Scaffolding

One of Rails great features is the ability to get a project up and going quickly and one of the fea
There are a few things you will want to do with most applications: CRUD (create, read, update a
pieces you need to do this. Generating a new scaffold:

```
rails g scaffold User first_name:string last_name:string email:string age:integer
```

We generate scaffolds just like we would a model because scaffold is going to create a model, a
when we create a normal model we still need to rake *db:migrate* after, though. There isn't anyth
take a look at the controller it generates piece by piece.

```
class UsersController < ApplicationController
  before_action :set_user, only: [:show, :edit, :update, :destroy]
```

First is this *before_action* which will run the method *set_user* before the *show, edit, update* or *c*

```
def set_user
  @user = User.find(params[:id])
end
```

The *set_user* method sets the user based upon the **id** in the *params,* we will go over how it gets
the private section of methods meaning it can only be called by other methods and not by the u

```
def index
  @users = User.all
end
```

The *index* method is used to show all of the objects, it then loads the **index.html.erb** which ther
users. Part of the **read** of CRUD.

```
def show
end
```

This method loads the **show.html.erb** and is used to display information about a specific user. Y
code is to bring up the users information but remember *before_action* set the user for us alreac
CRUD.

```
def new
  @user = User.new
end
def create
  @user = User.new(user_params)
  respond_to do |format|
    if @user.save
      format.html { redirect_to @user, notice: 'User was successfully created.' }
      format.json { render action: 'show', status: :created, location: @user }
    else
      format.html { render action: 'new' }
      format.json { render json: @user.errors, status: :unprocessable_entity }
    end
  end
end
```

These two functions together make up the **create** aspect of CRUD. The new method creates a n
new.html.erb where there is a form to create new users. When we plan on adding a new object
important that we pass them new versions of the model. The second part of this is the create m
to create a new user it gets sent to the create method (we will discuss how in a bit). The first lin

```
@user = User.new(user_params)
```

The user_params is not a different kind of params it is the last method on our page.

```
def user_params
  params.require(:user).permit(:first_name, :last_name, :email, :age)
end
```

This method requires that the params has a *:user_object and it will permit the fields first_name,
those items are permitted it returns the params with only the objects we required/permitted, th
field in your database that allowed someone to be an admin. A malicious user could alter the H
_user[admin] = true* and if we just used params instead of verifying the params with *user_param*
an admin or possibly worse! **We will always use a private function to determine which paramet
model.** After we have made a new user based on the parameters we permitted we have:

```
respond_to do |format|
  if @user.save
    format.html { redirect_to @user, notice: 'User was successfully created.' }
    format.json { render action: 'show', status: :created, location: @user }
  else
    format.html { render action: 'new' }
    format.json { render json: @user.errors, status: :unprocessable_entity }
  end
end
```

Now, this may look intimidating at first but there are simply two different things happening here
format that the request was in, was this an AJAX or just a regular HTTP request? If it was AJAX
otherwise the *format.html* will. This allows us to serve users who are not using JavaScript and ca
normally worry about this, just assume that they will be using JavaScript. Next we have the if sta
@user. If it is successful we redirect to *@user*(this will be covered shortly) with a notice that we
will return a false and send us to the *else* where we render the action *new*. Now in the scaffolde
check to see if there are any errors attached to the *@user* and display them there. The only reas
the user submitted doesn't pass your validations.

The next two functions represent our **update** in CRUD:

```
def edit
end
def update
  respond_to do |format|
    if @user.update(user_params)
      format.html { redirect_to @user, notice: 'User was successfully updated.' }
      format.json { head :no_content }
    else
      format.html { render action: 'edit' }
      format.json { render json: @user.errors, status: :unprocessable_entity }
    end
  end
end
```

Much like new/create the edit method loads the edit.html.erb with a form to edit the user in it. F
edit by *set_user*. When the form is submitted it will go to the update method and it will try to u
user_params, remember anything from the user could be tampered with, and will respond by ei
and loading the edit again to show the errors.

Our last method will be our **delete** in CRUD:

```
def destroy
  @user.destroy
  respond_to do |format|
    format.html { redirect_to users_url }
    format.json { head :no_content }
  end
end
```

Destroy simply deletes the user from the database.

You may have wondered through out how all these *redirect_to @user* and using *@user* for form
check out our *routes.rb*, we should see:

```
resources :users
```

This little bit of code is providing 7 routes! (actually 8 but PATCH/PUT are the same)

HTTP Verb	**Path**	**Action**	**Used for**
GET	/users	index	display a list of all users
GET	/users/new	new	return an HTML form for creating a new user
POST	/users	create	create a new user
GET	/users/:id	show	display a specific user
GET	/users/:id/edit	edit	return an HTML form for editing a user
PATCH/PUT	/users/:id	update	update a specific user
DELETE	/users/:id	destroy	delete a specific user

Report a mistake

With these seven routes you should be able to do most of what you want to users.

In Rails we can use objects to define access routes. On our **new.html.erb** we see this in the opening form

```
<%= form_for(@user) do |f| %>
```

Now, depending what is in *@user* the form will either go to create or update. Since this was loac
assume *@user* is a brand new *User* object without any information, meaning it doesn't have an i
the *new_user_path*. Now if we had provided an actual *@user* it would have an id and rails woulc
and the method to be PATCH, meaning when we submit the edit form it will go to the update m

IMPORTANT:

There is no HTTP verb for PATCH/PUT/DELETE in the actual browser, so you want a link or a fo
have to tell it that the method is " *delete*" or *"patch"*. But when we use *form_for* or *simple_form*
handled for us most of the time, unless we want something more custom.

```
#For a link
<%= link_to 'Log out', session_path(current_user), method: "delete" %>
#For a form tag if you want to teach the default method
form_tag(search_path, method: "patch")
#What the HTML will look like
```

Summary

We understand that there are lots of things going on with scaffolding. This can be intimidating i
wondering whether you should even use scaffolding to build your project. There is a big debate
used or not. We think it's beneficial to 1) serve as a guideline when you're learning Rails and 2) v
prototype. Most complex applications you build later will probably not rely too much on scaffol
the first time, understand what scaffolding is and you can print the PDF handout and have it as
assignments.   Download the handout
(http://s3.amazonaws.com/General_V88/boomyeah/company_209/chapter_2328/handouts/ch
guidelines.pdf).

Privacy Policy   To repor

‹ PREVIOUS (/M/9/4302/31443)