Rails routing, kontrollerek
Gyakorlat

Kovács Gábor

2013. április 16.

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

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

Ezután tegyük rendbe a feladatok nézeteit. A felhasználók alapján különbséget kellene tennünk aszerint, hogy a felhasználó oktató, aki a feladatkiírást elkészíti, hallgató, aki a feladatkiírást megnézi, és megoldást ad be rá, vagy adminisztrátor, aki nem készíthet új feladatot, és nem adhat be megoldást. Ennek modellezésére vegyünk fel egy új attribútumot `tipus` neven a `User` modellunkbe es a kapcsolodo `users` táblába az alábbi migrációval:

```
rails generate migration AddTipusToUsers student:
  integer
```

A `User` modellosztályban ezután tegyük a `student` attribútumot hozzáférhetővé az `attr_accessible` metódus paraméterlistájának kibővítésével. Majd módosítsuk az adatbázisunkban lébő objektumokat úgy, hogy az egyik felhasználó az `1` értékkel reprezentált oktató, a másik a `2` értékkel reprezentált hallgató.

```
u = User.find :all
u.each do |user| user.student=2; user.save end
u = find_by_neptun('oweyoa')
u.student = 1
u.save
```

Ahhoz, hogy a nézeteket szeparálni tudjuk a felhasználó típusa szerint, szükségünk van az aktuálisan bejelentkezett felhasználóra példányváltozóként. Mivel ezt a `tasks_controller` minden akciójában meg kellene valósí-

tanunk, e megoldás helyett használjuk 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 TasksController < ApplicationController
  before_filter :find_user
  private
  def find_user
    if session[:user]
      @user = User.find session[:user]
    end
  end
end
```

A felhasználó típusának ellenőrzése valószínűleg sok helyen előkerül majd a nézetet, a jobb karbantarthatóság végett ezért a típus megállapítására készítsünk helper metódusokat. Ha meggondoljuk magunkat a `student` attribútum értékével kapcsolatban, akkor csak egy helyen kell módosítanunk a kódot.

```
module TasksHelper
  def is_student?
    @user && @user.student==2
  end
  def is_lecturer?
    @user && @user.student==1
  end
end
```

Az feladatok kontroller `index` nézetén rejtsük el a szerkesztés, törlés és új létrehozása linkeket a nem oktató típusú felhasználók elől. Ezt a táblázat fejrészében, törzsében és a táblázat alatti link körül elhelyetett szűrőfeltétellel érhetjük el, például

```
<% if is_lecturer? %>
<%= link_to 'New Task', new_task_path %>
<% end %>
```

A következő lépés a `show` nézet rendbetétele. Az oktató ilyenkor az összes a feladatra született megoldást láthatja, és letöltheti azokat. A hallgató típusú felhasználó pedig a saját megoldását nézheti meg, illetve feltölthet egy új megoldást.

A megoldások listáját a feladatok kontroller `show` akciójában állítjuk be:

```
def show
  @task = Task.find(:params[:id])
  if @user && @user.student == 2
    @submission = Submission.
        find_by_user_id_and_task_id @user.id, @task.id
  elsif @user && @user.student ==1
    @submissions = Submission.find :all
  end
end
```

Majd a nézetek lehetővé tesszük ezek listázását, és letöltését egy linkkel, ami a `download` útvonalon keresztül lehetséges.

```
<% if is_lecturer? %>
  <ul>
  <% for s in @submissions %>
    <li><%= link_to s.user.neptun,
      "/download/#{s.task.id}/#{s.user.id}" %></li>
  <% end %>
  </ul>
<% elsif is_student? %>
  <% if @submission %>
    <%= link_to 'Download', "/download/#{@task.id}/#{
        @user.id}" %>
  <% end %>
<% end %>
```

A hallgató típusú felhasználók számára a letöltés link alatt hozzuk létre a feltöltést lehetővé tévő formot, amelynek eseményét a még nem létező `upload` akció kezeli le, `id` nevű paraméterként átadjuk a feladat azonosítóját, és `uid` nevű paraméterként átadjuk a felhasználó azonosítóját. Fájlfeltöltés esetén a `multipart` HTML opciót be kell állítanunk. A nyomógomb feliratának előállításakor felhasználjuk a `ActiveSupport` modul `humanize` metódusát. Mindezek mellett jelenítsük meg a fájlfeltöltéssel kapcsolatos feldolgozás során generált flash üzeneteket.

```
 <%= flash[:file] %>
<fieldset>
  <legend>Upload a solution</legend>
  <%= form_for :upload, :url=>{:action=>'upload', :id=>
      @task.id, :uid=>@user.id},
```

```
        :html=>{:multipart => true} do |form|%>
    <%= form.label :file %>
    <%= form.file_field 'file' %>
    <%= form.submit 'upload'.humanize %>
  <% end %>
</fieldset>
```

A fájlfeltöltést a feladatok 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 `Submission` modell osztályban definiálandó `save_attachment` osztálymetódussal eltároljuk a fájlrendszeren. 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].nil?
    if !params[:upload].nil?
      Submission.save_file params[:uid], params[:id],
          params[:upload]
      flash[:file] = "File uploaded successfully"
    else
      flash[:file] = "Empty file field"
    end
  else
    flash[:file] = "Please log in to upload file"
  end
  @task = Task.find params[:id]
  redirect_to "/tasks/#{@task.id}"
end
```

A fájl elmentéséhez három paraméterre van szükségünk. A létrehozó felhasználó azonosítójára, a feladat azonosítójára, melyhez a csatolmány tartozik és magára a csatolmányra, amit a HTTP kérés paraméterei között találhatunk meg. A feltöltött fájlokat tároljuk a Rails alkalmazás `public` könyvára alatt egy `data` könyvárban. Az esetleges fájlnévegyezéseket elkerülendő a megoldásokat a feladat sorszáma és a hallgató neptun kódja szerint alkönyvtárakra szétválogatjuk. A `public` és a `data` 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: a soreleji üres karaktereket és . karaktereket cseréljük le aláhúzásjelre. Végül fájlművelettel elmentjük az adatokat, és létrehozzuk az adatbázisbeli rekordot.

```
def self.save_file(user,task,upload)
  dir = Rails.root.join("public","data")
  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)
      }
  s = Submission.new
  s.file = fname
  s.mime = upload['file'].content_type
  s.task_id = task
  s.user_id = user
  s.save
end
```

A feladatok `show` nézetében lehetővé tettük a korábban feltöltött csatolmányok letöltését, az akciót azonban egy jelenleg nem létező kontrollerre mutat, ezért módosítanunk kell az útvonalak tábláját a `routes.rb`-ban. Rendeljük a `/download` útvonal prefixhez az `tasks` kontroller `download` akcióját, az útvonal következtő 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 `uid` paramétert.

```
match '/download/:id/:uid' => 'tasks#download'
match '/tasks/upload/:id/:uid => 'tasks#upload'
```

A letöltés akció a kérés paraméterei alapján kikeresi a fájlrendszerről és az adatbázisból a HTTP kérés paramétereinek megfelelő rekordot és fájlt, ha van ilyen, és csatolmányként elküldi a felhasználónak. Végül átirányítja a kérést az aktuális teendő `show` akciójára.

Egy feladathoz sok megoldás tartozhat. Ha ezek száma egy határon túlnyúlik, akkor az oldaluk magassága kezelhetetlen lehet, ezért e tartalmak

tekintetében megvalósítjuk a tördelés funkciót az oktató típusú felhasználók számára. A tördeléshez a `will_paginate` gemet használjuk, a `Gemfile`-ba beszúrjuk az alábbi sort, majd kiadjuk a `bundle install` parancsot:

```
gem 'will_paginate', '~> 3.0.0'
```

A tördeléshez három forrást kell módosítanunk. A megoldások modell osztályájában létrehozunk egy metódust, amelyek a paraméterként átadott oldalszámhoz rendelik az adatbázisbeli megoldások részhalmazát időben csökkenő sorrendben. Megoldásból egy oldalon egyet jelenítsünk csak meg. A tördeléshez a frissen telepített gem `paginate` vagy a `page ActiveRecord` kiegészítését használjuk.

```
class Submission < ActiveRecord::Base
  def self.get_solutions_page(page, task)
    Task.find(task).submissions.paginate(:page=>page,:
        per_page=>1)
  end
end
```

A feladatok kontroller `show` akciójában módosítsuk a `@submissions` példányváltozó értékét az oktató típusú felhasználó esetén, ne az összes rekordot jelenítsük meg, hanem csak a HTTP paraméterben megkapott oldalszámhoz tartozót.

```
def show
  if @user && @user.student == 2
  elsif @user && @user.student == 1
    @submissions = Submission.get_solutions_page params
        [:page], @task.id
  end
end
```

A `show` nézetbe mindössze egyetlen sort kell beszúrnunk, ami hozzáadja a lapozást megvalósító linkeket. A második paraméterként megadható `param_name` opcióval felüldefiniáljuk az alapértelmezett `page` kérés paraméternevet.

```
<%= will_paginate @submissions %>
```