Rails routing, kontrollerek
Gyakorlat

Kovács Gábor

2014. november 12.

A gyakorlat során az előző gyakorlatokon megkezdett feladat megoldását folytatjuk. Először fejezzük be a felhasználók és a játékok modellek szerkesztését. A felhasználók modellben az `authenticate` metódusban elkövettünk egy hibát, a `where` ugyanis enumerációt ad vissza, és nekünk egyetlen objektumra van szükségünk, ezért alkalmazzuk az eredményre a `first`, `last` vagy `take` metódusok egyikét.

A játékok modellben több-kettő reláció van a felhasználók modell felé. Az egyik kapcsolat neve a `Game` modellben `player1`, a másiké `player2`. Ezeket a `User` modellben nevezzük el `games1`-nek, illetve `games2`-nek tekintettel arra, hogy a játékos épp kihívó vagy kihívott volt-e. Ezek helyett egy olyan `games` függvényre van szükségünk, ami a felhasználó kihívó vagy kihívott szerepétől függetlenül tartalmazza az összes játékát. A metódus neve azt imitálja, mintha ez egy getterrel közvetlenül előállítható lehetne.

```
has_many :games1, class_name: 'Game', foreign_key: '
    player1'
has_many :games2, class_name: 'Game', foreign_key: '
    player2'
def games
  games1 + games2
end
```

Térjünk át a bejelentkezés, illetve a regisztráció ellenőrzésére! A webfelületen könnyen ellenőrizhetjük, hogy a bejelentezés, a regisztráció és a kijelentkezés megfelelően működik. A regisztráció során azonban fel kell készülnünk a hibás adatok megadására: nem egyező jelszómegerősítés, nem elég hosszú vagy foglalt felhasználónév stb. Ezeket a felhasználó modellben validációs metódusok hozzáadásával ellenőrizhetjük.

```
validates :username,
```

```
    presence: true,
    length: {in: 4..14, too_long: 'Too long username'},
    uniqueness: true
  validates :password,
    confirmation: true, :if => :password_required?
```

A fenti kódrészlet a `username` és a `password` attribútumokat vizsgálja. Az előbbi vonatkozásában ellenőrzi, hogy a felhasználó megadta-e egyáltalán, a hossza megfelelő-e, és egyedi-e, az utóbbi vonatkozásában pedig azt, hogy a megerősítés megegyezik-e a jelszóval, amit a `password_required?` metódussal ellenőrzünk.

```
  def password_required?
    encrypted_password.blank? || !password.blank?
    # self.new_record? || self.password?
  end
```

Rails 4-től a HTTP kérésben szereplő paraméterekre vonatkozó szűrés lépett életbe, aminek helye a kontroller `user_params` privát metódusa, amihez hozzáadjuk a `:password_confirmation` szimbólumot a `permit` paraméterlistájához.

Ellenőrizzük konzolon, hogy ki tudjuk-e ezeket a szabályokat játszani, illetve, hogy az általunk megadott üzenet megjelenik-e a hibák között. Láthatjuk, hogy létező felhasználónév megadása esetén a felhasználó létrehozása előtt lefut egy lekérdezés, ami a felhasználónév létezését ellenőrzi az adatbázisban, és miután találunk ilyet, a teljes felhasználólétrehozási tranzakciót visszagörgetjük. Ezután megvizsgálhatjuk a hiba okát az elmenteni kívánt objektumon a `errors.messages` metódussal. Kipróbálhatjuk a túl rövid és a túl hosszú felhasználóneveket, és az utóbbi esetén látjuk, hogy megjelenik a saját hibaüzenetünk. Végül azt is látjuk, hogy üres felhasználónévvel nem tudunk új felhasználót felvenni. A jelszó és megerősítésének hibás megadását a webfelületen ellenőrizhetjük.

```
rb(main):001:0> u = User.new
=> #<User id: nil, username: nil, encrypted_password:
    nil, email: nil, created_at: nil, updated_at: nil,
    salt: nil>
irb(main):002:0> u.username = 'valaki'
=> "valaki"
irb(main):003:0> u.email = 'valaki@bme.hu'
=> "valaki@bme.hu"
irb(main):004:0> u.save
   (0.3ms)  BEGIN
```

```
  User Exists (0.3ms)  SELECT  1 AS one FROM `users`
    WHERE `users`.`username` = BINARY 'valaki' LIMIT 1
   (0.2ms)  ROLLBACK
=> false
irb(main):005:0> u.errors
=> #<ActiveModel::Errors:0x00000003d67c40 @base=#<User
   id: nil, username: "valaki", encrypted_password: nil
   , email: "valaki@bme.hu", created_at: nil,
   updated_at: nil, salt: nil>, @messages={:username
   =>["has already been taken"]}>
irb(main):006:0> u.errors.messages
=> {:username=>["has already been taken"]}
irb(main):007:0> u.username = 'valaki2'
=> "valaki2"
irb(main):008:0> u.username = 'v'
=> "v"
irb(main):009:0> u.save
   (0.2ms)  BEGIN
  User Exists (1.4ms)  SELECT  1 AS one FROM `users`
    WHERE `users`.`username` = BINARY 'v' LIMIT 1
   (0.2ms)  ROLLBACK
=> false
irb(main):010:0> u.errors.messages
=> {:username=>["is too short (minimum is 4 characters)
   "]}
irb(main):011:0> u.username = '
   vvvvvvvvvvvvvvvvvvvvvvvvvvvvvv'
=> "vvvvvvvvvvvvvvvvvvvvvvvvvvvvvv"
irb(main):012:0> u.save
   (0.3ms)  BEGIN
  User Exists (0.4ms)  SELECT  1 AS one FROM `users`
    WHERE `users`.`username` = BINARY '
    vvvvvvvvvvvvvvvvvvvvvvvvvvvvvv' LIMIT 1
   (0.7ms)  ROLLBACK
=> false
irb(main):013:0> u.errors.messages
=> {:username=>["Too long username"]}
irb(main):014:0> u = User.new
=> #<User id: nil, username: nil, encrypted_password:
   nil, email: nil, created_at: nil, updated_at: nil,
   salt: nil>
```

```
irb(main):015:0> u.save
   (0.3ms)  BEGIN
  User Exists (0.6ms)  SELECT  1 AS one FROM `users`
    WHERE `users`.`username` IS NULL LIMIT 1
   (0.8ms)  ROLLBACK
=> false
irb(main):016:0> u.errors.messages
=> {:username=>["can't be blank", "is too short (
   minimum is 4 characters)"]}
```

Először távolítsuk el a harmadik gyakorlaton a felhasználók kontrollerében elhelyezett statikus értékeket, amit eddig csak a `create` akcióban tettünk meg. Azt látjuk, hogy a bejelentkezett felhasználó adatbázisból való lekérdezése több metódusban is előfordul lévén szükségük van rá vagy a kontrollerben vagy a nézetben. Ezt a lekérdezést kiválthatjuk egy `before_filter` szűrővel. A szűrő illeszkedjék a `edit`, a `friends`, a `show`, a `update` és a `destroy` metódusokra, és illeszkedés esetén fusson le a `find_user` privát metódus.

```
class UsersController < ApplicationController
  before_filter :find_user, :only => [:edit, :friends,
    :show, :update, :destroy]
  #before_filter :find_user, :except => [:new, :create,
      :index, :forgotten, :send_forgotten]

  private
    def find_user
      @user = User.find params[:id]
    end
end
```

Ahhoz, hogy a felhasználó kereshető lehessen, szükségünk van arra, hogy az `id` paraméter a HTTP kérés URI-jából elérkezhessen a kontrollerben. Ezért módosítanunk kell az útvonalaink konfigurációját a `config/routes.rb`-ben. Legelőször azonban állítsuk át a webszerverünk gyökerét a `hello` oldalra.

```
  root 'say#hello'
```

Azon akciók a felhasználók kontrollerben, ahol szükségünk van az `id` paraméterre a következők: `edit`, `update`, `friends` és a `show`. Módosítsuk az útvonalak listáját úgy, hogy az tartalmazza az eseménykezelők útvonalait is a `update`, a `create` és a `send_forgotten` akciókat, amelyekhez `post` HTTP

kéréssel férhetünk hozzá. A felhasználó profil oldalához rendeljünk útvonal helper metódusokat a `:as` hash kulccsal. Ez a példában egy `profile_path` és egy `profile_url` helpert hoz létre, amelyek paraméterezhetők egy felhasználóazonosítóval.

```
  get 'users/edit/:id' => 'users#edit'
  post 'users/update/:id' => 'users#update'
  get 'users/index/:page' => 'users#index'
  get 'users/:page' => 'users#index'
  get 'users/forgotten'
  post 'users/send_forgotten'
  get 'users/friends/:id' => 'users#friends'
  get 'users/show/:id' => 'users#show', :as => 'profile
    '
```

Visszatérve a felhasználók kontrollerére megírhatjuk a hiányzó metódusok törzsét. A profilt csak akkor endedjük módosítani, ha a felhasználó maga akarja azt elvégezni, különben irányítsuk át a felhasználót a saját profiljára. A felhasználó barátnak jelölt felhasználóinak listája üres volt, inicializáljuk az előző gyakorlaton bemutatott módon. A profil frissítése után mind a sikeres, mind a sikertelen átirányításban használjuk a `id` paramétert. Végül a felhasználó törléséhez is szükségünk van a törlendő felhasználó azonosítójára.

```
class UsersController < ApplicationController
  def edit
    if session[:user] != params[:id]
      redirect_to profile_path(session[:user])
    end
  end

  def friends
    @friends = @user.friends
  end

  def show
  end

  def update
    if session[:user] != params[:id]
      return
    end
    if @user.update_attributes params[:user]
```

```
        flash[:error] = 'Profile updated successfully'
        redirect_to :action => :show, :id => session[:
            user]
      else
        flash[:error] = @user.errors.message
        redirect_to action: :edit, id: session[:user]
      end
    end

    def destroy
      User.delete @user.id
      #@user.delete
    end
end
```

Hibás linkjeink találhatók a menüben is, és a formokban is. A menüben a felhasználót üdvözlő üzenetet testreszabhatjuk, hozzunk létre e célból egy `my_username` azonosítójú helper metódust. A bejelentkezett felhasználó menüjében vegyük fel a folyamatban lévő és a korábbi játékok listáját. A profil oldal linkjében használjuk fel a `profile_path` helpert a session hashben lévő felhasználóazonosítóval paraméterezve.

```
<% if logged_in? %>
  Welcome, <%= my_username %> !<br/>
  <%= link_to "Profile", profile_path(session[:user])
      %> <br/>
  <%= link_to "Statistics", profile_path(session[:user
      ]) %> <br/>
  <%= link_to "Show users", '/users/index' %><br/>
  <%= link_to "My ongoing games", url_for({controller:'
      games', action:'index', h:'1'}) %><br/>
  <%= link_to "My finished games", url_for({controller:
      'games', action:'index', h:'0'}) %><br/>
  <%= link_to "Logout", '/sessions/destroy' %>
<% else %>
```

A helper amennyiben van bejelentkezett felhasználó, kikeresi annak a felhasználónevét az adatbázisból, egyébként vendégként üdvözöljük.

```
module ApplicationHelper
  def my_username
    if session[:user]
      User.find(session[:user]).username
```

```
    else
      "Guest"
    end
  end
end
```

A játékok listázásához szükségünk van egy új útvonal felvételére. Ehhez a `games` erőforrás konfigurációjára van szükségünk, azonban nincs meg még a szükséges tudásunk, hogy ezt most megtegyük, viszont a kontroller megfelelő nézetét módosíthatjuk. A menüben a paramétert `h`-nak neveztük, amit csak akkor kell figyelembe vennünk, ha van bejelentkezett felhasználó. Ha `h` értéke `0`, akkor a folyamatban lévő, ha `h` értéke egy, akkor a befejezett, egyébként az összes játékra kíváncsiak vagyunk. A keresést a modell osztályban végezzük el egy-egy osztálymetódussal.

```
class GamesController < ApplicationController
  def index
    if params[:h] && session[:user]
      if params[:h] == '0'
        @games = Game.find_ongoing_games(session[:user
            ])
      elsif params[:h] == '1'
        @games = Game.find_finished_games(session[:user
            ])
      else
        @games = Game.all
      end
    else
      @games = Game.all
    end
  end
end
```

A kereső metódusokban olyan játékokra van szükségünk, amelyekben vagy a `player1` vagy a `player2` értéke megegyezik a paraméterként megkapott felhasználóazonosítóval. A keresés másik feltétele a `result` attribútumra vonatkozik.

```
class Game < ActiveRecord::Base
  def self.find_ongoing_games(user)
    Game.where("result = 0 and (player1 = #{user} or
        player2 = #{user})")
  end
```

```
  def self.find_finished_games(user)
    Game.where("result > 0 and (player1 = #{user} or
        player2 = #{user})")
  end
end
```

Az `edit.html.erb` nézetben hibás az eseménykezelő útvonalunk, hiányzik a szerkeszteni kíván felhasználó azonosítója, adjuk ezt hozzá.

```
<%= form_for @user, :url => { :action => 'update', :id
    => @user.id } do |f| %>
```

Az `index.html.erb` nézetben a kihívást és a barátok megtekintését kell lehetővé tennünk a bejelentkezett felhasználó számára. Ehhez a kontrollerben ellenőriznünk kell, hogy a felhasználó bejelentkezett-e egyáltalán. Ha igen, akkor a `userid` példányváltozóban adjuk át a felhasználó objektumát a nézetnek.

```
class UsersController < ApplicationController
  def index
    @users = User.all
    if session[:user]
      @userid = session[:user]
    else
      @userid = ''
      flash[:error] = "Please log in to challenge"
    end
  end
end
```

Csak az a felhasználó hívható ki egy linkre kattintással, akinek az azonosítója nem egyezik meg a bejelentkezett felhasználó azonosítójával, ezt a `myself` helperrel ellenőrizzük. A játék létrehozásának paraméterül át kell adnunk a kihívó felhasználó azonosítóját. A barátok megtekintése szintén egy linkre kattintással működik.

```
<ul>
<%= for u in @users do %>
  <li>
    <%= u.username %>
    <% if !myself(u.id) %>
      <%= link_to "Challenge", url_for({ :controller =>
          "games", :action => "create", :cid => u.id.
```

```
            to_s }) %>
      <% end %>
      <% if logged_in? %>
        <%= link_to "Friends", "/users/friends/"+@userid.
            to_s %>
      <% end %>
    </li>
<% end %>
</ul>
```

A `myself` helper csak a felhasználók kontrollerben és nézetekben használható fel, és az ellenőrzi, hogy be van-e jelentkezve egy felhasználó, és annak azonosítója megegyezik-e a paraméterül átadott azonosítóval.

```
module UsersHelper
  def myself(myid)
    session[:user] && session[:user] == myid
  end
end
```

A barátok nézet nem létezik, viszont a kontrollerben inicalizáltuk a `friends` enumerációt példányváltozóként, amit meg tudunk jeleníteni. A nézetben minden tegyünk minden barátot kihívhatóvá, és engedélyezzük a profil oldaluk megtekintését.

```
<h1>My dear friends</h1>
<ul>
<% for u in @friends %>
  <li>
      <%= link_to u.username, profile_path(u.id) %>
      <%= link_to "Challenge", url_for(controller: "
          games", action: "create", cid: u.id.to_s) %>
  </li>
<% end %>
</ul>
```

Mivel felhasználókból nagyságrendekkel több lehet, mint amennyit az `index` 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'
```

Az index nézetre a navigációs linkeket az alábbi függvényhívással adhatjuk hozzá, ami itt a `@users` enumerációt tördeli. A navigációs linkek a `page` paramétert fogja állítani a rákövetkező HTTP kérésben, de marad az oldalon. Ezért módosítanunk kell ismét az útvonalakat a jobb olvashatóság céljából.

```
<%= will_paginate @users %>
```

Legyen az index nézet elérhető úgy is, hogy explicite kiírjuk az `index` útvonaltagot, és anélkül is. Hogy a GET kérésben ne URL kódoltan jelenjék meg a `page` paraméter, azt nevesítjük az útvonalban.

```
  get 'users/index/:page' => 'users#index'
  get 'users/:page' => 'users#index'
```

Mivel az nézeten csak az aktuális lapon megjelenítendő felhasználók listájára van szükségünk, a `User.all` helyett keressük ki az aktuális töredék felhasználóit. A keresést a modell osztályban valósítsuk meg.

```
class UsersController < ApplicationController
  def index
    @users = User.get_users_page(params[:page])
  end
end
```

A felhasználók modellben a keresést egy osztálymetódussal valósítjuk meg, az oldalszámot paraméterként kapjuk meg, egy oldalon 2 felhasználót jelenítsünk meg. A törtelés előtt azonban még végezzünk egy felhasználónév szerinti rendezést.

```
  def self.get_users_page(page)
    #User.all
    User.order('username').paginate(:page => page, :
        per_page=>2)
  end
```