Rails routing, kontrollerek
Gyakorlat

Kovács Gábor

2014. április 15.

A gyakorlat során az előző gyakorlatokon megkezdett feladat megoldását folytatjuk. Azonban először távolítsuk el a harmadik gyakorlaton a teendők kontrollerében elhelyezett statikus értékeket, és távolítsuk el a hozzáadott kommentszimbólumot a `set_task` metódus törzséből.

Az előző gyakorlaton a webfelületen való jelszómódosítás nem úgy működött, ahogy azt szerettük volna. Ennek oka, hogy Rails 4-től a HTTP kérésben szereplő paraméterekre vonatkozó szűrés lépett életbe, és az átengedett paraméterek között nem szerepelt a jelszó ismétlésének paramétere. Ezen úgy tudunk segíteni, hogy a felhasználók kontrollerében a `user_params` privát metódusban hozzáadjuk a `:password_confirmation` szimbólumot a `permit` paraméterlistájához.

Legelőször az útvonalakat tartalmazó `config/routes.rb` fájlban állítsuk át a webszerverünk gyökerét a feladatok nézet `index` oldalára.

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

A felhasználómenedzsmentet is alakítsuk át! Új felhasználót csak az adminisztrátor vehet fel, létező felhasználót csak az adminisztrátor törölhet, ugyanakkor egy felhasználó adatait mind a felhasználó maga, mind az adminisztrátor megnézheti, szerkesztheti. A szükséges módosításokat két helyen kell megtennünk, az útvonalak forrásában, és a felhasználók kontrollerében. Először módosítsuk az utóbbit. A megnéz (`show`), szerkeszt (`edit`) és frissít (`update`) akciók kivételével letiltjuk az akciókhoz hozzáférést a nem adminisztrátor felhasználóknak, és az akciók ugyanezen halmazára ellenőrizzük, hogy épp az aktuálisan bejelentkezett felhasználó akar-e hozzáférni. Ezt két `before_action` metódushívással tesszük meg, amelyekkel a kijelölt akciók esetén meghívunk egy-egy privát ellenőrző metódust. A `check_admin` azt ellenőrzi, hogy a van-e épp bejelentkezett felhasználó, vagy csak vendégről van-e szó, illetve amennyiben igen, akkor az a felhasználó bír-e az adminiszt-

rátori jogosultságokkal. Ha nem, akkor a kérés kiszolgálása helyett átirányítja az illetéktelen felhasználót a kezdőoldalra. A `check_the_same_user` az ellenőrzi, hogy az aktuálisan bejelentkezett felhasználó akar-e hozzáférni a saját profiljához, ha nem, akkor az előző metódushoz hasonlóan járunk el.

```
class UsersController < ApplicationController
  before_action :check_admin, except: [:show, :edit, :
    update]
  before_action :check_the_same_user, only: [:show, :
    edit, :update]

  private
  def check_admin
    if !(session[:user] && session[:user].type == 1)
      redirect_to url_for(:controller=>:tasks, :action
        => :index)
    end
  end

  def check_the_same_user
    if session[:user] != params[:id]
      redirect_to url_for(:controller=>:tasks, :action
        => :index)
    end
  end
end
```

Az útvonalakat a kontrollereknek megfelelően kell módosítanunk. Az összes felhasználókra vonatkozó útvonalat az `admin` prefixszel tesszük elérhetővé, és a felhasználók számára elérhető nézetekre külön – a REST konvencióinak megfelelő – útvonalakat definiálunk.

```
  resources :users, :path => '/admin/users'
  get '/users/:id' => 'users#show'
  get '/users/:id/edit' => 'users#edit'
  put '/users/:id' => 'users#update'
```

Viszonylag gyakori feladat fájlfeltöltés és -letöltés megvalósítása. Ezt illusztrálandó azt a használati esetet találtuk ki, hogy egy feladat minden felhasználó által kommentezhető, és a komment lehet szöveg vagy kép. Az ehhez szükséges migrációkaz az előző alkalommal már létrehoztuk. Az adatbázisban csak a fájl nevét tároljuk, magát az állományt egy a webszerver könyvtárában (`public`) létrehozott alkönyvtárban helyezzük el.

A feltöltött kép egy fájl, bináris adat, amit a feladat megtekintés (`show`) oldalán tölthet fel, ugyanott, ahol szöveges kommenthet hagyhat. Vegyünk fel egy-egy új format, a fájlfeltöltés megvalósítására, illetve a szöveges megjegyzés hagyására. A fájlfeltöltő formunk eseménykezelője legyen a kommentek kontrollere `upload` akciója. A fájlfeltöltéshez a formhoz hozzá kell rendelnünk a `multipart` opciót. A szöveges kommentet létrehozó formunk eseménykezelője legyen a `create` akció a kommentek kontrollerben, aminek paraméterül adjuk át annak a feladatnak az azonosítóját, amelyre a komment vonatkozik.

```
<%= form_tag  '/comments/create/'+@task.id.to_s do %>
<%= label_tag  :comment %> <br />
<%= text_area_tag  'comment' %>
<%= submit_tag  'Send' %>
<% end %>

<%= form_tag  '/comments/upload/'+@task.id.to_s,
   multipart: true do %>
<%= label_tag  :file %> <br />
<%= file_field_tag  :file %>
<%= submit_tag  'Upload' %>
<% end %>
```

Hogy lássuk a feladat vonatkozásában korábban tett megjegyzéseket, feltöltött képeket, keressük ki a kontrollerben a feladathoz kapcsolódó megjegyzéseket, amiket adjunk át a nézetnek egy példányváltozóval. Egyelőre csak a képeket tekintsük kommentnek, ezt abból tudhatjuk, hogy a komment példányban a `filename` attribútum rendelkezik-e értékkel.

```
@comments = Comment.where('filename␣is␣not␣null')
```

A feladat `show` nézetében tegyük lehetővé a képek letöltését, ami a letöltés megvalósítását illusztrálja. A letöltés paramétere a komment azonosítója, és a `download` akció az eseménykezelője.

```
<%= for c in @comments do %>
<%= link_to 'Image', '/comments/download/'+c.id.to_s %>
<% end %>
```

A kommentek kontrollerében jelenleg nincs akciónk, így nincsenek útvonalaink sem eseménykezelők felé. A szöveges komment beküldését és a fájlfeltöltést HTTP POST művelettel tesszük elérhetővé, a képletöltést pedig HTTP GET művelettel. Mind a három rendelkezik `id` paraméterrel, ami-

nek kötelezően meg kell jelennie az útvonalban. Ezeket az útvonalmintákat leképezzük a kontroller megfelelő akcióira, amelyeket még definiálnunk kell.

```
post '/comments/create/:id' => 'comments#create'
post '/comments/upload/:id' => 'comments#upload'
get '/comments/download/:id' => 'comments#download'
```

Az eseménykezelőket a kommentek kontrollerében valósítjuk meg. A szöveges komment eseményét lekezelő `create` akciót később valósítjuk meg, most koncentráljunk a fel- és letöltésre. 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 `Comment` 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 az feladatok listáját mutatjuk meg neki.

```
class CommentsController < ApplicationController
  def create
  end

  def upload
    if !session[:user].nil?
      if !params[:file].nil?
        Comment.save_file params[:file], params[:id],
            session[:user]
        redirect_to :back
      else
        redirect_to :controller=>'tasks', :action => :
            index
      end
    else
      redirect_to :controller=>'tasks', :action => :
          index
    end
  end
end
```

A fájl elmentése a `Comment` 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 Comment < ActiveRecord::Base
  def self.save_file(upload, task_id, user_id)
    dir = Rails.root.join("public", "data")
    if !File.exists? dir
      Dir.mkdir dir
    end
    fname = File.basename(upload.original_filename)
    #fname.gsub!(/^\w\._/"_"/)
    path = File.join(dir, fname)
    File.open(path, "wb") do |f| f.write(upload.read)
      end
    c = Comment.new
    c.user_id = user_id
    c.task_id = task_id
    c.filename = fname
    c.mime = upload.content_type
    c.save
  end
end
```

Letöltéskor előkeressük a kérés `id` paramétere által kijelölt megjegyzés objektumot, előállítjuk az ahhoz kapcsolódó fájl elérési útját a fájlrendszeren, majd a `send_file` Rails helperrel csatolmányként elküldjük válaszként.

```
  def download
    c = Comment.find params[:id]
    file = Rails.root.join("public", "data", c.filename
      )
    send_file file, :disposition => :attachment, :type
      => c.mime
  end
```

Mivel a teendőkből 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'
```

A tördelés megvalósítására már nem maradt időnk, a következő gyakorlaton innen folytatjuk.