Rails routing, kontrollerek
Gyakorlat

Kovács Gábor

2013. november 13.

A gyakorlat során az előző gyakorlatokon megkezdett feladat megoldását folytatjuk. Legelőször az útvonalakat tartalmazó `config/routes.rb` fájlban állítsuk át a webszerverünk gyökerét az események nézet `index` oldalára.

```
root :to => "events#index"
```

Ezután tegyük rendbe az események nézeteit. A kezdőoldalon jelenleg még látszanak a vendégfelhasználók számára is a scaffolddal létrehozott táblázatban a mutat, szerkeszt és töröl linkek. Tegyük ezeket csak bejelentkezett felhasználó számára elérhetővé megjegyezve magunknak, hogy ezt még később pontosítanunk kell, és a szerkesztés valamint a törlés linkek csak a bejelentkezett adminisztátor jogosultsággal bíró felhasználók legyenek láthatók.

Ugyanezen az alapon módosítsuk a fogadások index oldalát is azzal a különbséggel, hogy egy fogadás adata akkor módosítható, ha a bejelentkezett felhasználó megegyezik a fogadás tulajdonosával. A bejelentkezett felhasználó azonosításához hívjuk segítségül a `before_filter` nevű callback metódust, ami minden kontrollerakció előtt meghívandó függvény azonosítójával paraméterezhető. A meghívott függvény egy `@user` nevű példányváltozót hoz létre a session alapján, és hogy a külvilágból ne legyen ez e metódus hozzáférhető, a láthatóságát `private`-re állítjuk.

```
class BidsController < ApplicationController
  before_filter :find_user
  private
  def find_user
    if session[:user]
      @user = User.find session[:user]
    end
  end
```

```
end
```

A kontrollerek (események, fogadások) `index` nézetén rejtsük el a szerkesztés, törlés és új létrehozása linkeket. Ezt a táblázat fejrészében, törzsében és a táblázat alatti link körül elhelyetett szűrőfeltétellel érhetjük el, a fogadások táblázata az alábbi kódrészletben szereplő lehet. A dátumot az alapértelmezett formázás helyett saját formátumban jelenítjük meg a dátum típus `strftime` metódusával

```
<% if @user then if session[:user] = @user.id %>
    <td><%= link_to bid.bidtime.strftime("%d/%m/%Y %H:%
       M"), "/events/show/#{bid.event.id}"
    <td><%= link_to 'Show', bid %></td>
    <td><%= link_to 'Edit', edit_bid_path(bid) %></td>
    <td><%= link_to 'Destroy', bid, method: :delete,
       data: { confirm: 'Are you sure?' } %></td>
<% end end %>
```

Mivel mind eseményből, mind fogadásból nagyságrendekkel több lehet, mint amennyit a nézeten meg tudunk jeleníteni, ezért e tartalmak tekintetében megvalósítjuk a tördelés funkciót. A tördeléshez a `will_paginate` Ruby függvénykönyvtárt használjuk, a `Gemfile`-ba beszúrjuk az alábbi sort, majd kiadjuk a `bundle update` parancsot:

```
gem 'will_paginate''
```

A tördeléshez három forrást kell módosítanunk. A modell osztályokban létrehozunk egy-egy metódust, amelyek a paraméterként átadott oldalszámhoz rendelik az adatbázisbeli rekordok részhalmazát valamely idő típusú attribútum szerint csökkenő sorrendbe rendezve egy oldalon öt rekordot megjelenítve. A tördeléshez a frissen telepített gem `paginate` vagy a `page ActiveRecord` kiegészítését használjuk.

```
class Event < ActiveRecord::Base
  def self.get_events_page(page)
    Event.order('eventtime desc').paginate(:page=>page,
        :per_page=>5)
  end
end
class Bid < ActiveRecord::Base
  def self.get_bids_page(page)
    Bid.order('updated_at desc').paginate(:page=>page,
       :per_page=>5)
  end
```

```
end
```

Mivel a tördelés paraméterére nincs illeszkedő útvonal szabályunk, ezért az az URL-ben megjelenik. Ezt elkerülendő vegyünk fel egy új szabályt a `routes.rb`-be:

```
match 'events(/:page)(.:format)' => 'events#index'
```

A megfelelő kontrollerben e függvényeket használjuk az összes rekord lekérését megvalósító `all` metódus helyett. A fogadások kontrollerében például ez így nézhet ki:

```
def index
  #@bids = Bid.all
  @bids = Bid.get_bids_page(params[:page])
  ...
end
```

A megfelelő `index` nézetet táblázatai felé és alá mindössze egyetlen sort kell beszúrnunk, ami hozzáadja a lapozást megvalósító linkeket. A második paraméterként megadható `param_name` opcióval felüldefiniáljuk az alapértelmezett `page` kérés paraméternevet. A fogadások nézetében ez így nézhet ki:

```
<%= will_paginate @bids %>
```

Viszonylag gyakori feladat fájlfeltöltés és -letöltés megvalósítása. Ezt illusztrálandó módosítjuk a felhasználókról tárolt adatokat felvéve azok közé a profilképet. Hozzunk létre egy migrációt, majd hajtsuk azt végre, és a modell osztályban módosítsuk a hozzáférhető attribútumok listáját. Az adatbázisban csak a fájl nevét tároljuk, magát az állományt a webszerver könyvtárában (`public`) létrehozott alkönyvtárban helyezzük el.

```
rails generate migration addProfilePictureToUsers
   profile_picture:string
rake db:migrate
```

A profilkép egy fájl, bináris adat, amit a felhasználó a saját profil oldalán tölthet fel. Vegyünk fel egy új, a fájlfeltöltés megvalósító formot a felhasználók kontrollerének `edit` nézetén. Az új formunk vonatkozzék az `upload` nevű hash-re, és az eseménykezelője legyen az `upload` akció. A fájlfeltöltéshez a formhoz hozzá kell rendelnünk a `multipart` opciót. A nyomógomb feliratának előállításakor felhasználjuk a `ActiveSupport` modul `humanize` metódusát.

```
<fieldset>
  <legend>Upload profile picture</legend>
  <%= form_for :upload, :url => {:action => 'upload', :
      id=>session[:user] }, :multipart => true do |form
      |%>$
    <%= form.label :file %>
    <%= form.file_field :file %>
    <%= form.submit "upload_file".humanize %>
  <% end %>
</fieldset>
```

A profilkép weboldalba ágyazása egyszerű feladat, ezért csavarjuk meg annyival, hogy megjelenítés helyett letölthetővé tesszük azt a felhasználó profilját megtekintők számára, vagyis a `show` nézeten. A letöltést egy linkkel valósítjuk meg, ami a `download` útvonalon keresztül lehetséges.

```
<p>Download profile picture</p>
<%= link_to "Profile␣picture", "/downloads/#{@user.id}"
    %><br/>
```

A profilkép feltöltését a felhasználók kontrollerének `upload` akciója végzi el, ami először ellenőrzi, hogy a felhasználó be van-e jelentkezve, majd megvizsgálja, hogy a feltöltési akcióhoz tartozik-e feltöltött állomány. Ha mindkét feltétel teljesül, akkor a fájlt a `User` modell osztályban definiálandó `save_file` metódussal eltároljuk. Az egyik feltétel nem teljesül, flash üzenet segítségével tudatjuk a felhasználóval a hiba okát. Végül mivel a feltöltéshez nem tartozik külön nézet átirányítjuk a kérést az aktuális teendő `show` akciójára. Itt használhattuk volna a `render` metódust, azonban akkor helytelen URL jelenne meg a böngésző címsorában.

```
def upload
  if session[:user]
    if !params[:upload].nil?
      @user.save_file params[:upload]
      flash[:reg]  = "File␣uploaded␣successfully"
      redirect_to :action => :show
    else
      flash[:reg] = "Empty␣file␣field"
      redirect_to :action => :edit
    end
  else
    flash[:reg] = "Please␣log␣in"
```

```
    redirect_to :action => :edit
  end
end
```

A fájl elmentéséhez a HTTP kérés `upload` paraméterét adjuk át. A feltöltött fájlokat tároljuk a Rails alkalmazás `public` könyvára alatt egy `profilepictures` könyvárban. A `public` és a `profilepictures` tagok statikusak az elérési úton, azonban a teendő azonosítójának megfelelő könyvtár létezését minden kérés során ellenőriznünk kell, és amennyiben nem lézezik, létre kell hoznunk. A kövezkező lépés a feltöltött állomány nevének kinyerése a HTTP kérés paraméterei közül és a weboldal elleni támadásra alkalmas karakterek lecserélése a fájlnévben: az üres és `.` és `_` kivételével az összes speciális karaktert cseréljük le aláhúzásjelre. Végül fájlművelettel elmentjük az adatokat, és létrehozzuk az adatbázisbeli rekordot. Az esetleges fájlnév-egyezéseket elkerülendő hosszabb távon érdemes úgy átalakítani az alkalmazásunkat, hogy az a képeket a felhasználó azonosítójávan címkézett néven tárolja a fájlrendszeren.

```
def save_file(upload)
  dir = Rails.root.join("public","profilepictures")
  if !File.exists? dir
    Dir.mkdir dir
  end
  fname = File.basename(upload[:file].original_filename
    )
  fname.gsub!(/^\w\.\_/,"_")
  path = File.join(dir,fname)
  File.open(path,"w") { |f| f.write(upload[:file].read)
    }
  self.profile_picture = fname
  save
end
```

A `show` nézet letöltés linkje egy jelenleg nem létező kontrollerre mutat, ezért módosítanunk kell az útvonalak tábláját a `routes.rb`-ban. Rendeljük a `/download` útvonal prefixhez az `users` kontroller `download` akcióját, az útvonal következtető két darabjából pedig képezzünk HTTP kérés paramétert, az első legyen a teendő azonosítója, a második a csatolmány azonosítója. A `upload` akcióra is definiáljunk egy saját útvonalat elfedve ezzel a `id` paramétert.

```
match 'downloads/:id' => 'users#download'
```

A letöltés akció a kérés paraméterei alapján kikeresi a fájlrendszerről adatbázisban tárolt nevű fájlt, ha van ilyen, és csatolmányként elküldi a felhasználónak.

```
def download
  send_file("public/profilepictures/#{@user.
      profile_picture}", :disposition => :attachment.
      to_s)
end
```