Rails routing, kontrollerek
Gyakorlat

Kovács Gábor

2015. április 14.

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"
```

Az előző gyakorlaton felvettük a felhasználó típusát jelölő `t` attribútumot, azonban azt nem definiáltuk. Legyen a `0` érték az adminisztrátoré, az `1` érték az oktatóé, és a `2` érték a hallgatóé. Módosítsuk ez alapján az `is_student?` helper metódusunkat!

```
def is_student?
    logged_in? && User.find(session[:user]).t == 2
end
```

A befelületen egyelőre még nem az aktuális felhasználót üdvözli a rendszer, ezért definiáljunk erre is egy helpert, ami a bejelentkezett felhasználó nevét adja vissza vagy azt, hogy *Guest*.

```
  def username
    if logged_in?
      User.find(session[:user]).name
    else
      "Guest"
    end
  end
```

A menüben használjuk ezt a helpert a felhasználó üdvözlésére.

```
Hello, <%= username %>!
```

A megoldások modellel a feladatok `index` nézetének táblázatában felesleges oszlopok maradtak benne, javítsuk ezt a hibát, távolítsuk el az `is_student?` feltételes oszlopokat. A feladatok megoldását ehelyett a feladat `show` nézetén tegyük lehetővé egy fájlválasztó elemet tartalmazó form segítségével.

Ez csakis hallgató típusú felhasználóknak jelenjen meg. A feltöltéshez meg kell adnunk a feladat azonosítóját, a feltöltő mindig az aktuális felhasználó lesz. Fájlfeltöltéshez meg kell adnunk a `multipart` opciót.

```
<% if is_student? %>
<fieldset>
  <legend>Upload solution</legend>
  <%= form_for :upload,
    :url => "/upload/#{@task.id}",
    :html => {:multipart => true} do |form| %>
    <%= form.label :file %>
    <%= form.file_field :file %>
    <%= form.submit "Upload" %>
  <% end %>
</fieldset>
<% end %>
```

Az útvonal a `/upload/`, vegyük fel a `routes.rb`-be. A fájlfeltöltés egy HTTP POST művelet, és a feladatok kontrollerének `upload` metódusa kezeli le. Egy paramétere van, a feladat azonosítója. Egyúttal vegyük fel a letöltés útvonalat is. Annak két paramétere lesz: a feladat azonosítója és a felhasználó azonosítója. A letöltés kérése HTTP GET művelet lesz.

```
post 'upload/:id', :to => 'tasks#upload'
get 'download/:id/:uid', :to => 'tasks#download'
```

A megoldás feltöltéséhez szükséges migrációkaz az előző alkalommal már létrehoztuk: tároljuk a fájl nevét és MIME típusát. Ezen kívül érdemes lehet még tárolni a fájl méretét és utolsó frissítésének időpontját. A fájlt magát a webszerver könyvtárában (`public`) létrehozott alkönyvtárban helyezzük el.

Az eseménykezelőket a feladatok kontrollerében valósítjuk meg. A feltöltést az `upload` akció kezeli, ami először ellenőrzi, hogy van-e bejelentkezett felhasználó, vagy valaki csak úgy próbálkozik, illetve, hogy létezik-e a feltöltés kéréshez csatolt fájl. Ha mindkettőre igen a válasz, akkor a `Submission` modellen keresztül elmentjük a csatolmányt, majd az előző nézetre küldjük a felhasználót. Ha bármelyik feltétel nem teljesük, akkor a felhasználó illegális műveletet próbált végrehajtani, ezért visszairányítjuk az előző oldalra.

```
class TasksController < ApplicationController
  def upload
    if !session[:user].nil?
      if !params[:upload].nil?
        Submission.save_file(session[:user], params[:id
            ], params[:upload])
        flash[:notice] = "File uploaded successfully"
      else
        flash[:notice] = "Empty file field"
      end
    else
      flash[:notice] = "Please, log in to submit
          solution"
    end
    redirect_to :back
  end
end
```

A fájl elmentése a `Submission` modell `save_file` osztálymetódusában történik. Paraméterként át kell adnunk magát a feltöltött fájlt, a feladat azonosítóját, amire a megjegyzés vonatkozik, és a felhasználó azonosítóját, aki a megjegyzést tette. Az utóbbi két paraméter a modell példány idegen kulcsai, és a komment `id` paraméteréből, illetve a `session` hashből jönnek. A metódus törzsében először létrehozzuk azt az útvonalat, ahova a kéréshez csatolt állomány el fogjuk tárolni. Ez a Rails alkalmazásunk `public` könyvtárában létrehozandó `data` alkönyvtár lesz. Először megnézzük, hogy létezik-e ez, és ha nem, akkor létrehozzuk. A létrehozandó fájl nevét e kérés `original_filename` mezőjéből vehetjük elő, amit összefűzve az előző útvonalszakasszal megkapjuk a kiírandó fájl abszolút útvonalát a fájlrendszeren. Ezután megnyitjuk a fájlt, és a feltöltött csatolmány tartalmát bájtról bájtra átmásoljuk, a fájl ennek hatására létrejön, és a metaadatait elmenthetjük az adatbázisba.

```
class Submission < ActiveRecord::Base
  def self.save_file(userid, taskid, upload)
    dir = Rails.root.join("public","data", taskid.to_s)
    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.filename = fname
    s.mime = upload[:file].content_type
    s.user_id = userid
    s.task_id = taskid
    t = Task.find taskid
    s.late = t.deadline < Time.now
    s.save
  end
end
```

Letöltéskor előkeressük a kérés feladat- és felhasználóazonosító paraméterei által kijelölt megoldás objektumot, előállítjuk az ahhoz kapcsolódó fájl elérési útját a fájlrendszeren, majd a `send_file` Rails helperrel csatolmányként elküldjük válaszként.

```
  def download
    @u = params[:uid]
    @t = params[:id]
    s = Submission.where(task_id:@t, user_id:@u).first
    file = Rails.root.join("public","data",@t,s.
        filename)
    send_file(file, :type=>s.mime, :disposition => :
        attachment)
  end
```

Mivel a megoldások nagyságrendekkel több lehet, mint amennyit a nézeten áttekinthetően meg tudunk jeleníteni egymás alatt, 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'
```

Három módosítást kell a tördelésre elvégeznünk. A nézethez hozzá kell adnunk a lapozó linkeket, a kontrollerben csak az aktuális oldalon megjelenítendő töredékeket kérjük el a modelltől. A modellben pedig definiáljuk a tördelés metódusát.

A feladatok `show` nézetében az oktató típusú felhasználóknak tördelve megjelenítjük a megoldásokat. A fájlt klikkelhető linkkel tesszük elérhetővé,

a linkbe belefűzzük a megoldás asszociációi alapján a feladat és a felhasználó azonosítóját. A navigációs linkeket a `will_paginate` helper helyezi ki az oldalra.

```
<% elsif is_admin? %>
  <%= @solutions.each do |s| %>
  <%= link_to s.filename, "/download/"+s.task.id.to_s+"
      /"+s.user.id.to_s %><br/>
  <% end %>
  <%= will_paginate @solutions %>
<% end %>
```

A `show` akcióban a nézet számára rendelkezésre bocsátjuk a `page` paramétertnek megfelelő megoldáshalmazt a `solutions` példányváltozón keresztül.

```
@solutions = Submission.get_submission_page(@task.id,
   params[:page])
```

Csak két megoldásunk van az adatbázisban, ezért egy oldalon csak egy megoldást jelenítünk meg. A tördelt adatok lekérdezését a modellben valósítjuk meg. Az oldalon megjelenítendő rekordok számát a `:per_page` értéke, az oldalt a `:page` értéke adja meg.

```
class Submission < ActiveRecord::Base
  def self.get_submission_page(id,page)
    Submission.where(task_id:id).paginate(:page=>page,
       :per_page=>1)
  end
end
```