MicroPost-app Workshop

Ruby on Rails Workshop at Kasetsart University.

Prepared by: Ino Techne Co., Ltd.

July 2012

Ino Techne Co., Ltd. 5th Floor - Siri Building, 3 Saladaeng Soi 1 Silom, Bangkrak, Bangkok, Thailand, 10500 **T** 02 636 3297 **W** www.inotechne.com

Table of Contents

MicroPost App

How To?

We will explain every step in plain text

1. If you see black box with '$' symbol you have to type following text into terminal.
2. If you see gray box you must type following color.
    - 2.1. For green text this mean code is new or edit, You need to type in text editor inside your project.
    - 2.2. For gray text is code that already in that file you can use it to inspect where is section of code in that file.
    - 2.3. For orange text which mean code comment it some advice or explain green code.

Goals

We will create simple app from scratch by using Ruby on Rails 3.2.1. in this application it should be working all this requirement.

1. Guest can register and login into website as an User.
2. User can post some message.
3. User can delete his message.
4. User can follow other user and see other user message that they followed.
5. User can unfollow other user.
6. User statistic e.g. post count, follower and following on other User and himself.

Step 1 : Create new rails application named micropost-app

cd to path that you want to put project

create new project named "micropost-app" by using rails. ***(Don't name your apps similar with your model)***

```
$ rails new micropost-app
```

cd inside the project

```
$ cd micropost-app
```

Step 2 : Modified Gemfile & config/database.yml and generate rspec

Gem is package of library that people created to make your application more simpler you can install and use in Ruby application

We will add gem 'authlogic' for authentication and gem 'rspec-rails', 'shoulda-matchers' and 'forgery' for testing

open Gemfile and add follow line:

```
# Deploy with Capistrano
# gem 'capistrano'

# To use debugger
# gem 'ruby-debug19', :require => 'ruby-debug'

# User authentication
gem 'authlogic'

# Unit testing
group :development, :test do
  gem 'forgery', '0.5.0'
  gem 'rspec-rails', '2.11.0'
end

group :test do
  gem 'shoulda-matchers', '1.2.0'
end
```

We can determine which gem will be load in specific environment eg. gem 'shoulda-matchers' will only load in test environment

- Install gem that we just add

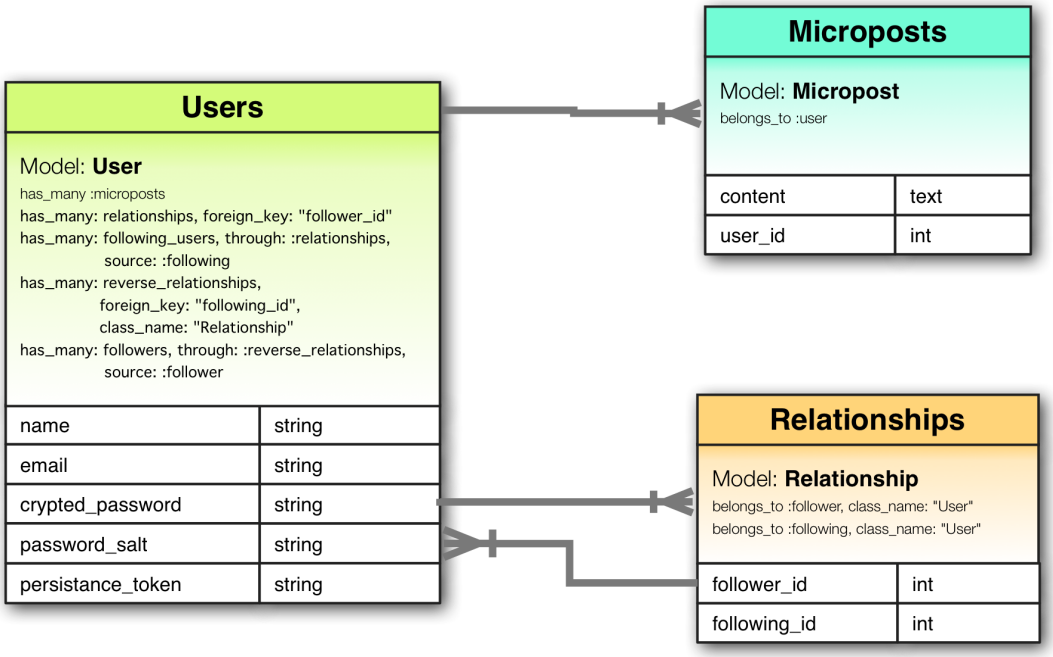
```
$ bundle install
```

- Generate spec directory for write unit testing inside and force rails generate to use rspec

```
$ rails g rspec:install
```

modified **config/database.yml** if needed. In this example we'll use sqlite and it's a default database so we don't have to config anything.

Step 3 : Create Model User, Micropost and Relationship



Following these statements rails will go to generate model in **app/models/** and migration file in **db/migrate/**. In migration file will see all fields that we define in each table and rails will auto generate field created_at and updated_at into every table.

Notice: Model name should be Singular (It's a rails convention)

Micropost model

```
$ rails g model Micropost content:text user_id:integer
```

Relationship model

```
$ rails g model Relationship follower_id:integer following_id:integer
```

User model

```
$ rails g model User name:string email:string crypted_password:string
password_salt:string persistence_token:string
```

You will have all migration files in **db/migrate/** make sure that every fields are correct if it's not correct you can edit it. (But if it's already in production environment we suggest to not do that, you have to create another migration file to edit column name or anything that you make a mistake about it)

Create Database

```
$ rake db:create
```

Create all table following migration file

```
$ rake db:migrate
```

Create database for test environment

```
$ rake test:prepare
```

Step 4 : Create Model UserSession

create UserSession model

```
$ rails g model UserSession email:string password:string --migration=false
```

Edit file **app/models/user_sessions.rb** to extend Authlogic::Session::Base

```
class UserSession < Authlogic::Session::Base

end
```

Step 5 : Create Controllers

Notice: Controller name should be Plural (It's a rails convention)

- Using rails to generate Controller and add some actions, it will create files in **app/controllers/**
- Generate UsersController with actions index, show, new and edit

```
$ rails g controller Users index show new edit
```

- Generate MicropostsController with action index

```
$ rails g controller Microposts index
```

- Generate RelationshipsController with no action

```
$ rails g controller Relationships
```

- Generate UserSessionsController with action new

```
$ rails g controller UserSessions new
```

- Modified **config/routes.rb** for routing

```
MicropostApp::Application.routes.draw do

  resources :users, :except => [:destroy]
  resources :microposts, :only => [:index, :create, :destroy]
  resources :relationships, :only => [:create, :destroy]
  resources :user_sessions, :only => [:new, :create, :destroy]

end
```

Purposes of Routing is to maps requests to controller action methods and it enables the dynamic generation of URLs for you. We use 'resource' routing to declare all of common routes for a given resourceful controller. Instead of declaring routes for each action (index, show, new, edit, create, update and destroy) in a single line of code.

- You can check that rails has already created routes that you have set by

```
$ rake routes
```

You will see the result like these :

```
Monsirees-Mac-mini:micropost-app monsiree_t$ rake routes
            root        /                           users#index
           users GET    /users(.:format)            users#index
                 POST   /users(.:format)            users#create
        new_user GET    /users/new(.:format)        users#new
       edit_user GET    /users/:id/edit(.:format)   users#edit
            user GET    /users/:id(.:format)        users#show
                 PUT    /users/:id(.:format)        users#update
      microposts GET    /microposts(.:format)       microposts#index
                 POST   /microposts(.:format)       microposts#create
       micropost DELETE /microposts/:id(.:format)   microposts#destroy
   relationships POST   /relationships(.:format)    relationships#create
    relationship DELETE /relationships/:id(.:format) relationships#destroy
   user_sessions POST   /user_sessions(.:format)    user_sessions#create
new_user_session GET    /user_sessions/new(.:format) user_sessions#new
    user_session DELETE /user_sessions/:id(.:format) user_sessions#destroy
```

we can use url that rails generate inside application to create link or redirect to controller and action that we want instead of hard code eg.

users_url GET will link to users#index (users controller action index)

Step 6 : Add CSS

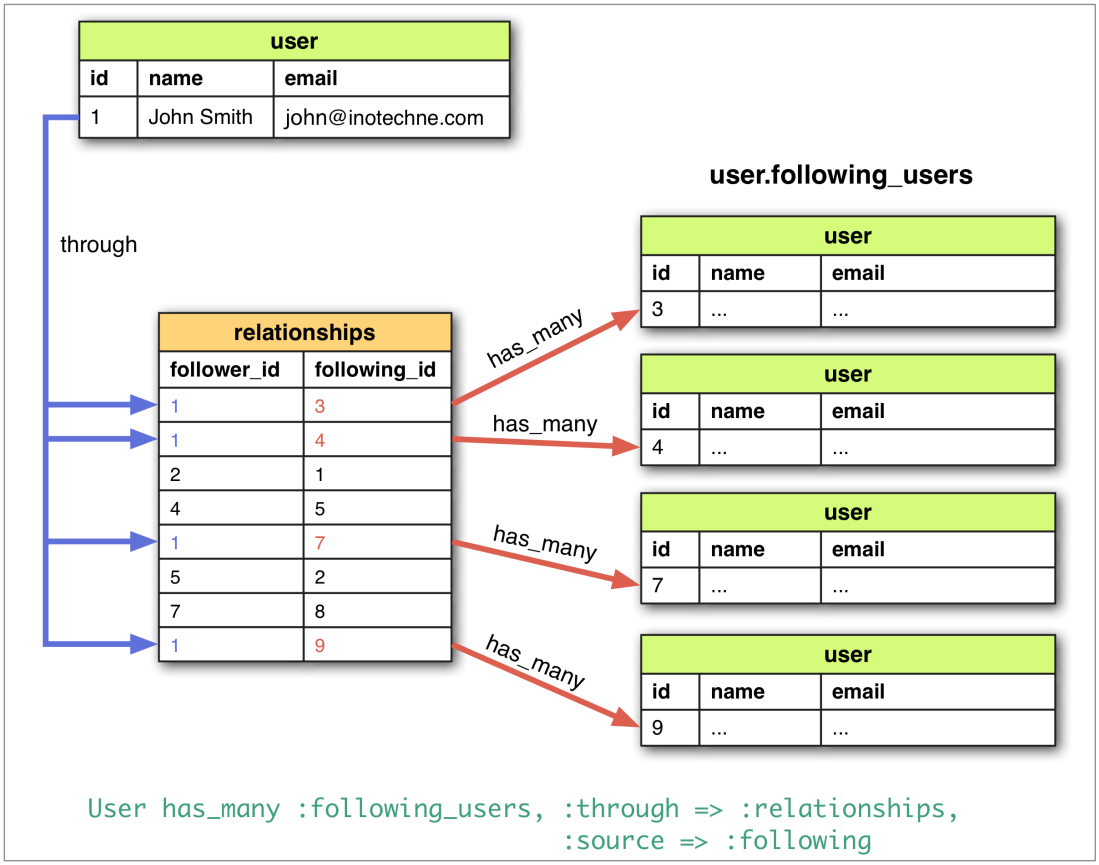
- You can create your own stylesheet or copy that already done. https://gist.github.com/3950603 to **app/assets/stylesheets/style.css.scss**

Step 7 : Assign relation to each model

- Add relation for User model by add following lines in **app/models/user.rb**

```
class User < ActiveRecord::Base

  has_many :microposts

  has_many :relationships,
           :foreign_key => :follower_id

  has_many :following_users,
           :through => :relationships,
           :source => :following

  has_many :reverse_relationships,
           :class_name => "Relationship",
           :foreign_key => :following_id

  has_many :follower_users,
           :through => :reverse_relationships,
           :source => :follower
end
```

1. User has many microposts
2. User has many relationships whereas user is a follower so it's like you can follow many people
3. User has many following users through relationships like you can have many users who you are following
4. User has many reverse relationships whereas user is a following so it's like you can have followed by many people
5. User has many follower users through reverse relationships like you can have many users following you



- Add relation for Micropost model by add following lines in **app/models/micropost.rb**

```
class Micropost < ActiveRecord::Base
  belongs_to :user

end
```

- Add relation for Relationship model by add following lines in **app/models/relationship.rb**

```
class Relationship < ActiveRecord::Base
  belongs_to :follower, :class_name => "User"
  belongs_to :following, :class_name => "User"

end
```

Step 8 : Set root url and add header in layouts

- Set root_url in **config/routes.rb**

```
MicropostApp::Application.routes.draw do

  root :to => "users#index"

  resources :users, :except => [:destroy]
  resources :microposts, :only => [:index, :create, :destroy]
  resources :relationships, :only => [:create, :destroy]
  resources :user_sessions, :only => [:new, :create, :destroy]

end
```

- Delete file **public/index.html**

These 2 steps will make application go to users controller action index at first page but if you don't remove **public/index.html** it will not work.

- Modified **app/views/layouts/application.html.erb** inside <body></body>

```
<body>
  <div id="header">
    <%= link_to "MICROPOST", root_url, :class => "site-name" %>
    <div id="navigation">
      <%= link_to "All users", users_url %>
      <%= link_to "Sign up", new_user_url %>
    </div>
  </div>

  <div id="content">
    <%= yield %>
  </div>
</body>
```

- Those lines will add following link in header (You can run "rake routes" to see all routes)

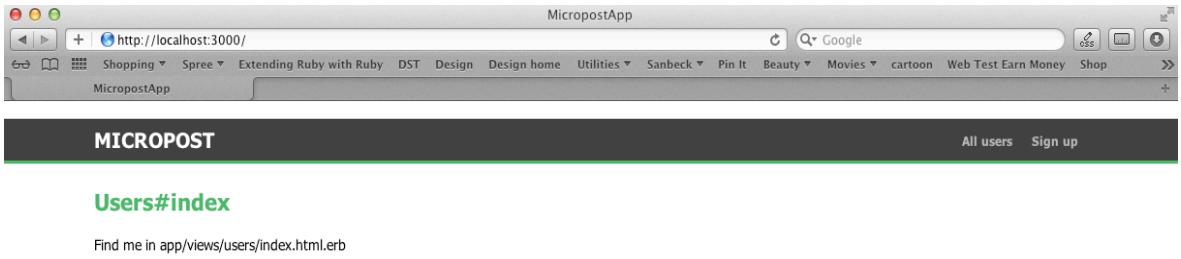
- "MICROPOST" link to site root url
- "All users" link to users#index (users controller action index)
- "Sign up" link to users#new (users controller action index)

Notice: If you and anything in **app/views/layouts/application.html.erb** it will show in every page.

- Check that all we modified are working correctly.
- Start server by

```
$ rails s
```

Open your web browser and go to http://localhost:3000/



Step 9 : Create Sign up page

- Modified User model **app/models/user.rb**

```
  has_many :follower_users,
           :through => :reverse_relationships,
           :source => :follower

  acts_as_authentic

  validates :name, :presence => true
  validates :email, :presence => true

end
```

- Use authlogic in user model
- Validate field name and field email, it can't be blank or nil

- Modified new method new in **app/controllers/user_controllers.rb**

```
    def show
    end

    def new
      @user = User.new
    end

  end
```

- Replace code in signup page **app/views/users/new.html.erb** to create sign up form which user have to input name, email, password and confirmation password

```
<div id="register_form">
  <h1>Sign up</h1>

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


      <%= f.label :name %>
      <%= f.text_field :name %>

      <%= f.label :email %>
      <%= f.text_field :email %>

      <%= f.label :password %>
      <%= f.password_field :password %>

      <%= f.label :password_confirmation %>
      <%= f.password_field :password_confirmation %>

      <%= f.submit %>
  <% end %>
</div>
```

- Create new file **app/views/shared/_form_error_messages.html.erb** and add these :

```
<% if model_object.errors.any? %>
    <div class="error-message">
      <ul>
        <% model_object.errors.full_messages.each do |msg| %>
            <li><%= msg %></li>
        <% end %>
      </ul>
    </div>
 <% end %>
```

We create this file to show error message, the reason that we separate file for error message is for other pages can reuse this again, just render this partial.

Notice: Partial file name will start with underscore.

- Add render partial in **app/views/users/new.html.erb** between form_for and end

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

  <%= render :partial => "shared/form_error_messages",
             :locals => { :model_object => @user} %>

  <%= f.label :name %>
  <%= f.text_field :name %>
```

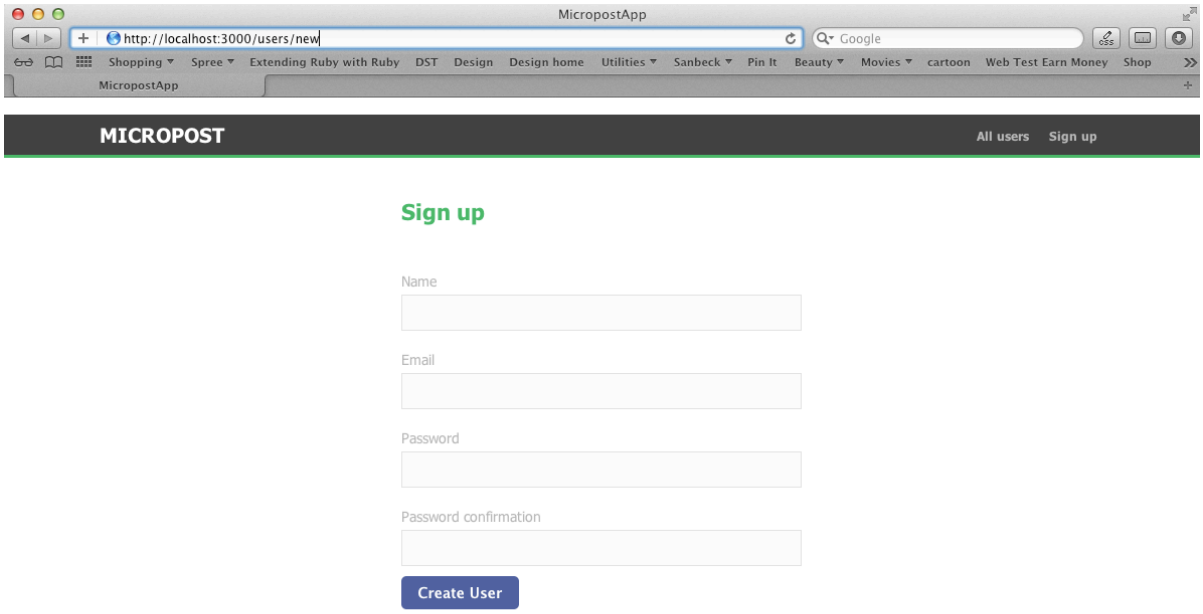
- Add create method in **app/controllers/user_controllers.rb**

```
def new
 @user = User.new
end

def create
  @user = User.new(params[:user])

  if @user.save
    flash[:notice] = "Sign up successfully."
    redirect_to root_url
  else
    render :action => :new
  end
end
```

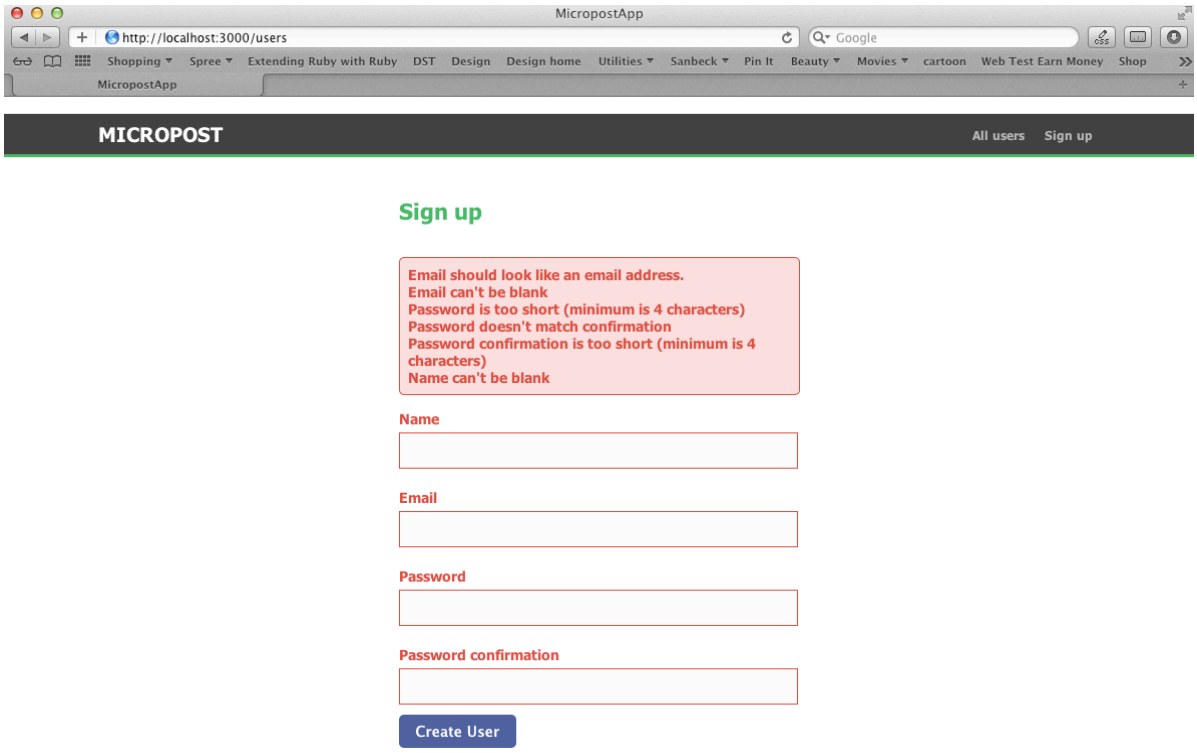
- Add notice message in **app/views/layouts/application.html.erb**

```
            <%= link_to "All users", users_url %>
            <%= link_to "Sign up", new_user_url %>
          </div>
        </div>

        <div id="content">

          <% if flash[:notice] %>
              <div class="notice-message">
                <%= flash[:notice] %>
              </div>
          <% end %>

          <%= yield %>
        </div>
```

Check to see how it works by open browser and go to http://localhost:3000/users/new



If click on Create User button, it will show all error messages.



You will see that authlogic have validate some fields for you like:

- Email have to be in the right format.
- Password cannot be blank
- Password must be more than 4 characters
- Password confirmation should be the same as password.

Step 10 : Create seed data

- open db/seeds.rb add followings lines:

```
puts "Create Users ..."

user1 = User.create(:name => "John Smith", :email => "john@inotechne.com",
        :password => "johntest", :password_confirmation => "johntest")

user2 = User.create(:name => "Anny Smith", :email => "anny@inotechne.com",
                    :password => "annytest", :password_confirmation=> "annytest")

Relationship.create(:follower => user2, :following => user1)
```

There are 3 lines that have gray highlight on it are use to create 2 users and 1 relationship.

- Execute it by run rake db:seed on terminal

```
$ rake db:seed
```

Step 11 : Create List all users page and Show each user page

- Modified method index in **app/controllers/users_controller.rb**

```
class UsersController < ApplicationController
  def index
    @users = User.all
  end
```

Query all users to show on index view

- Modified list all users view in **app/views/users/index.html.erb**

Replace the exist code with these code:

```
<h1> All Users </h1>

<!-- Use each block to create link of user name-->
<% @users.each do |user| %>
    <div class="user"><%= link_to user.name, user_url(user.id) %></div>
<% end %>
```

- Modified show method in **app/controllers/users_controller.rb**

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

# action show get id from post request then find user to show on views
def show
  @user = User.find(params[:id])
end
```

- Modified show page in **app/views/users/show.html.erb**

Replace the exist code with these code:

```
<div class="user-information">
  <span class="user-name">&lt;%= @user.name %&gt;</span>
</div>
```

Step 12 : Create login/logout page

- add this code to **config/routes.rb**

```
resources :relationships, :only => [:create, :destroy]
resources :user_sessions, :only => [:new, :create, :destroy]

match "login" => "user_sessions#new"
match "logout" => "user_sessions#destroy"
```

- add login link to **app/views/layouts/application.html.erb**

```
<div id="header">
  <%= link_to "MICROPOST", root_url, :class => "site-name" %>
  <div id="navigation">
    <%= link_to "All users", users_url %>
    <%= link_to "Sign up", new_user_url %>
    <%= link_to "Login", login_url %>
  </div>
</div>
```

- modified method new in **app/controllers/user_sessions_controller.rb**

```
    def show
    end

    def new
      @user_session = UserSession.new
    end

  end
```

- replace this code to **app/views/user_sessions/new.html.erb**

```
  <div id='login_form'>
    <h1>Login</h1>

    <%= form_for @user_session do |f| %>
        <%= render :partial => "shared/form_error_messages", :locals => {
:model_object => @user_session} %>

        <%= f.label :email %>
        <%= f.text_field :email %>

        <%= f.label :password %>
        <%= f.password_field :password %>

        <%= f.submit "Login" %>
    <% end %>
  </div>
```

- add method create in **app/controllers/user_sessions_controller.rb**

```
  def create
    @user_session = UserSession.new(params[:user_session])
    if @user_session.save
      flash[:notice] = "Logged in successfully."
      redirect_to root_url
    else
      render :action => :new
    end
  end
```

- add link to show current logged in user in **app/views/layouts/application.html.erb**

```
      <%= link_to "All users", users_url %>
      <% if UserSession.find %>
        <%= link_to UserSession.find.record.name,
user_url(UserSession.find.record.id) %>
      <% else %>
        <%= link_to "Sign up", new_user_url %>
        <%= link_to "Login", login_url %>
      <% end %>
```

This will do some trick and show link up to user is logged in. But as you can see the code is look ugly and not readable we will change some code by put it as helper_medthod.

- add this code to **app/controllers/application_controller.erb**

```
     class ApplicationController < ActionController::Base
       protect_from_forgery

       helper_method :current_user, :current_user_session

       def current_user_session
         return @current_user_session if @current_user_session
         @current_user_session = UserSession.find
       end

       def current_user
         return @current_user if @current_user
         @current_user = current_user_session && current_user_session.record
       end
     end
```

- replace **UserSession.find** and **UserSession.find.record** in **app/views/layouts/application.html.erb** with **current_userlivepage.apple.comlivepage.apple.com**

```
   <%= link_to "All users", users_url %>
   <% if current_user %>
     <%= link_to current_user.name, user_url(current_user) %>
   <% else %>
     <%= link_to "Sign up", new_user_url %>
     <%= link_to "Login", login_url %>
   <% end %>
```

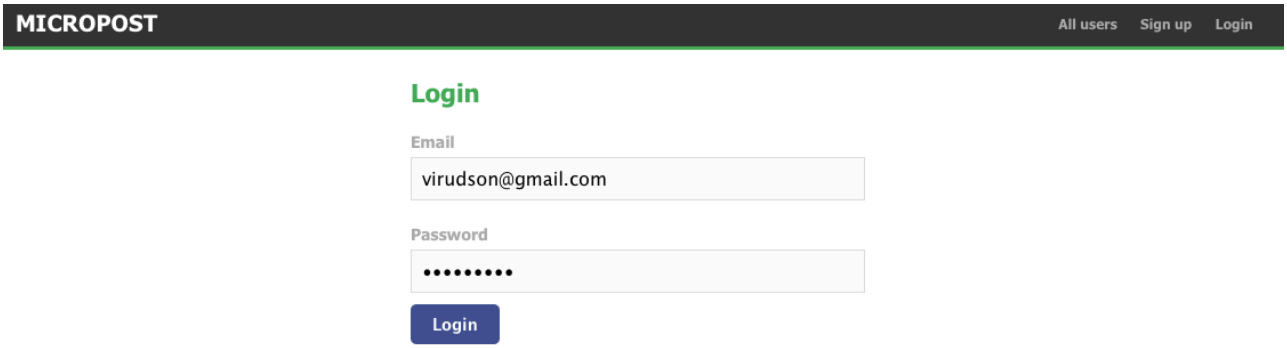
- add logout link for login user in **app/views/layouts/application.html.erb**

```
<% if current_user %>
  <%= link_to current_user.record.name, user_url(current_user.record.id)%>
  <%= link_to "Logout", logout_url %>
<% else %>
  <%= link_to "Sign up", new_user_url %>
  <%= link_to "Login", login_url %>
<% end %>
```

- add method destroy for clear user_session in **app/controllers/user_sessions_controller.rb**

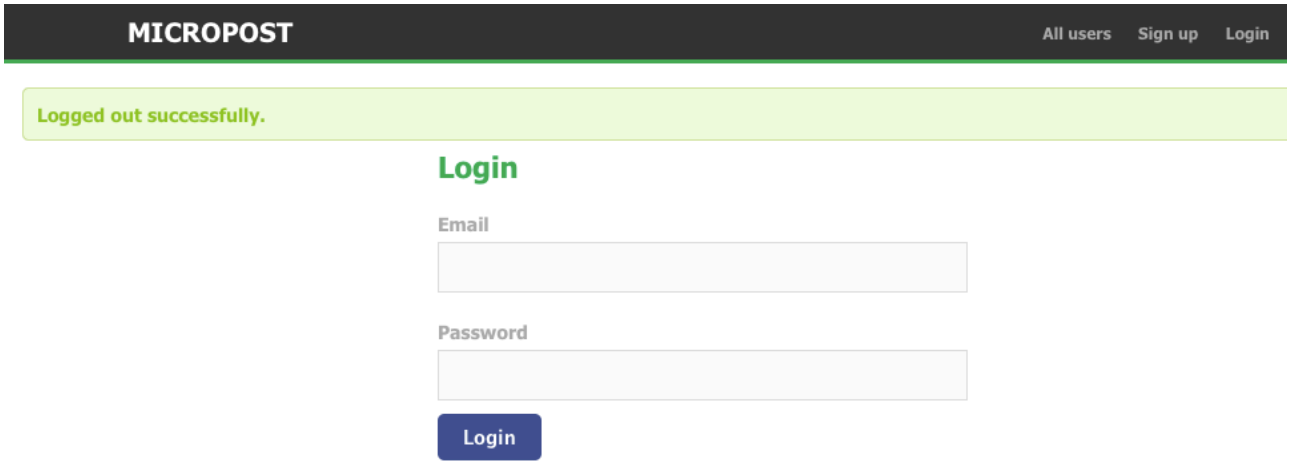
```
class UserSessionsController < ApplicationController
  def new
    @user_session = UserSession.new
  end

  def create
    @user_session = UserSession.new(params[:user_session])
    if @user_session.save
      flash[:notice] = "Logged in successfully."
      redirect_to root_url
    else
      render :action => :new
    end
  end

  def destroy
    current_user_session.destroy
    flash[:notice] = "Logged out successfully."
    redirect_to login_url
  end
end
```

Let's look what we have done. At login Page.



And after login to application

When click on log out button.

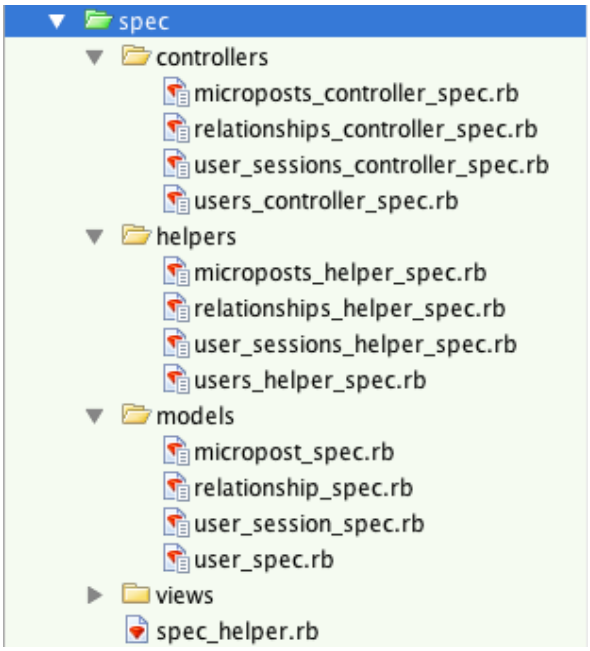


Step 13 : Write a Unit Test with RSpec

What is RSpec?

RSpec is testing tool for the Ruby programming language. It is designed for programmer making test productive with clean code and enjoyable experience with testing.

All the RSpec test is store file in /spec folder and it's has sub folder that store model spec, controller spec, view spec and helper spec. Each spec will automatic create after you run rails command create controller or model every time.



What to test and what not to test?

Before we get start testing we should know what we need to test and not need to test, usually we will focus testing on application feature and requirement should not test what is already tested e.g. testing email format because it already done on auth logic gem.

While as a beginner you shouldn't worry about what not to test much on one day you will know this better from experience, Then let's create simple unit test on our application.

- add default scope of test to **spec/models/user_spec.rb**

```
require 'spec_helper'

describe User do
  context "Database Schema" do
  end

  context "Associations" do
  end

  context "Validations" do
  end

  context "Methods" do
  end

  context "Create new User" do
  end
end
```

add first spec in User spec

```
describe User do
  context "Database Schema" do
    it { should have_db_column(:name).of_type(:string) }
    it { should have_db_column(:email).of_type(:string) }
  end

  context "Associations" do
    it { should have_many(:relationships) }
    it { should have_many(:following_users).through(:relationships) }
    it { should have_many(:reverse_relationships).class_name('Relationship') }
    it { should have_many(:follower_users).through(:reverse_relationships) }
  end

  context "Validations" do
    it { should validate_presence_of(:name) }
    it { should validate_presence_of(:email) }
  end

  context "Methods" do
  end

  context "Create new User" do
  end
end
```

As you can see in context "Database Schema" we test only name and email because other field in users table is generated by Rails and AuthLogic that why we do not need to test it.

- run test by using rspec command

```
$ rspec spec/models/user_spec.rb
```

if you are using **Windows** use this command

```
$ bundle exec rspec spec/models/user_spec.rb
```

Your console will show result look like this.

```
Earth-Mac-mini:micropost-app-inotechne Earth$ rspec spec/models/user_spec.rb
........

Finished in 0.11786 seconds
8 examples, 0 failures

Randomized with seed 43443
```

It's not readable for human. you can use option to show color and format by using this command for tell rspec to show result more readable.

```
$  rspec --color --format doc spec/models/user_spec.rb
```

Your console will show color and each spec group by context and color will tell you which is pass or failure, This is Awesome!!.

Notice For Windows user if your command-prompt will not display color but it will display color code.

```
Earth-Mac-mini:micropost-app-inotechne Earth$ rspec --color --format doc spec/models/user_spec.rb

User
  Database Schema
    should have db column named name of type string
    should have db column named email of type string
  Associations
    should have many relationships
    should have many following_users through relationships
    should have many reverse_relationships class_name => Relationship
    should have many follower_users through reverse_relationships
  Validations
    should require name to be set
    should require email to be set

Finished in 0.05462 seconds
8 examples, 0 failures

Randomized with seed 35661
```

You not necessary to put `**--color --format doc**` every time, you can create file named .rspec at application directory RSpec will autoload it every time you use rspec command.

- create .rspec file

```
$  echo --color --format doc >> .rspec
```

- and try to run rspec again

```
$ rspec spec/models/user_spec.rb
```

Your console will show result look like this.

```
Earth-Mac-mini:micropost-app-inotechne Earth$ echo --color --format doc >> .rspec
Earth-Mac-mini:micropost-app-inotechne Earth$ rspec spec/models/user_spec.rb

User
  Database Schema
    should have db column named name of type string
    should have db column named email of type string
  Associations
    should have many reverse_relationships class_name => Relationship
    should have many following_users through relationships
    should have many follower_users through reverse_relationships
    should have many relationships
  Validations
    should require name to be set
    should require email to be set

Finished in 0.05245 seconds
8 examples, 0 failures

Randomized with seed 36917
```

- update context ‘Create new User’ and spec in case of name or email is not preset

```
context "Create new User" do
  before(:each) do
    password = 'bq87/9h'
    @user = User.new(:name => 'Virudson',
                     :email => 'virudson.t@inotechne.com',
                     :password => password,
                     :password_confirmation => password)
  end

  it "when name is not present" do
    @user.name = " "
    @user.should_not be_valid
    @user.should have(1).error_on(:name)
  end

  it "when email is not present" do
    @user.email = " "
    @user.should_not be_valid
    @user.should have(2).error_on(:email)
  end

  it "when password is not present" do
    @user.password = @user.password_confirmation = " "
    @user.should_not be_valid
    @user.should have(1).error_on(:password)
  end

end
```

introduce to Forgery gem https://github.com/sevenwire/forgery/

document: http://sevenwire.github.com/forgery/

Forgery is gem for random data it very useful because when we are test some thing we should use variety of data and sometime it must meaningful.

- open rails console and try some sample data

```
$ rails c
```

```
$ Forgery::Basic.hex_color
```

```
$ Forgery::Name.full_name
```

```
$ Forgery::Personal.shirt_size
```

- refactor code in context by using forgery

```
context "Create new User" do
  before(:each) do
    password = Forgery(:basic).password
    @user = User.new(:name => Forgery(:name).first_name,
                     :email => Forgery(:internet).email_address,
                     :password => password,
                     :password_confirmation => password)
  end
```

Rails frame work has command for check line of code in application and compare with test line of code, that command is `**rake stats**`.

Step 14 : Add edit profile button

- replace code in **app/views/users/edit.html.erb**

```
<div id="register_form">
  <h1>Edit Profile</h1>

  <%= form_for(@user) do |f| %>
    <%= render :partial => "shared/form_error_messages",
               :locals => {:model_object => @user} %>

      <%= f.label :name %>
      <%= f.text_field :name %>

      <%= f.label :email %>
      <%= f.text_field :email %>

      <%= f.label :password %>
      <%= f.password_field :password %>

      <%= f.label :password_confirmation %>
      <%= f.password_field :password_confirmation %>

      <%= f.submit %>
  <% end %>
</div>
```

- add this code to **app/views/users/show.html.erb**

```
<div class="user-information">
  <span class="user-name">&lt;%= @user.name %&gt;</span>
  <% if @user == current_user %>
    <div class="edit-profile-button">
      <%= link_to "Edit profile", edit_user_url(current_user) %>
    </div>
  <% end %>
</div>
```

- update method edit and update in **app/controllers/users_controller.rb**

```
class UsersController < ApplicationController
  def edit
    @user = current_user
  end

  def update
    @user = current_user

    if @user.update_attributes(params[:user])
      flash[:notice] = "Profile was updated."
      redirect_to user_url(@user)
    else
      render :action => :edit
    end
  end
end
```

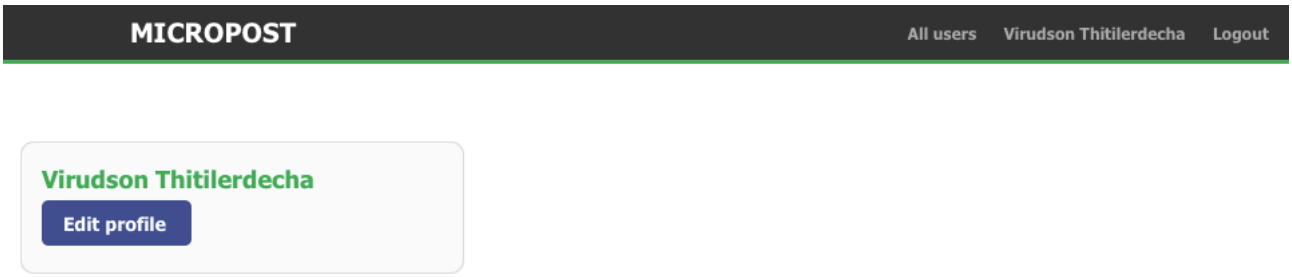
- cosmetic url by match url to use edit profile in **config/routes**

```
MicropostApp::Application.routes.draw do
  …
  …

  match "login" => "user_sessions#new"
  match "logout" => "user_sessions#destroy"
  match "edit_profile" => "users#edit"
end
```

- update url in **app/views/users/show.html.erb**

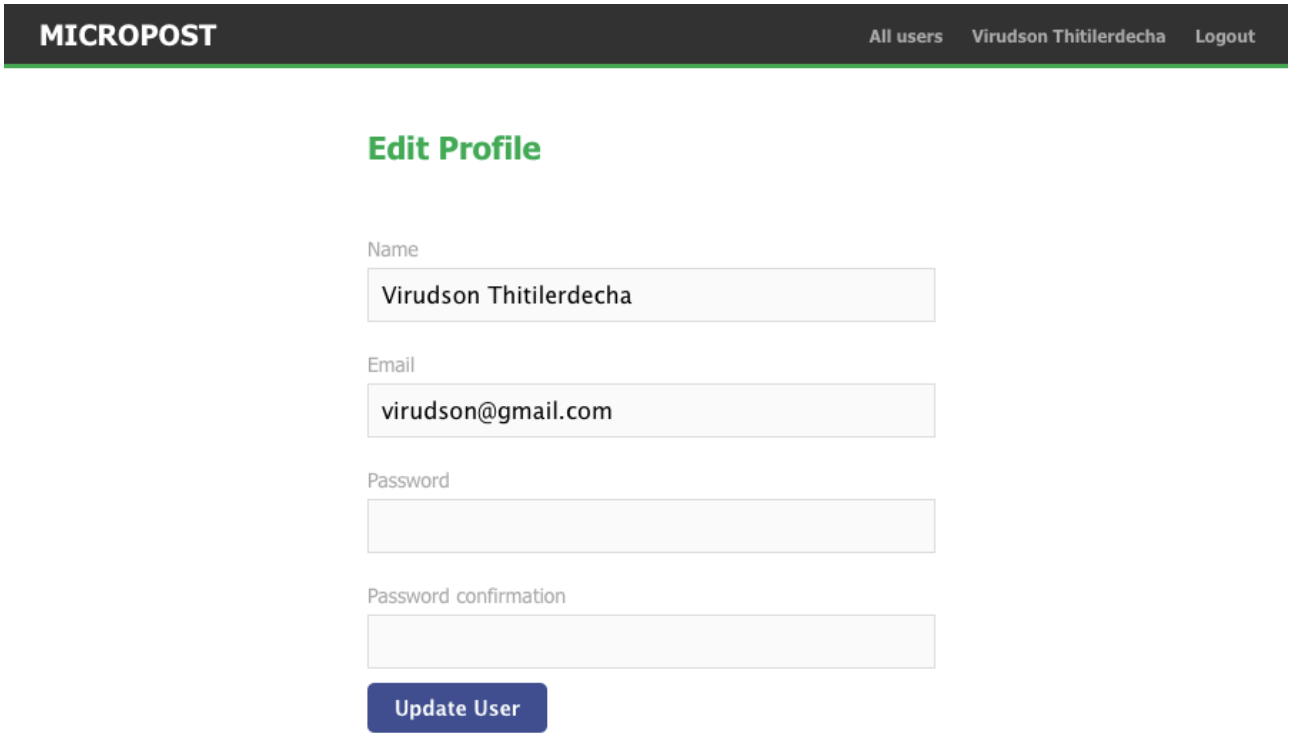
```
<% if @user == current_user %>
  <div class="edit-profile-button">
    <%= link_to "Edit profile", edit_user_url %>
  </div>
<% end %>
```

There is user page, If you are the logged in user in this page you will see edit profile button under your name.

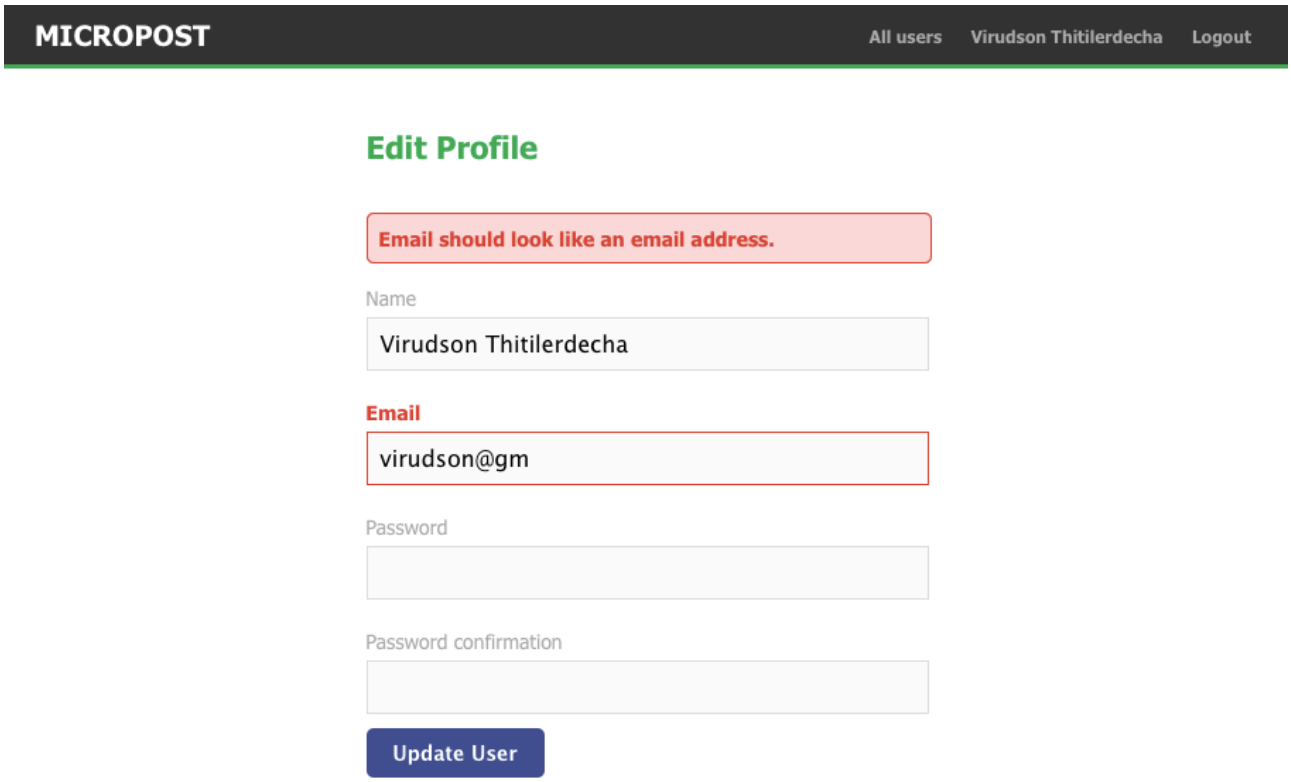


There is edit profile page. Let's try to change information.

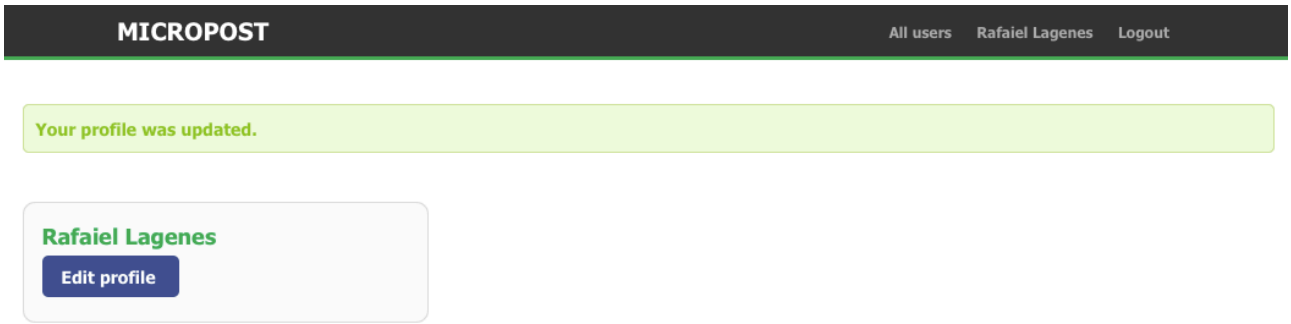
Notice: you can leave password filed except you want to change it.



In case you put some thing wrong in field this will show an error message.



If you done it right you will see your profile information has been changed.



Step 15 : Add filter to prohibit access if user was not logged in

- add check login method to **app/controllers/application_controller.rb**

```
class ApplicationController < ActionController::Base
  before_filter :require_login
  …
  …

  def require_login
    unless current_user
      flash[:error] = "Please login first."
      redirect_to login_url
    end
  end

  def require_not_login
    if current_user
      flash[:error] = "You're already logged in."
      redirect_to request.env["HTTP_REFERER"] || root_url
    end
  end

end
```

- add before_filter to **app/controllers/users_controller.rb**

```
class UsersController < ApplicationController
  skip_before_filter :require_login, :only => [:new, :create]
  before_filter :require_not_login, :only => [:new, :create]
  …
  …
end
```

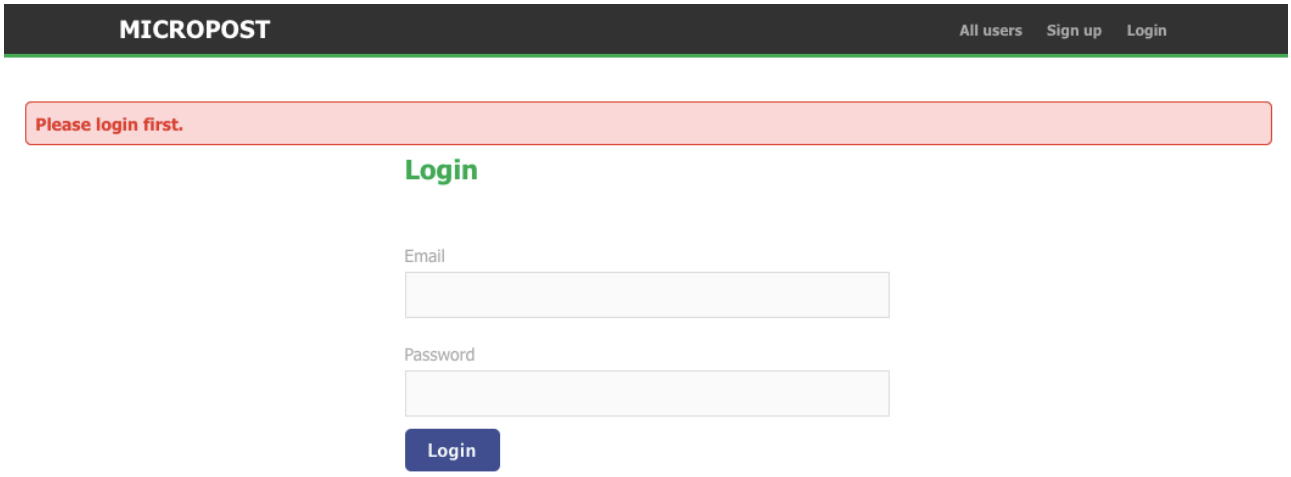
- add before_filter to **app/controllers/user_sessions_controller.rb**

```
UserSessionsController < ApplicationController
  skip_before_filter :require_login, :only => [:new, :create]
  before_filter :require_login, :only => [:destroy]
  before_filter :require_not_login, :only => [:new, :create]
  …
  …
end
```

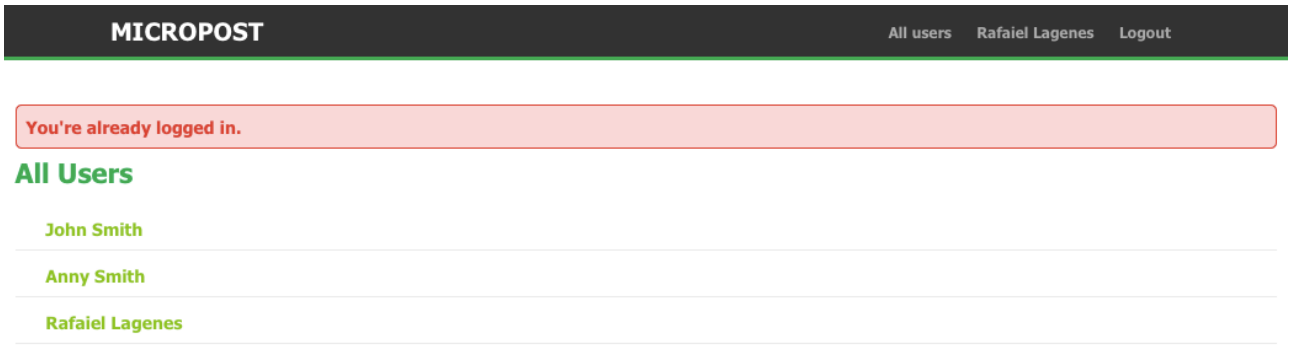
- add this code to **app/views/layouts/application.html.erb**

```
<div id="content">
  <% if flash[:notice] %>
      <div class="notice-message">
        <%= flash[:notice] %>
      </div>
  <% end %>

  <% if flash[:error] %>
    <div class="error-message">
      <%= flash[:error] %>
    </div>
  <% end %>

  <%= yield %>
</div>
```

Let's get try on our application. **When you not login** and try to edit you profile (via this link http://localhost:3000/edit_profile) or try to see other user's profile (via this link http://localhost:3000/users/1) the application will redirect you to login page and show error "Please login first.".



When you logged on try to access login page (via this link http://localhost:3000/login) error message will show up like this.



Step 16 : Create post form and page for display posts

- add action create and modify action index for provide user can create new post in **app/controllers/microposts_controller.rb**

```
class MicropostsController < ApplicationController
  def index
    @micropost = Micropost.new
  end

  def create
    @micropost = current_user.microposts.new(params[:micropost])

    if @micropost.save
      flash[:notice] = "Post successfully."
      redirect_to root_url
    else
      render :action => "index"
    end
  end
end
```

- edit home page to micropost#index in **config/routes.rb**

```
MicropostApp::Application.routes.draw do

  root :to => "microposts#index"
  …

end
```

- add validates and default scope to **app/models/microposts.rb**

```
class Micropost < ActiveRecord::Base
  belongs_to :user

  validates :user, :presence => true    # validate User should be exists
  validates :user_id, :presence => true # validate require user_id
  validates :content, :presence => true

  default_scope :order => "created_at DESC" # order post newer come first
end
```

- create form for create post to **app/views/microposts/index.html.rb**

```
<div class="user-information">
  <span class="user-name"><%= current_user.name %></span>

  <%= form_for @micropost do |f| %>
      <%= render :partial => "shared/form_error_messages",
                 :locals => {:model_object => @micropost} %>
      <%= f.text_area :content, :placeholder => "What's on your mind?" %>
      <%= f.submit "Post" %>
  <% end %>
</div>
```

- add code to get user feed to **app/models/user.rb**

```
class User < ActiveRecord::Base
…

  def feed
    microposts.reload # always get newest feed
  end
end
```

- create new partial for display each post in **app/views/microposts/_post_body.html.erb**

```
<div class="post-content"><%= post.content %></div>
<span class="post-time">
  posted <%= time_ago_in_words(post.created_at) %> ago.
</span>
```

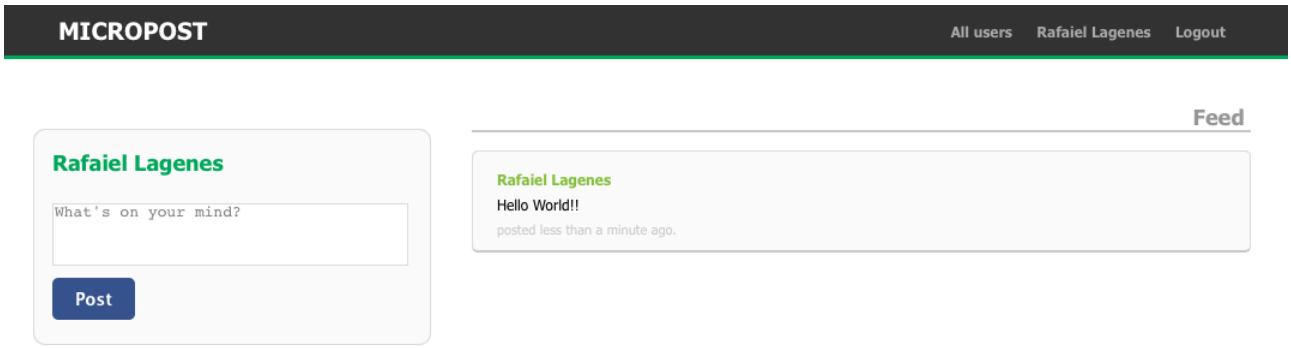
- create view for show all of logged in user's post to **app/views/microposts/index.html.erb**

```
<div class="user-information">
  …
</div>

<div class="user-post">
  <h2>Feed</h2>

  <% if current_user.feed.empty? %>
    <p class="none">Not have any post yet.</p>
  <% else %>
    <% current_user.feed.each do |post| %>
      <div class="post">
        <p class="post-owner">
          <%= link_to post.user.name, user_url(post.user_id) %>
        </p>
        <%= render :partial => "post_body", :locals => {:post => post} %>
      </div>
    <% end %>
  <% end %>
</div>
```

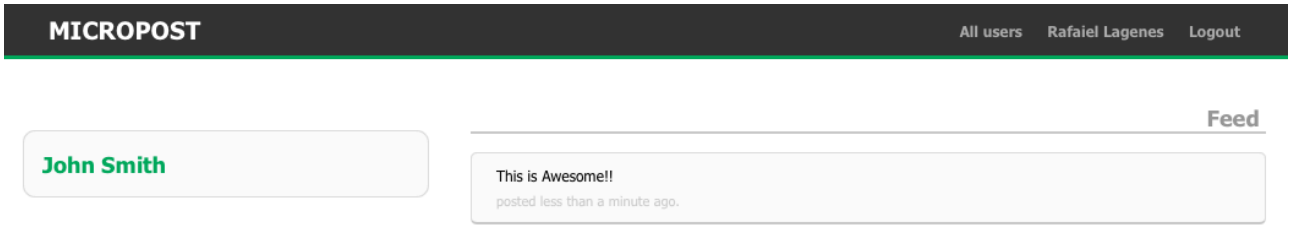
- create view for show all of other user's post to **app/views/users/show.html.erb**

```
<div class="user-information">
  …
</div>

<div class="user-post">
  <h2>Feed</h2>

  <% if @user.microposts.empty? %>
    <p class="none">Not have any post yet.</p>
  <% else %>
    <% @user.microposts.each do |post| %>
      <div class="post">
        <%= render :partial => "microposts/post_body",
                   :locals => {:post => post} %>
        </div>
    <% end %>
  <% end %>
</div>
```

Now you can post something in your mind. Let's post hello world into our application.



And you can see other's post in their page.



Step 17 : Add Delete Post Button

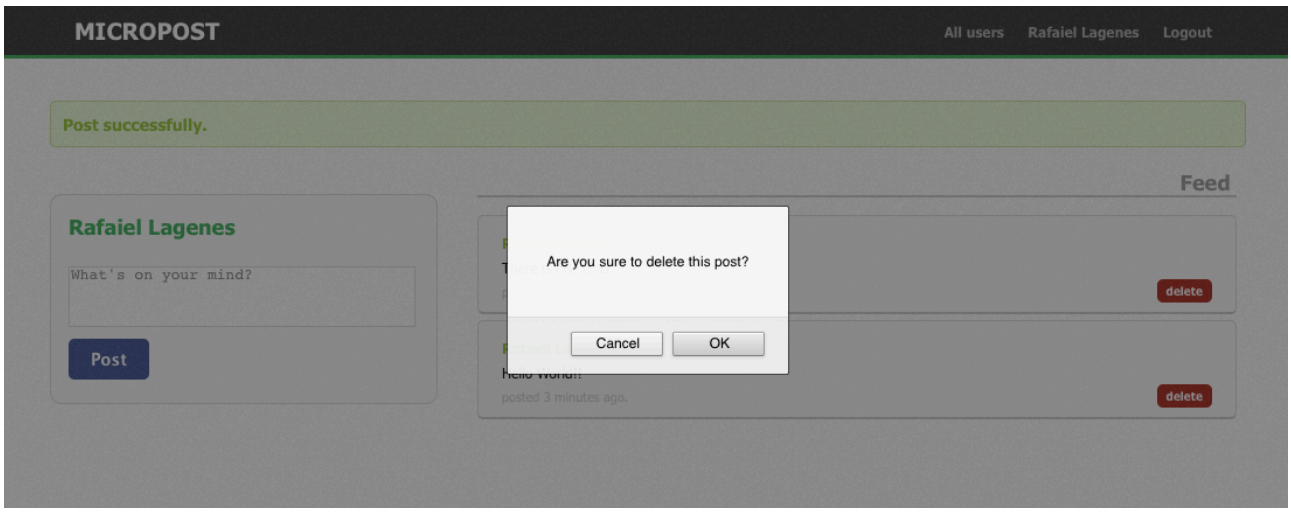
- create partial for render delete post button in **app/views/microposts/_post_body.html.erb**

```
<div class="post-content">&lt;%= post.content %&gt;</div>
<span class="post-time">
    posted <%= time_ago_in_words(post.created_at) %> ago.
</span>
<% if post.user_id == current_user.id %>
    <%= link_to "delete", post,
                :method => :delete,
                :confirm => "Are you sure to delete this post?",
                :class => "delete-button" %>
<% end %>
```

- In **app/controllers/microposts_controller.rb**

```
class MicropostsController < ApplicationController
  ...
  ...

  def destroy
    @micropost = Micropost.find(params[:id])
    @micropost.destroy
    redirect_to :back
  end
end
```

When you do want to remove you just click on delete button and select OK.



Step 18 : Add follow/unfollow and show stats of post

- add this code to **app/models/user.rb**

```
class User < ActiveRecord::Base
  …

  def following?(user)
    relationships.exists?(:following_id => user)
  end

  def follow(user_id)
    relationships.create(:following_id => user_id)
  end

  def unfollow(user_id)
    relationships.find_by_following_id(user_id).destroy
  end

end
```

- add follow/unfollow button to **app/views/users/show.html.erb**

```
<div class="user-information">
  <span class="user-name"><%= @user.name %></span>
  <% if @user == current_user %>
      <div class="edit-profile-button">
          <%= link_to "Edit profile", edit_profile_url %>
      </div>
    <% else %>
      <% if current_user.following?(@user) %>
          <div class="unfollow-button">
           <%= form_for current_user.relationships.find_by_following_id(@user),
                        :html => {:method => :delete} do |f| %>
               <%= f.hidden_field :following_id %>
               <%= f.submit "Unfollow" %>
             <% end %>
          </div>
      <% else %>
          <div class="follow-button">
            <%= form_for @relationship do |f| %>
                <%= f.hidden_field :following_id, :value => @user.id %>
                <%= f.submit "Follow" %>
            <% end %>
          </div>
      <% end %>
  <% end %>
</div>
```

- modify action show in **app/controllers/user_controller.rb**

```
class UsersController < ApplicationController
  …

  def show
    @user = User.find(params[:id])
    @relationship = Relationship.new
  end

  …

end
```

- create action foe make follower/following relation between each user and destroy for remove relation **app/controllers/relationships_controller.rb**

```
class RelationshipsController < ApplicationController

  def create
    current_user.follow(params[:relationship][:following_id])
    redirect_to user_url(params[:relationship][:following_id])
  end

  def destroy
    current_user.unfollow(params[:relationship][:following_id])
    redirect_to user_url(params[:relationship][:following_id])
  end

end
```

- update routing for follower/following action in **config/routes.rb**

```
MicropostApp::Application.routes.draw do

  root :to => "microposts#index"

  resources :users, :except => [:destroy] do
    member do
      get :follower, :following
    end
  end

  resources :microposts, :only => [:index, :create, :destroy]
  resources :relationships, :only => [:create, :destroy]
  resources :user_sessions, :only => [:new, :create, :destroy]

  …

end
```

- create user to **app/controllers/users_controller.rb**

```
class UsersController < ApplicationController
  …
  …

  def follower
    @user = User.find(params[:id])
    @followers = @user.follower_users
  end

  def following
    @user = User.find(params[:id])
    @followings = @user.following_users
  end

end
```

- create partial for show link to following page **app/views/users/follower.html.erb**

```
<h1><%= @user.name %> Followers</h1>

<% if @followers.empty? %>
    <p class="none">Not have any followers yet.</p>
<% else %>
    <% @followers.each do |user| %>
        <div class="user"><%= link_to user.name, user_url(user.id) %></div>
    <% end %>
<% end %>
```

- create partial for show link to following page **app/views/users/following.html.erb**

```
<h1><%= @user.name %> Followings</h1>

<% if @followings.empty? %>
    <p class="none">Not have any following users yet.</p>
<% else %>
    <% @followings.each do |user| %>
        <div class="user"><%= link_to user.name, user_url(user.id) %></div>
    <% end %>
<% end %>
```

- modified feed method in **app/models/user.rb** to display following user post in feed

```
class User < ActiveRecord::Base
  …

  def feed
    # get all post from following users included self post
    monitored_user_id = following_users.map(&:id) << self.id
    Micropost.where(:user_id => monitored_user_id)
  end

  …

end
```

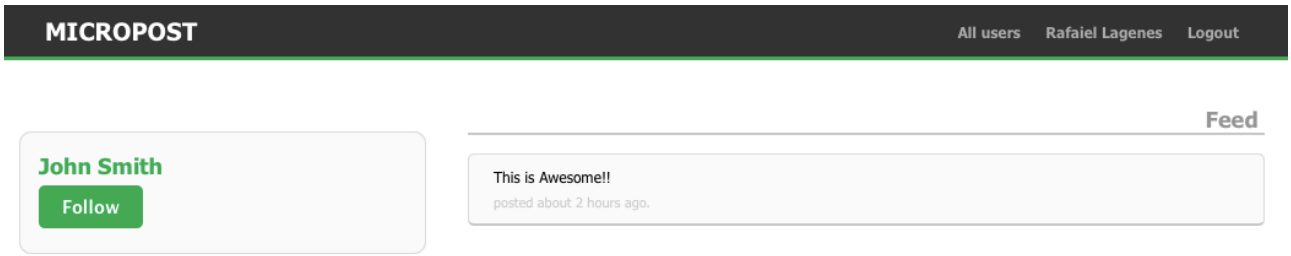
- create partial for show links follower/following and post count in **app/views/users/_stats.html.erb**

```
<div class="follower-followed">
  <span class="user-post-count first">
    <%= pluralize(user.microposts.count, "post") %>
  </span>
  <span class="followed-user">
    <%= link_to "#{user.following_users.count} following",
                following_user_url(user.id) %>
  </span>
  <span class="follower-user">
     <%= link_to "#{user.follower_users.count} follower",
                 follower_user_url(user.id) %>
  </span>
</div>
```

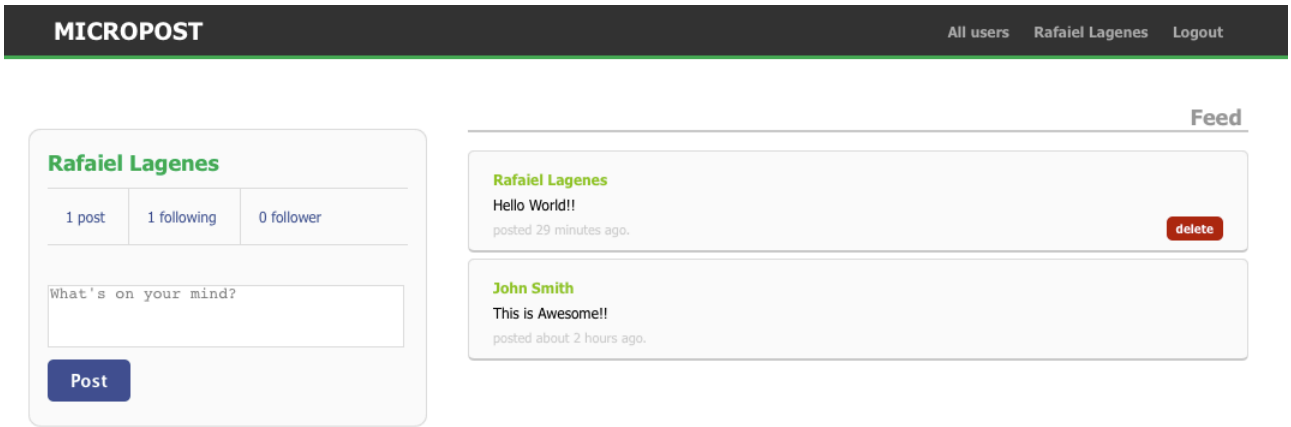
- render partial in **app/views/microposts/index.html.erb**

```
<div class="user-information">
  <span class="user-name"><%= current_user.name %></span>
  <%= render :partial => "users/stats", :locals => {:user => current_user} %>

  <%= form_for @micropost do |f| %>
      <%= render :partial => "shared/form_error_messages",
                 :locals => {:model_object => @micropost} %>
      <%= f.text_area :content, :placeholder => "What's on your mind?" %>
      <%= f.submit "Post" %>
  <% end %>
</div>
```

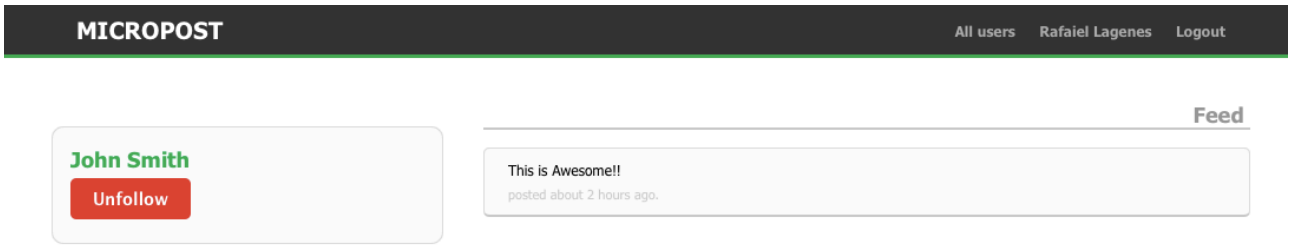
Now you can follow other user for see their post.



And you will see your stat on top of post box.



If you need to unfollow user yo can click button on his page.



And follower/following link will show all of people who following/follower.