Photo Gallery

INFO 2310:
Topics in Web Design and Programming

Want an internship this summer? Interested in an insider's look into these companies?

interaction design lab

imagination at work

YAHOO! flickr

Google

Microsoft

ISSA General Meeting
Student Internship Panel
Monday, 11/3 @ 4:30 PM, Upson 205

infoscience STUDENT ASSOCIATION

All together now…

We now know pretty much everything we need to know to put together a pretty good website quickly.

We'll demonstrate by putting together a photo gallery during this lecture.

Create the gallery

```
rails –d sqlite3 gallery
```

(or in Komodo: New Project from Template, choosing Ruby on Rails, naming the project whatever you want, and using the sqlite3 db).

Create some models

```
ruby script/generate scaffold
  photo title:string
  description:text
  taken_on:date
  image_file_name:string
  image_content_type:string
  image_file_size:integer
```

or whatever attributes you want.

Albums

```
ruby script/generate scaffold
  album title:string
```

or whatever fields you think you want.

Albumizations

These will connect photos and albums.
`ruby script/generate model albumization`
To the migration add

```
class CreateAlbumizations < ActiveRecord::Migration
    def self.up
      create_table :albumizations do |t|
        t.integer  :album_id, :photo_id, :position

        t.timestamps
      end
    end
```

Create the DBs

Go ahead and run `rake db:migrate.`

Making associations

Now we should add to the models the associations we intend.
To the album model, we note that it has many photos (and that the specification of the photos happens through the albumizations model).

```
has_many :albumizations, :dependent =>
  :destroy, :order => "position"
has_many :photos, :through =>
  :albumizations, :uniq => true, :order
  => "position"
```

More associations

To the photo model, we note that it "has many" albums (since a photo can be in multiple albums).

```
has_many :albumizations,
  :dependent => :destroy
has_many :albums, :through =>
  :albumizations
```

More associations

Last, albumizations belong to both photos and albums via its foreign keys.

```
belongs_to :photo
belongs_to :album
```

Update the routes

To `config/routes.rb,` add the routes (to the top of the file)

```
map.resources :photos
map.resources :albums,
 :has_many => :photos
```

Paperclip

We're going to use a gem called paperclip that will handle the images for us. This uses the underlying Imagemagick program that we've already installed.

In `config/environment.rb`, add (in the appropriate place)

```
config.gem "paperclip"
```

Paperclip

Go ahead and `rake gems:install`. If you're using a lab machine with a flash drive, you probably want to then `rake gems:unpack`.

Adding images to photos

Now we can tell the photo model about images. Add this to the photo model:

```
has_attached_file :image,
  :styles => {
      :square => "75x75#",
      :thumb => "100x100>",
      :small => "240x240>",
      :medium => "500x500>",
      :large => "1024x1024>" }
```

This will let us display images in these various sizes, with Imagemagick taking care of resizing and cropping as needed (the '#' and '>' commands specify resizing and cropping).

Selecting a file

To add a form input for the image in a view, use

```
<%= f.file_field :image %>
```

(e.g. probably want to add this to views/photos/new.html.erb). We also need to say that the form is multipart, e.g.

```
<% form_for(@photos), :html =>
  {:multipart => true} do |f| %>
```

Showing an image

Now in a view that you want to display a photo (e.g. views/photos/show.html.erb)

```
<%= image_tag
  @photo.image.url(:medium) %>
```

/photos/1 vs. /albums/1/photos/1

The way we have things set up, both /photos/1 and /albums/1/photos/1 will show photo 1, even if it isn't in album 1. Let's fix this in photos_controller.rb.

photos_controller.rb

Change show action from:

```
@photo = Photo.find(params[:id])
```

to

```
@album = Album.find(params[:album_id]) if
   params[:album_id]
@photo = @album ? @album.photos.find(params[:id])
   : Photo.find(params[:id], :include => :albums)
```

Then we get an error if the photo isn't in the album.

More controller editing

We'd like /albums/:id to show all photos in the album, not /albums/:id/photos, so let's redirect the latter to the former.

In photos_controller.rb, we add the following at the start of the index action:

```
redirect_to :controller =>
  'albums', :action => 'show', :id
  => params[:album_id] if
  params[:album_id]
```

Some thoughts on views

One idea on how to decide which albums hold which photos: let's do a checklist of albums when we add/edit a photo.

We'll do this via a partial that we'll request to be displayed from within app/views/photos/new.html.erb and app/views/photos/edit.html.erb.

Rendering the partial

We can first add the request to render the partial (at the appropriate place in the form):

```
<%= render :partial => 'albums',
  :locals => {:f => f} %>
```

Note that we pass in the form object f as a local variable.

The partial

Then add the partial views/photos/_albums.html.erb:

```
<h4>Albums</h4>
<ul>
<% @albums.each do |a| %>
<li><%= check_box_tag 'photo[album_ids][]', a.id,
   @photo.albums.include?(a) %>
      <%= hidden_field_tag 'photo[album_ids][]', ''
   %>
      <%= f.label :id, a.title %>
</li>
<% end -%></ul>
```

Making albumizations work

To get this to work, we need to have the albumization (:photo_id, :album_id) saved *after* we save the photo (since a photo_id won't exist until after the save).

To do this, we play some tricks with the album_ids getters/setters in the Photos model.

Album_ids getters/setters

In app/models/photo.rb we add

```
after_save :update_albums

def album_ids=(ids)
  if new_record?
    @new_album_ids = ids.reject(&:blank?)
  else
    self.albumizations.each { |z| z.destroy unless
    ids.delete(z.album_id.to_s) }
    ids.reject(&:blank?).each {|id| self.albumizations.create(:album_id =>
    id) }
  end
end

def album_ids
  new_record? ? @new_album_ids : albums.map(&:id)
end

private
  def update_albums
    self.album_ids = @new_album_ids if @new_album_ids
  end
```

This calls a method we need to add to the album model.

```
def to_s
  self.title
end
```

Orphans

What should we do about photos that aren't in any album?

One possibility: we can set up a separate route (/photos/orphaned) to display them.

In config/routes.rb, change the map.resources for :photos to

```
map.resources :photos, :collection
  => {'orphaned' => :get}
```

Orphans

Now we add an action to the photos_controller to handle this.

```
# GET /photos/orphaned
def orphaned
  @photos = Photo.orphaned
end
```

Orphans

We add some code to the Photo model to return all the orphaned photos.

```
def self.orphaned
  find(:all, :include =>
  :albumizations).select {|p|
  p.albumizations.empty? }
end
```

Finally, we need to add a view in app/views/photos/orphaned.html.erb. It should display all the photos from @photos.

Secondary content

Recall that we can create an application-wide layout in views/layouts/application.html.erb (as long as the other *modelname*.html.erb files aren't there).

One thing we might want to do is to have a div for secondary content (for some sort of secondary navigation).

In the layout

We can do this in application.html.erb by saying

```
<div id="primary-content">
  <%= yield  %>
</div>

<div id="secondary-content">
  <%= yield(:secondary) %>
</div>
```

content_for

Now in the view we can populate the content of the div using `content_for`. E.g. in app/views/photo/index.html.erb, we add

```
<% content_for :secondary do %>
<h3>Albums</h3>
<ul id="albums_list">
  <% @albums.each do |album| %>
  <li>
    <%= link_to album.title, album %><br />
    <%= link_to
   image_tag(album.photos.last.image.url(:square)), album if
   album.photos.last %>
  </li>
<% end -%>
</ul>
<% end -%>
```

Then add to index, new, edit actions (in photos_controller.rb) something to pass in @albums array:

```
@albums = Album.find(:all)
```

Next and previous

We can define next and previous messages on a photo, so that if viewing a photo inside an album, we can get the next photo in that album.

Next and previous

In our Photo model (app/models/photo), we define

```
def next(album=nil)
  collection = album ? album.photos :
   self.class.find(:all)
  collection.size == collection.index(self) + 1 ?
   collection.first : collection[ collection.index(self)
   + 1 ]
end

def previous(album=nil)
  collection = album ? album.photos :
   self.class.find(:all)
  collection[ collection.index(self) - 1 ]
end
```

To do

There's still plenty to do to get this to work nicely.

- Add the orphaned view (app/views/photos/orphaned.html.erb)
- Fix the album show view (app/views/albums/show.html.erb) to show all the photos in the album (in the correct position!)
- Add 'next' and 'previous' to work for displaying photos in an album (or just looking at all photos)
- Make it look nice with some CSS…

Other things

- Play with JavaScript/Ajax effects to allow for drag-n-drop sorting of photos in an album.

Want an internship this summer? Interested in an insider's look into these companies?

interaction design lab

imagination at work

YAHOO! flickr

Google

Microsoft

ISSA General Meeting
Student Internship Panel
Monday, 11/3 @ 4:30 PM, Upson 205

infoscience STUDENT ASSOCIATION