# Ruby on Rails 101 – Chapter 6

In this chapter, we will add user model to the photo gallery.



# Managing asset files

The entry point of the assets:

1. The application.js
2. The application.css

It is an entry point because in each file it includes the other files into the assets.

```
require name     <-- means include that file.
require_tree .   <-- means include all the files in
the current directory.
```

There is a list of folders the asset pipeline will search.

```
/app/assets/
/lib/assets/
/vendor/assets/
```

Actually we can use the file others than the “application” name. Just clone the `application.css` and .js into a new file and update the asset include name in the `views/layout/application.html.erb` file.

How about we want to include specific controller assets only?

1. Remove the `require_tree .` in the application js and css file.
2. Use `<%= javascript_include_tag params[:controller] %>` when needed
3. or `<%= stylesheet_link_tag params[:controller] %>` when needed

# Creating gallery home page

Let's start styling the home page.

First we need a generic pages controller for the index page.

1. Remove the `public/index.html` file.
2. $ rails generate controller pages index

And the routes

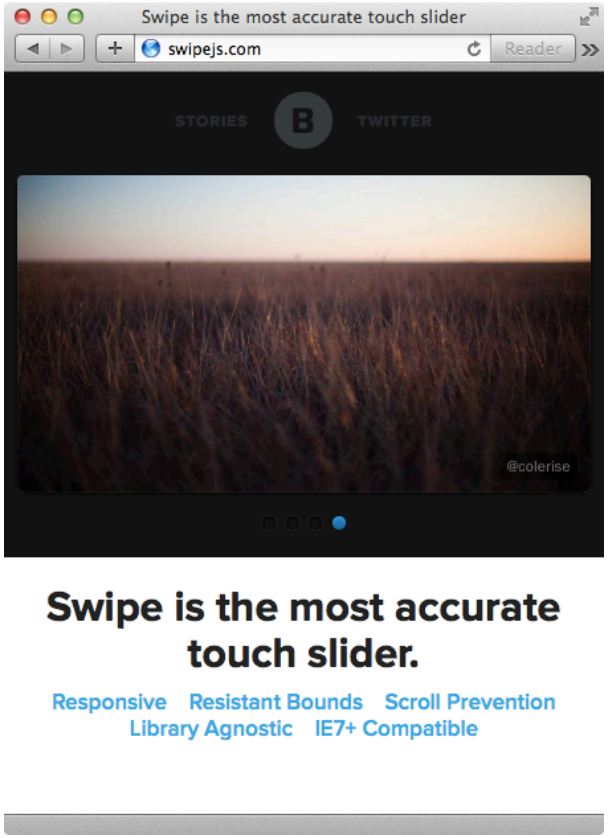
```
PhotoGallery::Application.routes.draw do
  resources :albums do
    resources :photos
  end

  root :to => 'pages#index'
end
```

So we got a temporary index page.

# Adding swipejs library

How about creating a slideshow in the home page? For slideshow, we will use the SwipeJS library.




Let's find it at: https://github.com/bradbirdsall/Swipe

Then put the swipe.js in the `vendor/assets/javascripts/` folder.

And we need to include it in the `application.js` file.

```
//= require jquery
//= require jquery_ujs
//= require swipe
//= require_tree .
```

From the SwipeJS doc, we copy and paste the styling to `app/assets/stylesheets/pages.css.scss` file.

```
.swipe {
  overflow: hidden;
  visibility: hidden;
  position: relative;
}
.swipe-wrap {
  overflow: hidden;
  position: relative;
}
.swipe-wrap > div {
  float:left;
  width:100%;
  position: relative;
}
```

And then the logic to get it started. `pages.js.coffee` file:

```
$ ->
  window.mySwipe =
Swipe(document.getElementById('slider'), {
    auto: 3000
  })
```

**Create the cover cropping style**

It would be best to create a dedicate cropping ratio for our slideshow, so we will add it to our paperclip gem.

```
has_attached_file :file, styles: {
  cover: "1000x600#",
  medium: "300x300>",
  thumb: "100x100>" },
  default_url: "/images/:style/missing.png"
```

Then we would refresh the cropping thumbnail generation with the paperclip command.

```
$ rake paperclip:refresh CLASS=Photo
```

Now we need some photos for the home page. In the pages controller:

```
class PagesController < ApplicationController
  def index
    @slide_photos = Photo.limit(5)
    @photos = Photo.limit(10)
  end
end
```

And we can use the following code in view for the related `@slide_photos` collection.

```
<div id='slider' class='swipe'>
  <div class='swipe-wrap'>
    <%- @slide_photos.each do |p| %>
      <div><%= image_tag p.file.url(:cover) %></div>
    <%- end %>
  </div>
</div>
```

Until now, we got a slideshow in the home page.


It is not looking good. What we want is a full layout. In the next session, we will build the entire layout for the gallery.

# Building gallery website layout

A full layout usually contians a bunch of codes. For the ease of demonstration, we will just create a basic grid layout with 1 or 2 columns.

Here is the style for a generic layout that we can place in the pages.css.scss.

```
/* Layout */
* {
    box-sizing: border-box;
}
.row {
  width: 960px;
  max-width: 100%;
  margin: auto;
  overflow: auto;

  .row {
    width: auto;
    max-width: none;
    margin: 0 -5px;
  }
}

.col {
  float:left;
  padding: 0 5px;

  &.full {
    width: 100%;
  }

  &.half {
    width: 50%;
  }
}

img {
  max-width: 100%;
}
```

*Note*: For a more completed CSS grid layout. please check my jsFiddle: http://jsfiddle.net/makzan/jktAT/

And some CSS rules for specific elements.

```
/* page element */
nav {
  ul {
    list-style: none;
    padding: 0;
    margin: 0;

    li {
      float: left;
      padding-right: 10px;
    }
  }
}

.copyright {
  text-align: right;
}
```

Now we have the layout rule, we can apply them to the `application.html.erb` file.

```
<body>
  <header class='row'>
    <div class='full col'>
      <nav>
        <ul>
          <li><%= link_to 'Home', root_path %></li>
          <li>Link</li>
          <li>Link</li>
        </ul>
      </nav>
    </div>
  </header>

  <div class='row'>
    <div class='full col'>
      <%= yield %>
    </div>
  </div>

  <footer class='row'>
    <div class='half col'>
      <nav>
        <ul>
          <li>Link</li>
```

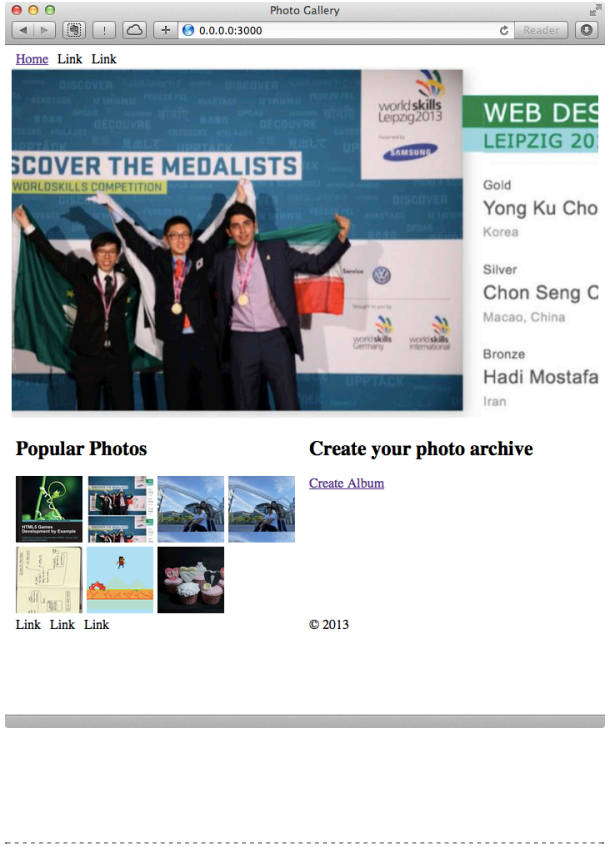
```
          <li>Link</li>
          <li>Link</li>
        </ul>
      </nav>
    </div>
    <div class="half col">
      <span class='copyright'>&copy; 2013</span>
    </div>
  </footer>
</body>
```

Back to the `pages/index.html.erb` file, we update the view with some layout elements.

```
<div class='row'>
  <div id='slider' class='swipe'>
    <div class='swipe-wrap'>
      <%- @slide_photos.each do |p| %>
        <div><%= image_tag p.file.url(:cover) %></div>
      <%- end %>
    </div>
  </div>
</div>

<div class="row">
  <div class="half col">
    <h2>Popular Photos</h2>
    <%- @photos.each do |p| %>
      <%= link_to album_photo_path(p.album, p) do %>
      <div class='popular-photo'
style='background:url(<%= image_path
p.file.url(:thumb) %>)'></div>
      <%- end %>
    <%- end %>
  </div>
  <div class="half col">
    <h2>Create your photo archive</h2>
    <p><%= link_to 'Create Album', new_album_path %>
</p>
  </div>
</div>
```

And finally we get a page layout with header, content and footer.



# Installing devise gem

In the `Gemfile`, we add the 'devise' gem.

```
gem 'devise'
```

```
$ bundle install
```

Execute the devise installation script.

```
$ rails generate devise:install
    create  config/initializers/devise.rb
    create  config/locales/devise.en.yml
===============================================================


Some setup you must do manually if you haven't yet:

  1. Ensure you have defined default url options in
your environments files. Here
     is an example of default_url_options appropriate
for a development environment
     in config/environments/development.rb:

       config.action_mailer.default_url_options = {
:host => 'localhost:3000' }

     In production, :host should be set to the actual
host of your application.

  2. Ensure you have defined root_url to *something*
in your config/routes.rb.
     For example:

       root :to => "home#index"

  3. Ensure you have flash messages in
app/views/layouts/application.html.erb.
     For example:

       <p class="notice"><%= notice %></p>
       <p class="alert"><%= alert %></p>

  4. If you are deploying on Heroku with Rails 3.2
only, you may want to set:

       config.assets.initialize_on_precompile = false

     On config/application.rb forcing your application
to not access the DB
     or load models when precompiling your assets.

  5. You can copy Devise views (for customization) to
your app by running:

       rails g devise:views

===============================================================
```

The setup generates two files.

1. `config/initializers/devise.rb` that contains all the devise setting.
2. `config/locales/devise.en.yml` that describes the English wordings.

**Setting up Devise model**

And now we can add devise to the User model. If the model isn't existed, it will create one.

```
$ rails generate devise User
    invoke  active_record
    create
db/migrate/20131011153539_devise_create_users.rb
    create    app/models/user.rb
    invoke    test_unit
    create      test/unit/user_test.rb
    create      test/fixtures/users.yml
    insert    app/models/user.rb
     route  devise_for :users
```

The key file is the `app/models/user.rb` file. There is a devise method that set the enabled modules.

```
class User < ActiveRecord::Base
  # Include default devise modules. Others available
are:
  # :confirmable, :lockable, :timeoutable and
:omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable,
:validatable
```

```
  # Setup accessible (or protected) attributes for
your model
  attr_accessible :email, :password,
:password_confirmation, :remember_me
  # attr_accessible :title, :body
end
```

**Devise routing** Devise add a route setting to the `routes.rb` file.

Let's take a look at the routes by running `rake routes`.

```
$ rake routes
          new_user_session GET
/users/sign_in(.:format)
devise/sessions#new
              user_session POST
/users/sign_in(.:format)
devise/sessions#create
      destroy_user_session DELETE
/users/sign_out(.:format)
devise/sessions#destroy
             user_password POST
/users/password(.:format)
devise/passwords#create
         new_user_password GET
/users/password/new(.:format)
devise/passwords#new
        edit_user_password GET
/users/password/edit(.:format)
devise/passwords#edit
                           PUT
/users/password(.:format)
devise/passwords#update
  cancel_user_registration GET
/users/cancel(.:format)
devise/registrations#cancel
         user_registration POST   /users(.:format)
devise/registrations#create
     new_user_registration GET
/users/sign_up(.:format)
devise/registrations#new
    edit_user_registration GET
/users/edit(.:format)
devise/registrations#edit
                           PUT    /users(.:format)
devise/registrations#update
                           DELETE /users(.:format)
devise/registrations#destroy
              album_photos GET
/albums/:album_id/photos(.:format)
photos#index
                           POST
/albums/:album_id/photos(.:format)
photos#create
           new_album_photo GET
/albums/:album_id/photos/new(.:format)      photos#new
          edit_album_photo GET
/albums/:album_id/photos/:id/edit(.:format)
photos#edit
               album_photo GET
/albums/:album_id/photos/:id(.:format)
photos#show
                           PUT
/albums/:album_id/photos/:id(.:format)
photos#update
                           DELETE
/albums/:album_id/photos/:id(.:format)
photos#destroy
                    albums GET    /albums(.:format)
albums#index
                           POST   /albums(.:format)
albums#create
                 new_album GET
/albums/new(.:format)                       albums#new
                edit_album GET
/albums/:id/edit(.:format)
albums#edit
                     album GET
/albums/:id(.:format)
albums#show
                           PUT
/albums/:id(.:format)
albums#update
                           DELETE
/albums/:id(.:format)
albums#destroy
                      root        /
pages#index
```

Do you see the registration routes. If you try to remove the `:registerable` option in the User model class, you route becomes the following without the registrable path.

```
$ rake routes
      new_user_session GET    /users/sign_in(.:format)
devise/sessions#new
          user_session POST   /users/sign_in(.:format)
devise/sessions#create
  destroy_user_session DELETE
```

```
/users/sign_out(.:format)
devise/sessions#destroy
          album_photos GET
/albums/:album_id/photos(.:format)
photos#index
                       POST
/albums/:album_id/photos(.:format)
photos#create
       new_album_photo GET
/albums/:album_id/photos/new(.:format)      photos#new
      edit_album_photo GET
/albums/:album_id/photos/:id/edit(.:format)
photos#edit
           album_photo GET
/albums/:album_id/photos/:id(.:format)
photos#show
                       PUT
/albums/:album_id/photos/:id(.:format)
photos#update
                       DELETE
/albums/:album_id/photos/:id(.:format)
photos#destroy
                albums GET    /albums(.:format)
albums#index
                       POST   /albums(.:format)
albums#create
             new_album GET    /albums/new(.:format)
albums#new
            edit_album GET
/albums/:id/edit(.:format)
albums#edit
                 album GET    /albums/:id(.:format)
albums#show
                       PUT    /albums/:id(.:format)
albums#update
                       DELETE /albums/:id(.:format)
albums#destroy
                  root        /
pages#index
```

**Migratting the database**

Since this is a new model, the database should reflect the new table. Before we start running the server, we need migrate the database.

```
$ rake db:migrate
==  DeviseCreateUsers: migrating
=============================================
-- create_table(:users)
   -> 0.0331s
-- add_index(:users, :email, {:unique=>true})
   -> 0.0026s
-- add_index(:users, :reset_password_token,
{:unique=>true})
   -> 0.0015s
==  DeviseCreateUsers: migrated (0.0377s)
=====================================
```

**Contrilling the views and forms**

If we want to manage our own login view, we can generate the view from the gem and it will be used by rails.

```
$ rails generate devise:views
    invoke  Devise::Generators::SharedViewsGenerator
    create    app/views/devise/shared
    create    app/views/devise/shared/_links.erb
    invoke  form_for
    create    app/views/devise/confirmations
    create
app/views/devise/confirmations/new.html.erb
    create    app/views/devise/passwords
    create    app/views/devise/passwords/edit.html.erb
    create    app/views/devise/passwords/new.html.erb
    create    app/views/devise/registrations
    create
app/views/devise/registrations/edit.html.erb
    create
app/views/devise/registrations/new.html.erb
    create    app/views/devise/sessions
    create    app/views/devise/sessions/new.html.erb
    create    app/views/devise/unlocks
    create    app/views/devise/unlocks/new.html.erb
    invoke  erb
    create    app/views/devise/mailer
    create
app/views/devise/mailer/confirmation_instructions.html.e

    create
app/views/devise/mailer/reset_password_instructions.html

    create
app/views/devise/mailer/unlock_instructions.html.erb
```

Now we get all the devise view files so we can change whatever we want.

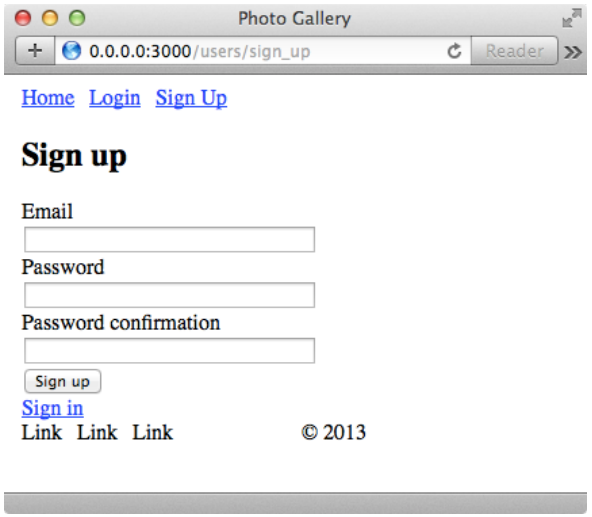
# Authenticating user

Now we can add the registraiton and login link to the view.

The `header` tag in `layout/application.rb` file becomes:

```
<header class='row'>
  <div class='full col'>
    <nav>
      <ul>
        <li><%= link_to 'Home', root_path %></li>
        <% if user_signed_in? %>
          <li><%= link_to "Logout (#
{current_user.email})", destroy_user_session_path,
:method => :delete %></li>
        <% else %>
          <li><%= link_to 'Login',
new_user_session_path  %></li>
          <li><%= link_to 'Sign Up',
new_user_registration_path  %></li>
        <% end %>
      </ul>
    </nav>
  </div>
</header>
```

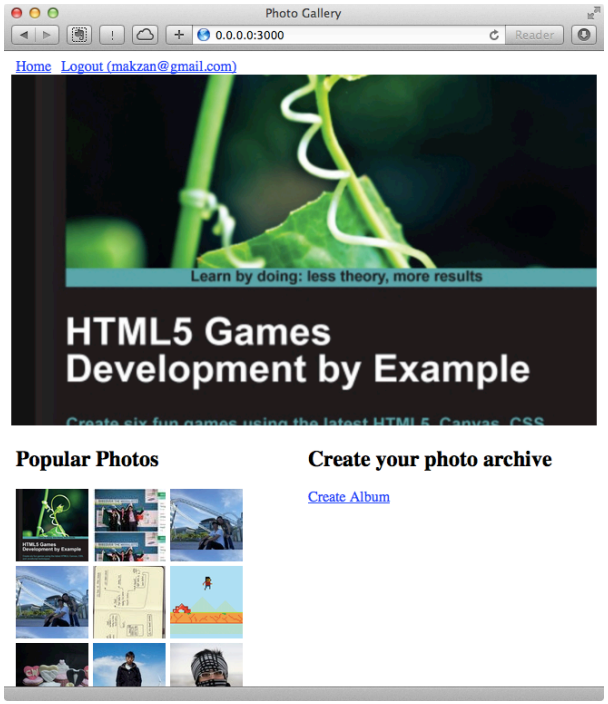
Now we have the sign up page linked from the top nav.



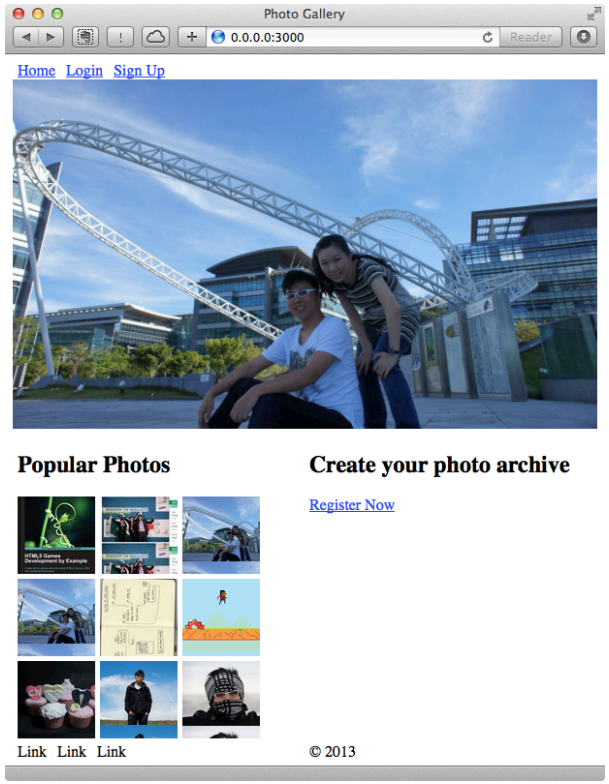

Also in the `views/pages/index.html.erb` file, we want to either link to the new album page or new registraiton page base on the current user session.

```
<% if user_signed_in? %>
  <p><%= link_to 'Create Album', new_album_path %></p>
<% else %>
  <p><%= link_to 'Register Now',
new_user_registration_path %></p>
<% end %>
```

The index page when logged out.




And the index page when logged in. Note the “login” and “logout” between these two images.



# Protecting upload

Before protecting uploading, we need to add the association between the user and the albums/photos.

Execute the following migration generation commands and migrate the database.

```
$ rails generate migration AddUserIdToAlbum
user_id:integer
$ rails generate migration AddUserIdToPhoto
user_id:integer
$ rake db:migrate
==  AddUserIdToAlbum: migrating
============================
-- add_column(:albums, :user_id, :integer)
   -> 0.0016s
==  AddUserIdToAlbum: migrated (0.0018s)
=================

==  AddUserIdToPhoto: migrating
============================
-- add_column(:photos, :user_id, :integer)
   -> 0.0009s
==  AddUserIdToPhoto: migrated (0.0011s)
=================
```

Then add `belongs_to :user` to both `album.rb` class and `photo.rb` class.

And the following to `user.rb` class.

```
has_many :photos
has_many :albums
```

Next, we would like to protect user upload in the photos controller.

```
before_filter :authenticate_user, only: [:new, :edit]

def authenticate_user
  redirect_to new_user_session_url unless
user_signed_in? and !@album.user.nil? and current_user
== @album.user
end
```

And we add the user association in the create method.

```
    def create
      @photo = @album.photos.new params[:photo]
      @photo.user = current_user
      if @photo.save
        redirect_to @album
      else
        render :new
      end
    end
```

Now when we go to any photo upload URL without logged in to the album owner, we will be redirected to the sign_in page.

Then we apply the same to the albums controller.

In the `albums_controller` file.

```
before_filter :authenticate_user, only: [:new]

def authenticate_user
  redirect_to new_user_session_url unless
user_signed_in?
end
```

And the user-albums association in the `create` method.

```
  def create
    @album = Album.new params[:album]
    @album.user = current_user
    if @album.save
      redirect_to @album
    else
      render :new
    end
  end
```

It's time to test the function in web browser.

We can create albums and upload photos as normal after we logged in. Now try to create another user account and access the album you just created, you should not be able to upload any new photos.

# The view

One last thing, we don't want the `upload photo` or `edit` link appears on the photo that isn't belonged to the current user.

In the `views/photos/show.html.erb` file.

```
<%- if user_signed_in? and current_user == @photo.user
%>
  <%= link_to 'Edit', edit_album_photo_path(@album,
@photo) %>
<%- end %>
```

And the `views/albums/show.html.erb` file.

```
<%- if user_signed_in? and current_user == @album.user
%>
  <p><%= link_to 'Upload new photo',
new_album_photo_path(@album) %></p>
<%- end %>
```

# Challenges

What's more in using devise gem? Check their wiki.

Want admin? You can `rails generate devise AdminUser` to create an Admin user.

Want more access control? I would suggest to use cancan gem together with the devise gem.

Want private albums/photos? Add a attribute to albums and photos to indicate it is private. Then ignore public/private photos when list them in index.