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 Article 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>
```

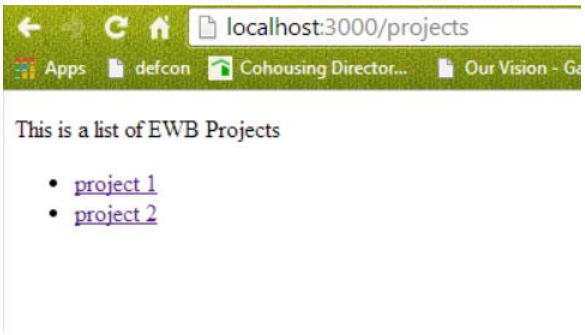
***Whats going on here? @projects is the variable that is holding all the projects because in the controller we set it equal to "Project.all". So just like we did in the ruby console we can access the*

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.

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
```

Now go to your view and print the article titles in your database

* This page shows the project in more detail *

* <%= @project.body %> *

<%= link_to "<<Back to all proejcts", projects_path%>

STOP: Now you should understand…

Basic routing with resources, RESTful routing, how to link to routes, how to pass params through your controllers.

Extra Challenges for practice if you want

1) Edit the routes file so that the root path points to your list of articles
2) Add an image to your database that shows up when you display the project description
3) Stylize your page

Delete the project and do it again

rm –r ewb_projects

----------PHASE II----------

Picking up where you left of from Phase I, we will now allow users to create, edit and destroy projects and later allow only privileged users to create, edit and destroy articles.

Following RESTful conventions we will write a form in the view associated with the projects#new action. This uses the form helper form_for,.submit, etc

```
<%= 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 view via our controller

Add @article = Article.new to the appropriate event in your controller

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

```
def create
    fail
end
```

Now change it 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

What if we want to delete a project? Add this link to the show view

>> <%=link_to "Delete Project", project_path(@project), method: :delete, data: {confirm: "Really delete the project?"}%></br>

But now we have to actually delete it via the controller

def destroy

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

@project.destroy

redirect_to projects_path

end

Test your modification by creating a new article. It should work just the same

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)

We can add a flash notice if we want.

>>you figure it out

Now replace the form we made in the "new" and "update" view with the line below and put the form in a partial called '_form.html.erb'

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

****Challenge: add all your flash notices to your layouts file**

****Challenge: add images to your articles with paperclip gem**

>>you figure it out or ask me next class

STOP: YOU SHOULD NOW ALL UNDERSTAND

1) **How what the create, new, edit, destroy RESTful actions do and how to implement them**
2) **Know how to build a form using the "form_for" helper**
3) **Use flash notices**
4) **Use the link_to, redirect commands**
5) **Be able to rout using prefixes from rake routes**
6) **Be able to add and manipulate objects in the irb console**
7) **Know what form helpers are for and be able to use them in your views**
8) **Understand and be able to add to and edit migrations**