Rails routing, kontrollerek
Gyakorlat

Kovács Gábor

2016. április 19.

A gyakorlat során az előző gyakorlatokon megkezdett feladat megoldását folytatjuk. A harmadik gyakorlaton kialakított menüben a felhasználóra vonatkozó útvonalak közösek, azokat testre kell szabnunk az egyes felhasználók tekintetében, vagyis a felasználóazonosítóknak szerepelniük kell a linkekben, ezért módosítjuk a `routes.rb`-ben a megfelelő bejegyzéseket. Az értintett linkek a felhasználói üzenőfal, a regisztráció és a profil szerkesztése nézetek, illetve az utóbbi kettőhöz tartozó eseménykezelők. Nevezzük el ezeket az útvonalakat a jobb karbantarthatóság végett.

```
  get 'users/:id', to: 'users#show', as: :feed

  get 'users/new', to: 'users#new', as: :new_user
  post 'users', to: 'users#create', as: :create_user

  get 'users/:id/edit', to: 'users#edit', as: :edit_user
  put 'users/:id', to: 'users#update', as: :update_user
```

Módosítsuk ennek megfelelően a bejelentkezett felhasználó menüjét. A menüben már az elnevezett útvonalhoz létrejött helper metódussal hivatkozunk az útvonalra az aktuális felhasználóval paraméterezve. Egyúttal módosítjuk az üdvözlő üzenetet is.

```
Welcome, <%= @user.name %>!

<fieldset>
  <legend>Menu</legend>
  <%= link_to "Feed", feed_path(@user) %><br/>
  <%= link_to "Profile", edit_user_path(@user) %><br/>
  <%= link_to "Logout", '/sessions/destroy' %>
</fieldset>
```

Az üzenőfalon meg akarjuk jeleníteni a felhasználó összes üzenetét. Új üzenet létrehozásakor az üzenet objektum `user` attribútuma az a felhasználó

lesz, akinek az oldalán épp tartózkodunk, vagyis az URI-ból derül ki az azonosítója. Az üzenet küldője az aktuálisan bejelentkezett felhasználó, akit a `session`-ből vehetünk elő. A kontrollerben inicializáljunk ennek megfelelően a példányváltozókat.

```
class UsersController < ApplicationController
  def show
    @post = Post.new
    @posts = @user.posts
    @u = User.find params[:id]
  end
end
```

Az üzenőfal, vagyis a `show` akció a felhasználók nézetében megjeleníti az összes a felhasználónak szánt üzenetet, és lehetőség van új üzenet beszúrására. Az egyes üzeneteket megjelenítő nézet a `posts` kontroller `show` nézete, az új üzenetet beszúró nézet pedig ugyanezen kontroller formja, ágyazzuk ezeket be a `show` nézetbe. Az üzenetek megjelenítéséhez iterálunk a `@posts` példányváltozóban lévő üzeneteken, majd megformázzuk az üzenet fejrészét, hogy az tartalmazza az üzenet küldőjét és létrehozásának időpontját, valamint magát az üzenetet.

```
<% @posts.each do |post| %>

  <fieldset>
    <legend><strong><%= post.author.name %> left a message at <%=
        l post.updated_at %></str
ong></legend>
    <%= post.text %>
  </fieldset>
<% end %>
```

Az üzenetet feltöltő formban, adjuk át rejtett mezőként a két felhasználó, a küldő és a címzett azonosítóit, valamint az üzenet szövegét.

```
<%= form_for(@post) do |f| %>
  <div class="field">
    <%= f.label :text %><br>
    <%= f.text_area :text, rows: 5, cols: 60 %>
  </div>
  <%= f.hidden_field :author_id, value: @user.id %>
  <%= f.hidden_field :user_id, value: @u.id %>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

Mivel üzenetből akár végtelenül sok lehet, az oldalunk vertikális dimenziója viszont véges, tördeljük az üzeneteket lapozhatóan több oldalra. Ehhez felveszünk egy új függőséget a `Gemfile`-ba:

```
gem 'will_paginate'
```

majd frissítjük a függőségeinket:

```
bundle update
```

A tördelés megvalósítása négy lépésből áll. A modell osztályban definiálunk egy függvény, ami az oldalszám paraméterhez előkeresi az adott oldalra eső töredéket a felhasználó összes, létrehozási idő tekintetében csökkenő sorrendbe rendezett bejegyzése közül. Egy oldalon 5 üzenetet engedélyezzünk.

```
  def get_posts_page(page)
      self.posts.order('updated_at␣desc').paginate(page: page,
          per_page: 5)
  end
```

A kontrollerben ezután nem a felhasználó összes üzenetét kérjük el az adatbázisból, hanem csak az aktuális töredéket.

```
class UsersController < ApplicationController
  def show
    ...
    @posts = @user.get_posts_page(params[:page])
    ...
  end
end
```

A nézethez pedig hozzáadjuk a lapozó linkeket. A linkek alapéertelmezés szerint a `page` paraméterhez rendelt értékben adják át a megjelenítendő töredés sorszámát.

```
<% @posts.each do |post| %>
  ...
<% end %>
<%= will_paginate @posts %>
```

Negyedik lépésként a linkünk szebb kinézete végett módosítjuk az útvonalat, hogy az opcionálisan tarlamazhassa a töredéket is:

```
  get 'users/:id(/:page)', to: 'users#show', as: :feed
```

Tegyük lehetővé, hogy ne csak szöveges üzenetek küldhessen egy felhasználó, hanem képet is. Ehhez fájlfeltöltést valósítunk meg, amihez szükségünk lesz egy új modellre, ami a feltöltött képekről tárol adatot. Egy csatolmányról megjegyezzük a MIME-típusát, az az üzenetet, amihez kapcsolódik, a

feltöltött fájl elérési útvonalát a fájlrendszeren, az eredeti nevét és méretét. A képeket a fájlrendszeren a `public` könyvtárban létrehozott `data` alkönyvtárban tároljuk, mert bináris adatot az adatbázis sem tud a fájlrendszernél hatékonyabban tárolni.

```
kovacsg@debian:~/gyakorlat/public# rails g model Attachment mime:
    string post:references filename:string original_name:string
    size:integer
```

Hajtsuk végre a migrációt, majd indítsuk újra a webszerverünket.

```
kovacsg@debian:~/gyakorlat/app/models# rake db:migrate
(in /home/kovacsg/gyakorlat)
== 20160419110214 CreateAttachments: migrating
   ===============================
-- create_table(:attachments)
   -> 0.0155s
== 20160419110214 CreateAttachments: migrated (0.0163s)
==========================
```

Egy üzenethez egy kép tartozhat, ezért felvesszük ezt az egy-egy relációt a két modellbe. A tulajdonos az üzenet modell.

```
 class Post < ActiveRecord::Base
  has_one :attachment
end
```

A birtokolt objektum a csatolmány.

```
class Attachment < ActiveRecord::Base
  belongs_to :post
end
```

A fájlfelöltéshez módosítanunk kell az üzenőfal nézetét, hogy az a szöveges mellett bináris adatok átküldésére is alkalmas legyen.

```
<%= form_for(@post, html: { multipart: true }) do |f| %>
  ...
  <div class="field">
    <%= f.label :attachment %><br>
    <%= f.file_field :attachment %>
  </div>
  ...
<% end %>
```

Ennek az eseménykezelője a `create` metódus a `posts` kontrollerben. Az üzenet sikeres mentésse esetén mentsük el az üzenethez asszociált csatolmányt is.

```
      ...
      if @post.save
```

```
        unless params[:post][:attachment].nil?
          Attachment.saveFile params[:post][:attachment], @post
        end
        ...
      else
      ...
```

A fájl mentését a csatolmány modell osztálymetódusában valósítjuk meg, aminek két paramétere lesz, a feltöltött temporális fájl és az asszociált üzenet. Először definiáljuk a fájl mentésének a helyét, és ellenőrizzük, hogy létezik-e az, ha nem, akkor létrehozzuk a könyvtárat. Létrehozunk egy új csatolmány példányt, és inicializáljuk annak mezőit a feltöltött temporális fájl attribútumai alapján. Eltároljuk az adatokat az adatbázisban, majd elmentjük a fájlt az adatbázisazonosítóval megegyező névvel a kijelölt helyre. Végül lekérdezzük a létrejött fájl méretét, és elmentjük azt az adatbázisba.

```
class Attachment < ActiveRecord::Base
  def self.saveFile(upload, post)
    dir = Rails.root.join("public", "data")
    unless File.exists? dir
      Dir.mkdir dir
    end
    fname = upload.original_filename
    #fname.gsub!(/^\w\._/,"_")
    a = Attachment.new
    a.filename = path
    a.mime = upload.content_type
    a.post = post
    a.original_name = fname
    a.save
    path = File.join(dir, a.id)
    File.open(path, "wb") do |f| f.write(upload.read) end
    a.size = File.size(path)
    a.save
  end
end
```

Jelenítsük meg a feltöltött képet az üzenőfalon, ha az adott üzenethez tartozik feltöltött kép is. A képet beágyazzuk, ezért azt `<img>`-ként jelenítjük meg, aminek átadjuk az elérhetőségét és adatbázisbeli azonosítóját.

```
    <% unless post.attachment.nil? %>
      <img src="<%= download_path(post.attachment.id)%>"></img>
    <% end %>
```

A kép URL-jének definiálnunk kell egy új útvonalat:

```
get 'download/:id', to: 'post#download', as: :download
```

Ennek az eseménykezelője a `posts` kontrollerben `download` néven definiálandó metódus, ami az adatbázisból előkeresi a kérés paraméterének megfelelő csatolmányt, és visszaküldi a fájrendszerről az alapján előkeresett fájlt válaszként.

```
  def download
    a = Attachment.find params[:id]
    send_file a.filename, type: a.mime, disposition: :inline
  end
```

Végül az útvonalakat tartalmazó `config/routes.rb` fájlban állítsuk át a webszerverünk gyökerét a hello nézetre.

```
root :to => "say#hello"
```

Ami hátramaradt a következő gyakorlatra az a felhasználók keresése és kölcsönös barátnak jelölése.