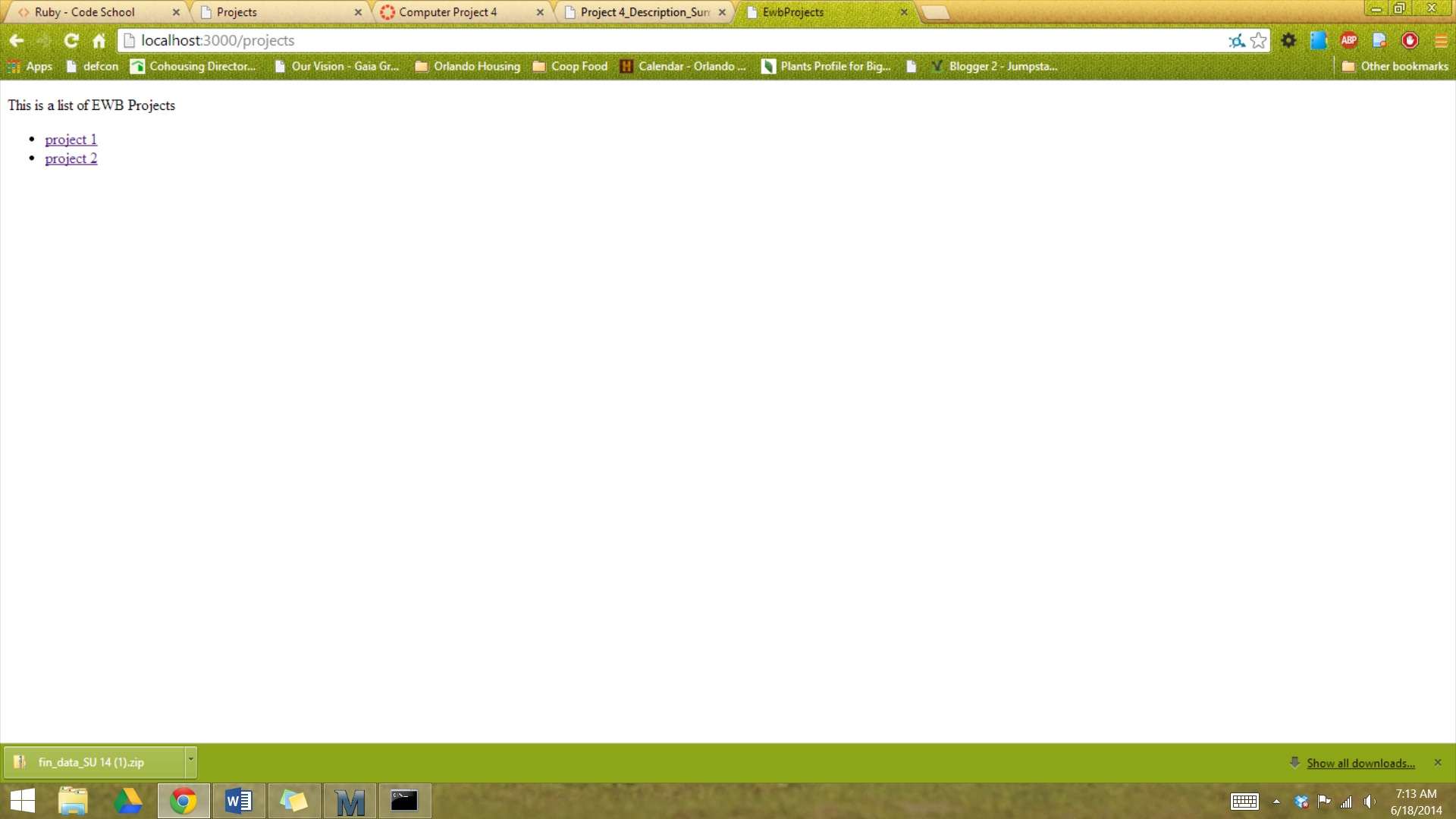
*</ul>*

*\*\*Whats going on here? @projects is the variable that is holding all the projects because in the controller we set it equal to “Project.all”. So just like we did in the ruby console we can access the projects. So we write a loop to loop through all the projects and create a link to the project with the project title as the clickable link.*

*“project” is the prefix that shows up when we rake routes and we append path to it and feed it the proj*

*“link\_to” is a rails helper function for linking to things. Look up its documentation.*

**Now start your server and you should see a list of projects on the index page of projects!**



**THIS IS WHERE WE STOPPED TUESDAY. ERASE YOUR PROJECT AND REDO IT! :D You need to be fast at procedurally creating everything above and understand what it all means.**

**You should now understand:**

* **How to generate and destroy controllers with empty actions**
* **What the 7 RESTful actions are**
* **What “resources :projects” does in our routes file**
* **How to create and destroy a model**
* **How to add columns to your database by generating a new migration or by editing the migration file directly**
* **How to write a loop in ruby**
* **How to access what is in your database through your view via @variable that you assigned in your controller**

**----------------------------------------------------------THURSDAY 6/19----------------------------------------------------------**

**Brief review. Answer questions. Recreate the project quickly.**

**Starting where we left off from the project above. Add a show action to your controller such that you can access your model with a variable from your view**

*def show*

*@project = Project.find(params[:id])*

*end*

*\*\*What is this?*

*@project is how we will access the project in our view*

*Project.find() is a function to get a specific project from our database*

*params[] returns a hash of the request parameters. Here we request the project ID which gives us access to the project with id = <whatever ID is passed from the view>*

**Now we should have access to that specific project in our show view. Go to your view and print the project titles in your database like below**

*<p>This page shows the project in more detail</p>*

*<p><%= @project.description %></p>*

*<%= link\_to ‘<<Back to all projects’, projects\_path%>*

Extra Challenges for practice if you want

1. Edit the routes file so that the root path points to your list of projects
2. Stylize your page

**We will now allow users to create, edit and destroy projects and later allow only privileged users to create, edit and destroy projects but all users to view them.**

**Following RESTful conventions we will write a form in the view associated with the projects#new action. This uses the form helper form\_for**

*<%= form\_for(@project) do |p| %>*

*<ul> <% @project.errors.full\_messages.each do |error| %>*

*<li><%= error %></li>*

*<% end %>*

*</ul>*

*<p>*

*<%= p.label :title %><br />*

*<%= p.text\_field :title %>*

*</p>*

*<p>*

*<%= p.label :description %><br />*

*<%= p.text\_area :description %>*

*</p>*

*<p>*

*<%= p.submit %>*

*</p>*

*<% end %>*

**Now we have to make this @project variable accessible to the show view via our controller just like we created a new project in the irb>> console**

*@project = Project.new*

**We also need the create action to process the “new” form. Check out the params available using “fail”**

*def create*

*fail*

*end*

**Now change the create function to save what the user entered**

*def create*

*@project = Project.new*

*@project.title = params[:project][:title]*

*@project.description = params[:project][:description]*

*@project.save*

*redirect\_to project\_path(@project)*

*end*

**Realistically we don’t want to blindly pass params like this into our database so we use what are called “Strong Parameters” via “require” and “permit”. Permit returns the hash while require returns the hash.**

**Your projects helper should look like this:**

*def project\_params*

*params.require(:project).permit(:title, :description)*

*end*

**Your projects controller should now look like this:**

*def create*

*@project = Project.new(project\_params)*

*@project.save*

*redirect\_to project\_path(@project)*

*end*

*PS—don’t forget to include the projects helper in your controller*

**Now we want people to be able to edit a project so we have to add an edit view.**

*<h1>Edit an Event</h1>*

*<%= render partial: 'form' %>*

**When the user submits the edit form it goes to an action in the controller called update. Then lets add to our update action in the controller. Add:**

*@project = Project.find(params[:id])*

*@project.update(project\_params)*

*redirect\_to project\_path(@project)*

**Now you should understand:**

* **How to create a form using rails form\_helper**
* **How to use strong parameters for security**
* **How to use “redirect\_to” in rails**
* **Be able to access any part of your models in your views.**

**REMOVE THE ENTIRE PROJECT AND DO IT AGAIN….@ least twice. You should be fast at doing these simple things so you can move to more complex things and not have to think about to hard about these things.**

**------------------------------------------------------------TUESDAY 6/24-----------------------------------------------------------**

**Last time we added index, show, new, create, edit, update.**

**Questions from last time:**

1. **When do you use single vs double quotes?**
2. **Do you need to create two objects when you pass new to create or edit to update?**
   1. **No.**
3. **What is update for? Why is it needed?**
   1. **Similar to New🡪Create, Edit🡪Update**
4. **How can you change “/project/1/bla” to “/project/myProjectName/bla”?**
   1. **Friendly ID**

**Review. Answer questions.**

**Today, we will add the destroy action and then get into using a gem called paperclip for adding an image to our project.**

**If we add this link to the show view, it will do to the destroy action in our controller.**

*>>* *<%=link\_to "Delete Project", project\_path(@project), method: :delete, data: {confirm: "Really delete the project?"}%></br>*

**Then we have to actually tell the controller destroy method how to delete it.**

*def destroy*

*@project = Project.find(params[:id])*

*@project.destroy*

*redirect\_to projects\_path*

*end*

**Test your modifications.**

**STOP: At this point all of your 7 Restful actions should be working.**

**🡪 What is RubyGems?**

**Just another package manager (like apt-get, yum, etc)**

**>>gem install <gemName>**

[**.gemspec files**](http://jeffkreeftmeijer.com/2010/be-awesome-write-your-gemspec-yourself/) **as YAML files which we’ve seen**

**“Yes Another Markup Language” … “YAML Aint Markup Language”**

**The gems we’ll utilize will be:**

[**paperclip**](https://rubygems.org/gems/paperclip)**—a simple file attachment library**

[**sorcery**](https://rubygems.org/gems/sorcery)**—a simple authentication library**

[**cancan**](https://rubygems.org/gems/cancan)**—adds authorization via “Ability class”**

[**devise**](https://rubygems.org/gems/devise)**—more sophisticated authentication (optional)**

[**active merchant**](https://rubygems.org/gems/activemerchant)**—for making payments, donations, etc**

**The Paperclip Gem**

**Add the ‘paperclip’ gem**

*>>gem install paperclip*

**Check and make sure its there**

*>>gem list paperclip*

**Now add it to your Gemfile and run bundler**

*#Paperclip Gem for file attachments*

*gem ‘ paperclip’*

*>>bundle install*

*>>bundle show paperclip*

**Now we have to add the proper fields in our database for the attachment**

*>>rails g migration add\_paperclip\_to\_project*

**Now edit the migration file you just created and run your migration**

*def change*

*add\_column :project, :image\_file\_name, :string*

*add\_column :project, :image\_content\_type, :string*

*add\_column :project, :image\_file\_size, :integer*

*add\_column :project, :image\_updated\_at, :datetime*

*end*

**\*Alternatively how could you have generated this migration shorthand?**

*>>rails g migration add\_paperclip\_to\_project image\_file\_name:string image\_content\_type:string etc etc*

**Now we have to tell our model it has an attachment**

*has\_attached\_file :image*

*validates\_attachment\_content\_type :image, :content\_type => [“image/jpeg”,”image/png”,”image/jpg”]*

*What is this?*

*<has\_attached\_file> is part of the paperclip library and looks for attachments starting with ‘image\_’*

*<validates\_content\_type> is required in 4.0 to validate the attachment*

**How do we have to edit our helper file? We have to whitelist the image to come through. Add the :image to your params helper file**

*params.require(:project).permit(:bla, :bla, :bla, :image)*

**Add the image tag to your show view also**

*<p><%= image\_tag @project.image.url %></p>*

**What if we only want to show the image if there is an image attached to the project?**

*<% if @project.image\_file\_name != nil %>*

*<p><%= image\_tag @project.image.url %></p>*

*<% end %>*

*OR we could use*

*if @project.image.exists?*

*.*

*.*

*.*

*end*

**Also add it to your form**

*<p>*

*<%= p.label :image, "Attach an image" %><br />*

*<%= p.file\_field :image %>*

*</p>*

**\*\*Challenge: play with paperclip to control the size of the image (look up documentation)**

**Now you should understand:**

* **What gems are and how to use them in your project**
* **How to use paperclip gem to add an attachment**
* **How to add validations for content type**
* **How to use conditionals in your view**

**------------------------------------------------------------------------------------------------------------------------------------------**

**The Sorcery Gem**

**Install it just like we did for paperclip and check to make sure it’s there.**

**Create**

*>>rails g sorcery:install*

**Add username column in the new migration file just generated.**

**Migrate your DB.**

**Rails generate a user’s scaffold:**

*>>rails g scaffold\_controller user username:string email:string password:password password\_confirmation:password*

**Add a validation to your user model**

*validates\_confirmation\_of :password, message: "passwords should match", if: :password*

**Add the users resource to your routes file and test the validation you just added to your model (ie. Enter different passwords)**

**Edit your layouts file to add flash and if logged in.**

*<h6>*

*<% if logged\_in? %>*

*<%= "Logged in as #{current\_user.email}" %>*

*<%= link\_to "(logout)", logout\_path %>*

*<% else %>*

*<%= link\_to "(login)", login\_path %>*

*<% end %>*

*</h6>*

**Now we need new, create and destroy to login.**

*>>rails g controller UserSessions*

*def new*

*end*

*def create*

*if login(params[:email], params[:password])*

*redirect\_back\_or\_to(projects\_path, message: 'Logged in successfully.')*

*else*

*flash.now.alert = "Login failed."*

*render action: :new*

*end*

*end*

*def destroy*

*logout*

*redirect\_to(:users, message: 'Logged out!')*

*end*

**Add this to your routes**

*resources :user\_sessions, only: [ :new, :create, :destroy ]*

*get 'login' => 'user\_sessions#new'*

*get 'logout' => 'user\_sessions#destroy'*

*\*\*Why do we create a separate sessions controller for our users with new, create, destroy actions?*

*\*\*So when do you create a new controller in general? When you think about putting non RESTful actions in an existing one, when you need a separate resource*

**Now we want to be able to restrict certain users to have access to certain pages right? Well, Rails gives us this handy thing called ‘before\_filter’. We can now ‘filter’ our users very easily.**

**For example, at the top of our projects controller if we only want the logged in user to be able to edit, destroy etc, then we can require login for all pages except index and show.**

*before\_filter :require\_login, except: [:index, :show]*

**Alternatively we can use “only” to show only edit, destroy, etc for users that are logged in.**

*before\_filter :require\_login, only: [:edit, :update, :destroy, :new, :create]*

*\*\*Try adding a custom filter with ‘except’ or ‘only’ keywords.*

*\*\*Test it (ie. Try accessing these actions when you are/aren’t logged in).*

**----------------------------------------------------------THURSDAY 6/26---------------------------------------------------------**

**Finish up with paperclip and sorcery.**

**Design a project from scratch—moving on Tuesday.**

**Project Ideas:**

**🡪create blog page**

**🡪articles**

**🡪personal site**

**🡪Recreate a basic project site from scratch then add events and a photo gallery.**

**------------------------------------------------------------TUESDAY 7/1-----------------------------------------------------------**

**Last time:**

* **Got all 7 RESTful actions working with two gems (paperclip and sorcery)**
* **Questions:**
* **Today:**
  + **Advanced routing and scope**
  + **CanCan Gem**
  + **Mailers**

**Types of Associations**

* belongs\_to
* has\_one
* has\_many
* has\_many :through
* has\_one :through
* has\_and\_belongs\_to\_many

**CanCan Gem**

**Install the gem**

**Run the CanCan generator**

*rails g cancan:ability*

**Add an admin role to your user model as a string or boolean**

**Edit your views with “can ?” and “cannot ?”**

**Twitter Bootstrap Gem & Scaffolding**

**Generate the scaffold and LOOK at what it generated.**

*>>rails g scaffold Blog title:string content:text*

**Migrate your DB**

**LINUX USERS ONLY**

**Add ‘twitter-bootstrap-rails’ gem**

*>>rails g bootstrap:install static*

*>>rails g bootstrap:layout application fixed –f*

**Note: you can use a fluid layout too….look up how it looks ;)**

*>>rails g bootstrap:themed Blog –f*

**Look at your theme**

**OTHER USERS**

**…have to user sass bootstrap**

*https://www.youtube.com/watch?v=oq-HNuxFPdE*

**Mailers (confirmation emails, etc)**

**Generate the mailer**

*>>rails g mailer welcome\_mailer*

**Set up your initializers file (or your development/production/text environment)**

*>>rails g mailer welcome\_mailer*

**Set up your mailer controller**

*def registration\_confirmation(user)*

*mail :to => user, :from => “email@domain.com”, :subject => “Subject”*

*end*

**Create your view that is the email**

*WelcomeMailer.registration\_confirmation(@user).deliver*

**TROUBLESHOOTING**

**🡪check your log files**

**-----------------------------------------------------------THURSDAY 7/3-----------------------------------------------------------**

**NO CLASS!!!!! COME READY ON TUESDAY…LOTS TO COVER.**

**------------------------------------------------------------TUESDAY 7/8-----------------------------------------------------------**

**Last time:**

* **Associations, mailers, advanced routing and scope**
* **Questions?**

**Today:**

* **PayPal + Security**
* **Active merchant**

**\*\*No class Thursday—McK and Josh are out of town**

**PayPal + Security**

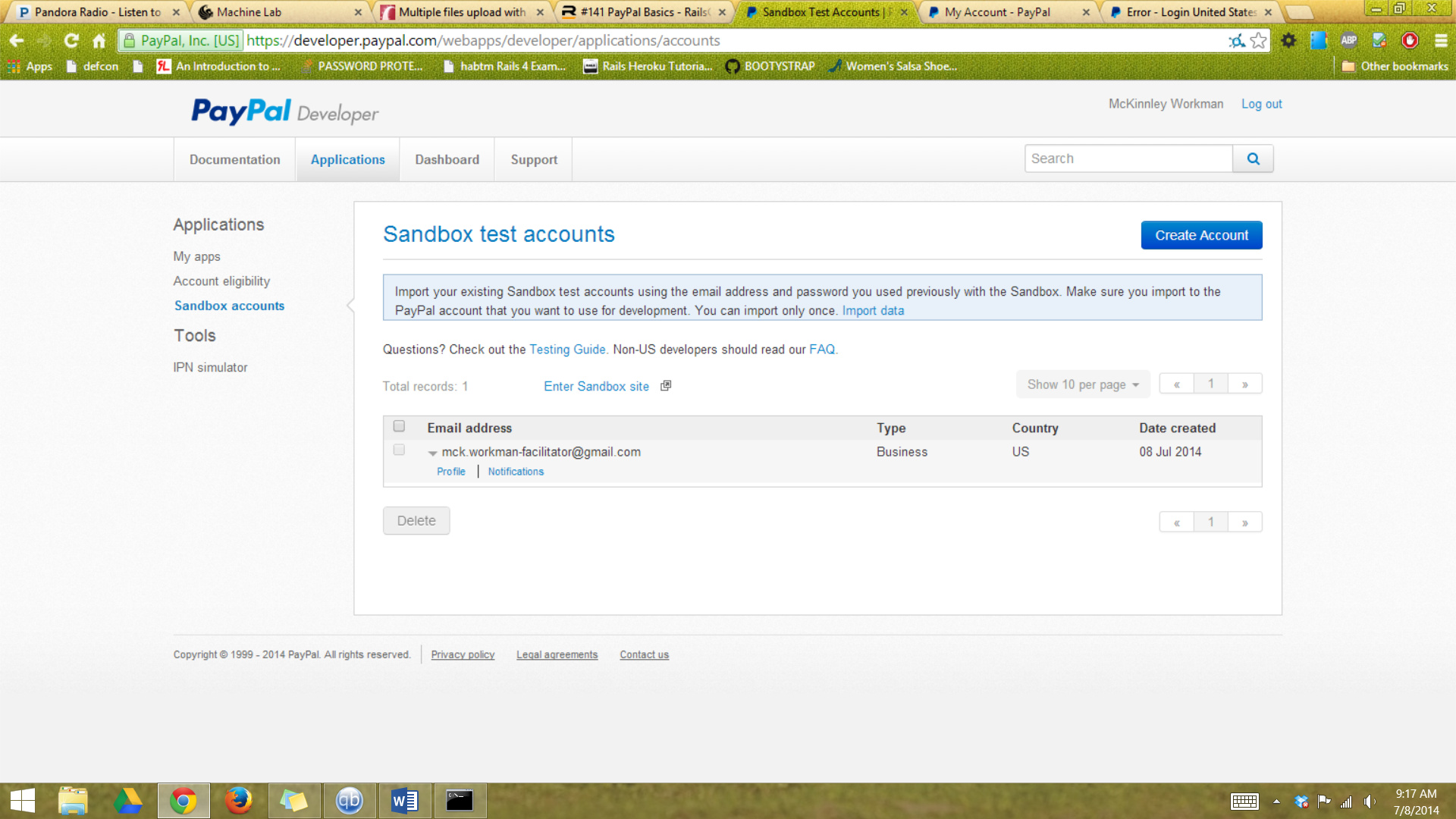
**Secure Payments with Active Merchant (one time payment processing, recurring payments)**

TWO WAYS TO DO IT:

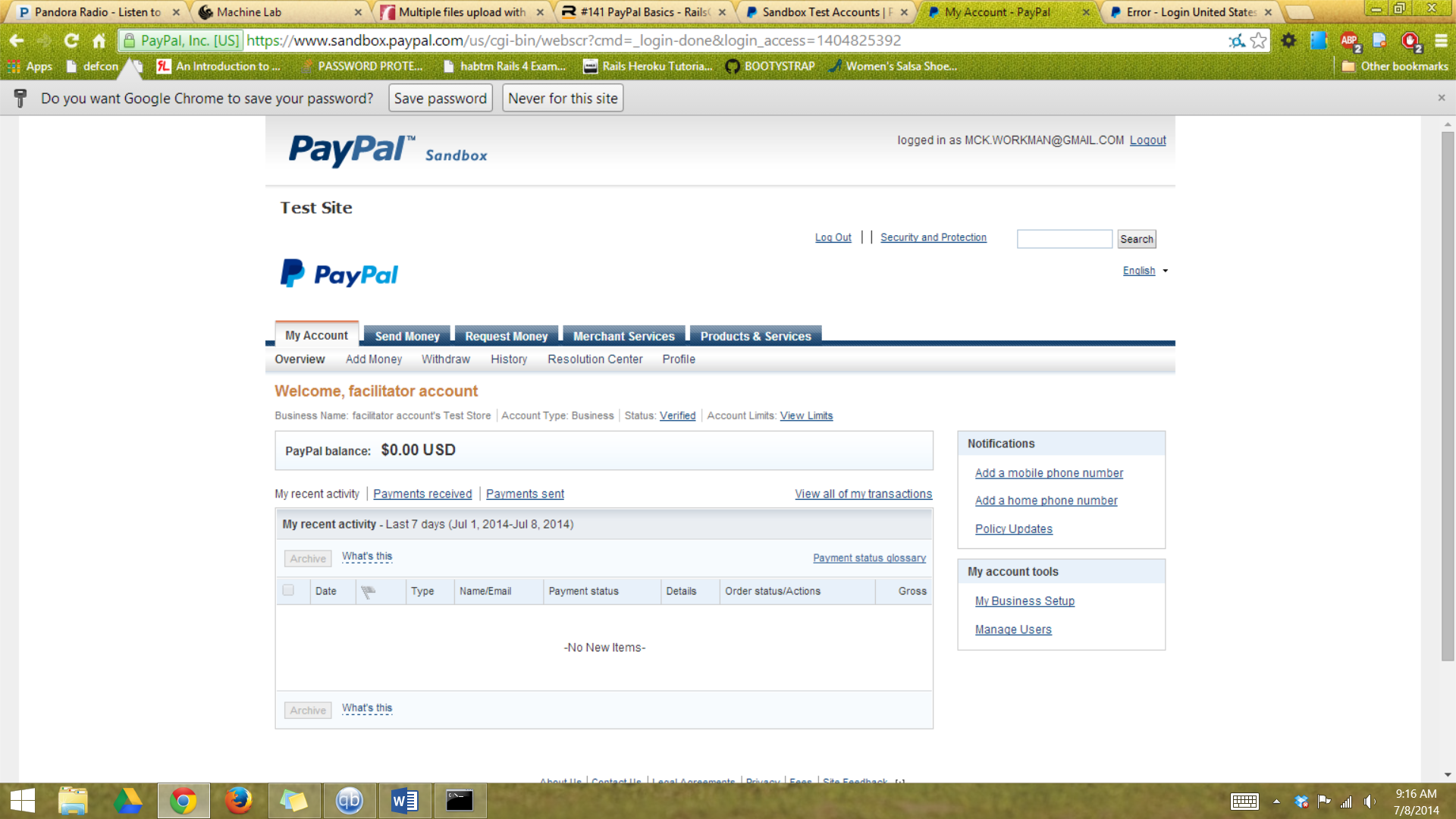
link to paypals site for checkout or integrate paypals backend for your site

**Get Set up:**

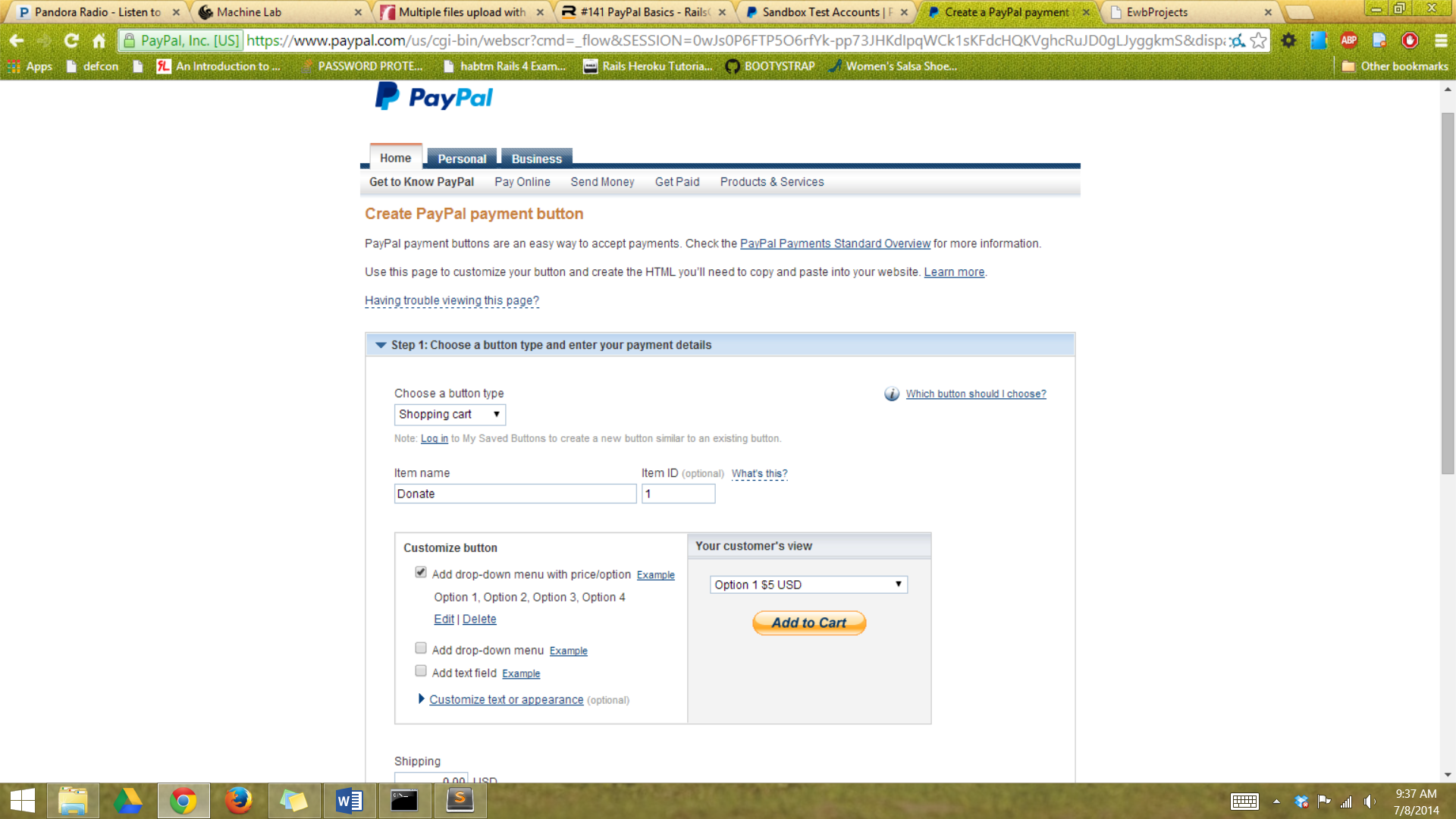
1. **Make an account: developer.paypal.com**
2. **Create a seller account (Applications**🡪 **Sandbox accounts)**



1. **Enter sandbox site and sign in with your credentials**



1. **Create a buyer account also and give yourself come money**
2. **Click on “Merchant Services”**
3. **Now we can choose how we want to integrate PayPal. First we will simply pass a URL to paypal with all of the order information and the payment will be processed on paypal’s site.**
4. **Now just add a donate button link to your view**
5. **Create PayPal payment button**



1. **Test it.**
2. **What if we want to pass values in for different items?**
   1. **Make a form with variable names as form input and append those the the paypal url that you pass**

THIS IS IN YOUR VIEWS FILE

<%= link\_to "Checkout", @cart.paypal\_url(products\_url) %>

THIS IS IN YOUR MODELS FILE

def paypal\_url(return\_url)

values = {

:business => 'seller\_1229899173\_biz@railscasts.com',

:cmd => '\_cart',

:upload => 1,

:return => return\_url,

:invoice => id

}

line\_items.each\_with\_index do |item, index|

values.merge!({

"amount\_#{index+1}" => item.unit\_price,

"item\_name\_#{index+1}" => item.product.name,

"item\_number\_#{index+1}" => item.id,

"quantity\_#{index+1}" => item.quantity

})

end

"https://www.sandbox.paypal.com/cgi-bin/webscr?" + values.to\_query

end

1. **But what about security?** 
   1. **Problems: people can change the values manually very easily**
      1. **Try viewing source of your code**
   2. **Solution: encryption**
   3. **How does it work**

**Public Key Encryption or Asymmetric Cryptography**

*Public certificate*: *public key* + identification + expiration date

You and paypal share your public certificates and generate a shared secret key

*Private keys*: generated by the receiver and kept on your system privately

Encryption to send a message: my private key + receivers public key => send encrypted message

Decryption to receive a message: receivers private key + senders public key => get decrypted message

**How? (different ways to do this for windows vs unix systems)—will post if you can’t figure it out ☺**

1. **NOW…..If you want to not use PayPals front end you can use your sites front end so you can display pictures or whatnot you can keep your front end and use PayPals backend. Active Merchant gem makes it super easy for us.**
2. **Install the gem**
3. **Edit active merchant configuration settings**
4. **Edit your controller, environments and models to**
5. **MAKE SURE YOU DON’T FORGET: Block all non-encrypted requests in your PayPal settings and use a shared secret in your create method**

**Aside:**

**How do we paypal to tell us when the payment was processed successfully so we can thank the user? We won’t do this now because you need to have a live server running but I will talk about it.**

**You would need to create a new resource and model for payment notifications**

*>>rails g scaffold payment\_notification params:text status:string transaction\_id:string create*

*>>rake db:migrate*

**And then allow (in your controller) paypal to access the create action without having to pass in the authenticity token which is enabled by default. Something like this:**

*protect\_from\_forgery :except => [:create]*

*def create*

*PaymentNotification.create!(:params => params, :status => params[:payment\_status], :transaction\_id => params[:txn\_id])*

*render :nothing => true*

*end*

**But now it makes sense to modify our model such that it belongs\_to a donations model.**

So our model would look something like this now:

belongs\_to :donations

serialize :params

after\_create :donated

private

if status == “Completed”

project.update\_attribute(:donated\_at, Time.now)

end

end

**-----------------------------------------------------------THURSDAY 7/10---------------------------------------------------------**

**NO CLASS!!!! McKinnley and Josh are gone**

**Photogallery Tutorial (follow step by step and come Tues with questions)**

**We need to create an album model that has\_many images and tell our image model that it belongs\_to an album.**

*>>rails g scaffold album title:string description:text*

*In “models/album.rb”*

*has\_many images*

*In “models/image.rb”*

*belongs\_to album*

**Then we need to add paperclip fields to our image model just like we did before (see “Paperclip Gem” section above).**

**Then we need to add a table in our DB to associate these two and allow the \_id params through in our helper files or private classes (wherever the params are being whitelisted)**

*rails g migration create\_album\_image image\_id:integer album\_id:integer*

*Then permit the params in each controller for the other items \_id:*

*image\_ids => []*

**Now we can add/create/edit/destroy an album but we also want to be able to add/create/edit/destroy the images that are in that album so modify your views so the user can do this.**

*In “controllers/albums\_controller/index”—here I want to show thumbnails of the albums with the first picture in the album (if it exists) being on the thumbnail)*

*In “controllers/albums\_controller/show”—here I want to show thumbnails of all images that belong to the album. Go to irb console and figure out how you access the images associated with albums.*

*Hint: albumObject.image*

**Add whatever theme/style you want:** [**http://startbootstrap.com/thumbnail-gallery**](http://startbootstrap.com/thumbnail-gallery)

**In addition maybe we want the user to be able to upload multiple files at once, or drag and drop a folder or do something more high tech with uploading images.**

[**https://stackoverflow.com/questions/15703594/rails-paperclip-drag-and-drop-multiple-files**](https://stackoverflow.com/questions/15703594/rails-paperclip-drag-and-drop-multiple-files)

[**http://dannemanne.com/posts/drag-n-drop\_upload\_that\_works\_with\_ror\_and\_paperclip**](http://dannemanne.com/posts/drag-n-drop_upload_that_works_with_ror_and_paperclip)

**------------------------------------------------------------TUESDAY 7/15-----------------------------------------------------------**

**Transitioning to Other Environments**

**Changing environments**

**Config/environments has development, test and production (you can also add more).**

**--hiding passwords in environment variables**

**--change the log levels**

**--note: you can migrate to a specific environment with RAILS\_ENV**

**Then start the server in a specific environment:**

*rails s –e test*

**--running tests in rails with the test environment**

**See my “environments” code to see the configurations.**

**Amazon EC2 or Heroku**

Use Heroku if you have something small and simple that you want to run on a small [Dyno](https://devcenter.heroku.com/articles/dynos). Otherwise EC2 is highly customizable and a great option for larger applications.

Heroku has great [documentation](https://devcenter.heroku.com/articles/getting-started-with-rails4) for getting up and running in RoR 4.

EC2 has great documentation also but requires a bit more set up. There are several good guides. User them or come into office hours if you can’t figure it out.

**-----------------------------------------------------------THURSDAY 7/17---------------------------------------------------------**

**Special Topics coming soon on the blog:**

**--“Remember me” check box for login**

**--Writing tests for testing as you go (rspec)**

**--Processing payments through your site with PayPal backend**

**----------------------------------------------------SECURITY WEEK 7/22-24-----------------------------------------------------**

**Security Week—Brute forcing WEP demo, using NMAP, modern “hacking”**