Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2021. október 26.

1. Session, felhasználókezelelés

1.1. Titkosított jelszó

Az előző gyakorlaton megkezdett példát folytatjuk a felhasználói sessionök kezelésének megvalósításával, illetve az adatmodell kialakításával. Az előző alkalommal három modellt hoztunk létre, a felhasználók `User` nevű modelljét, a méhlegelők `BeePasture` nevű modelljét, valamint a megfigyelések `Sightings` nevű modelljét. A felhasználók modellje a következőképp néz ki az adatbázisban.

```
MariaDB [gyakorlat_development]> desc users;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| name       | varchar(255) | YES  |     | NULL    |                |
| password   | varchar(255) | YES  |     | NULL    |                |
| email      | varchar(255) | YES  |     | NULL    |                |
| created_at | datetime(6)  | NO   |     | NULL    |                |
| updated_at | datetime(6)  | NO   |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
6 rows in set (0.001 sec)
```

Először növeljük portálunk biztonságát azzal, hogy a jelszavakat nem szövegesen, hanem titkosítva tároljuk. Ez a jelszó mező átnevezéséből és egy új, a titkosítás során a felhasználó számára egyedi attribútum felvételéből áll. A Rails az új attribútum felvételéhez képes automatikusan invertálható migrációt generálni jól definiált migrációnév esetén. A hozzáadott attribútumnak ilyen esetben a migráció neve után kell szerepelnie. Azonban az attribútum átnevezését magunknak kell majd hozzáadnunk a migrációhoz.

```
kovacsg@debian:~/gyakorlat> rails g migration AddSaltToUsers salt:string
Running via Spring preloader in process 22208
      invoke  active_record
```

```
      create    db/migrate/20211026102543_add_salt_to_users.rb
```

Nézzük meg, milyen hash függvények állnak a rendelkezésünkre, amelyek felhasználhatók a titkosítás során. Szükségünk lesz véletlen, visszafejthetetlen és egyben olvashatatlan karaktersorozatokra, és stringből olvashatatlan, visszafejthetetlen karaktersorozatot előállító függvényre.

```
irb(main):001:0> SecureRandom
=> SecureRandom
irb(main):002:0> SecureRandom.hex 8
=> "cca99993cfb38706"
irb(main):003:0> SecureRandom.hex 8
=> "4394292027fea07b"
irb(main):004:0> Digest::SHA1.hexdigest 'titok'
=> "46ff53e764c4acf97b54db2020573049d2e3dab3"
irb(main):005:0> SecureRandom.base64 16
=> "xE6WHjPSwW4Z+rEJPEYHeQ=="
```

Mivel az attribútum átnevezése nem invertálható, vagy a `change` metódusban használjuk a `reversible` függvényt, vagy a `change` helyett két függvényt definiálunk `up`, illetve `down` néven. Most ez utóbbit választjuk. Az automatikusan generált invertálható műveletről el kell feledkeznünk, magunknak kell kettéválasztanunk a műveletet. Mivel az adatbázisban már van adat, fel irányú migráció esetén gondoskodunk kell azok új attribútumainak inicializálásáról.

```
class AddSaltToUsers < ActiveRecord::Migration[6.1]
  # def change
    #reversible.do |dir|
    #  dir.up
    #  dir.down
    # end
  # end

  def up
    add_column :users, :salt, :string
    rename_column :users, :password, :encrypted_password
  end

  def down
    drop_column :users, :salt
    rename_column :users, :encrypted_password, :password
  end
end
```

Hajtsuk végre a migrációt érvényre juttatandó az adatmodellünk változásait!

```
kovacsg@debian:~/gyakorlat/db/migrate# rails db:migrate
== 20211026102543 AddSaltToUsers: migrating ===================================
-- add_column(:users, :salt, :string)
   -> 0.0305s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0193s
```

```
== 20211026102543 AddSaltToUsers: migrated (0.0528s)
   ==========================
```

Nézzük meg a migráció eredményét. Az adatbázisban már jelen lévő rekorddal nem tudunk most mit kezdeni. Ugyan titkosíthatjuk a jelszó attribútumát, de egy le-, majd egy felirányú migrációval az érvénytelen lesz, hiszen a hash egy egyirányú művelet, és így nem visszaállítható.

```
MariaDB [gyakorlat_development]> desc users;
+--------------------+--------------+------+-----+---------+----------------+
| Field              | Type         | Null | Key | Default | Extra          |
+--------------------+--------------+------+-----+---------+----------------+
| id                 | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| name               | varchar(255) | YES  |     | NULL    |                |
| encrypted_password | varchar(255) | YES  |     | NULL    |                |
| email              | varchar(255) | YES  |     | NULL    |                |
| created_at         | datetime(6)  | NO   |     | NULL    |                |
| updated_at         | datetime(6)  | NO   |     | NULL    |                |
| salt               | varchar(255) | YES  |     | NULL    |                |
+--------------------+--------------+------+-----+---------+----------------+
7 rows in set (0.001 sec)
```

A `new_record?` függvény azt állapítja meg egy ActiveRecord objektumról, hogy azt elmentettük-e már az adatbázisba, ezt az `id` attribútum ellenőrzésével végzi el, ami alapértelmezés szerint `nil`. A memóriában létrehozott `ActiveRercord` objektumok az első mentés műveletig új rekordnak számítanak, az `id` attribútumuk csak annak hatására inicializálódik.

A migrációban minden egyes felhasználói rekordhoz hozzáadtunk egy egyéni a jelszó titkosításához használt random kulcs tárololására használt attribútumot és egy a típust tároló attribútumot, továbbá a jelszó attribútumot pedig átnevezzük, így az titkosítatlanul nem kerülhet bele az adatbázisba.

A jelszó attribútumot átneveztük, viszont a felületen továbbra is használjuk, ezért csak a modell osztályra korlátozva elérhetővé újra tesszük. Az attribútum csak a memóriában él, az adatbázisba nem kerül ki az értéke.

```
class User < ApplicationRecord
  attr_accessor :password
end
```

Bejelentkezéskor a megadott jelszót már az adatbázisban található titkosított jelszóval kell összevetnünk, ezért a `User` modell példányának mentésekor (regisztráció során vagy a felhasználói jelszó módosításakor) a `password`

példányváltozotót `encrypted_password` attribútummá kell transzformálnunk. Ezt a következőképp tesszük meg. Definiálunk egy `encrypt` azonosítójú osztálymetódust, amely a `password` és `salt` példányváltozók alapján egy hash függvénnyel egy kódolt karaktersorozatot hoz létre. Definiálunk továbbá egy `encrypt_password` azonosítójú metódust, amely az összes nem üres jelszó esetére elvégzi a titkosítást, illetve új, még el nem mentett rekord esetén inicializálja a `salt` attribútum értékét egy véletlen számmal. Végül a `before_save` metódussal jelezzük, hogy a `save` metódus minden egyes meghívása előtt hívódjék meg a `encrypt_password` metódus.

```
class User < ApplicationRecord
  before_save :encrypt_password

  def self.encrypt(pass, salt)
    Digest::SHA1.hexdigest pass+salt
  end

  def encrypt_password
    return if password.blank?
    if self.new_record?
      self.salt = SecureRandom.base64 16
    end
    self.encrypted_password = User.encrypt(password, salt)
  end
end
```

Hozzunk létre egy felhasználót (1. sor), mentsük el (2. sor), és nézzük meg, hogy működik-e a jelszó titkosítása.

```
kovacsg@debian:~/gyakorlat> rails c
irb(main):001:0> u = User.new name: 'senki', email: 'senki@mail.bme.hu',
    password: 'titok'
   (0.4ms)  SET NAMES utf8mb4,  @@SESSION.sql_mode = CONCAT(CONCAT(
       @@sql_mode, ',STRICT_ALL_TABLES'), ',NO_AUTO_VALUE_ON_ZERO'),
       @@SESSION.sql_auto_is_null = 0, @@SESSION.wait_timeout = 2147483
=> #<User id: nil, name: "Valaki", encrypted_password: nil, email: "
    valaki@mail.bme.hu", created_at: nil, updated_at: nil, salt: nil>
irb(main):002:0> u.save
   (0.2ms)  BEGIN
  User Create (7.0ms)  INSERT INTO `users` (`name`, `encrypted_password`, `
      email`, `created_at`, `updated_at`, `salt`) VALUES ('senki', '6270
      eec526ea30148289478f2ab88b0be8bb2b54', 'senki@mail.bme.hu', '
      2021-10-26 11:25:47.407559000', '2021-10-26 11:25:47.407559000', '0
      FFiqHGn1kqACQUzP9YAsQ==')
   (2.0ms)  COMMIT
=> true
```

Nézzük meg, hogy milyen elemi műveletekkel férhetünk egy vagy több adatbázisbeli rekordhoz hozzá Rails-ből. A `find` (1. sor) az `id` attribútum értéke alapján keres, és egy rekordhoz tartozó Ruby objektumot ad vissza. A `first` (4. sor), `last` (2. sor) és `take` (3. sor) rendre az elsődleges kulcs szerinti első, utolsó, és az alapértelmezett rendezés szerinti első rekordot adják vissza. Ezeknek egész paramétert adva a rendezés után visszaadott rekordok számát adhatjuk meg (5-6. sor), a visszatérési érték

`ActiveRecord::Base` objektumról tömbre változik. Egy attribútum alapján egy rekordot a `find_by` metódussal kereshetünk (8.sor). Az általános megoldás a keresésre a `where` (9-10. sor), amely esetén a visszatérési érték `ActiveRecord::Base` objektumról `ActiveRecord::Relation` objektumra változik, ami tömbként működik, és kaszkádosan végrehajthatók rá a fenti keresési műveletek, például a `take`, amellyel egy találatot kivehetünk abból. Az `all` függvény az modell összes rekordját visszaadja egy tömbben.

```
kovacsg@debian:~/gyakorlat> rails c
Loading development environment (Rails 6.0.3.3)
irb(main):001:0> User.find 2
  User Load (0.9ms)  SELECT `users`.* FROM `users` WHERE `users`.`id` = 2
      LIMIT 1
=> #<User id: 2, username: "senki", encrypted_password: [FILTERED], email: "
    senki@mail.bme.hu", created_at: "2021-10-26 10:37:27.429657000 +0000",
    ated_at: "2021-10-26 10:37:27.429657000 +0000", salt: [FILTERED]>
irb(main):002:0> User.last
  User Load (0.9ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id`
      DESC LIMIT 1
=> #<User id: 3, username: "valaki", encrypted_password: [FILTERED], email:
    "valaki@mail.bme.hu", created_at: "2021-10-26 11:25:47.407559000 +0000"
    pdated_at: "2021-10-26 11:25:47.407559000 +0000", salt: [FILTERED]>
irb(main):003:0> User.take
  User Load (0.6ms)  SELECT `users`.* FROM `users` LIMIT 1
=> #<User id: 2, username: "senki", encrypted_password: [FILTERED], email: "
    senki@mail.bme.hu", created_at: "2021-10-26 10:37:27.429657000 +0000",
    ated_at: "2021-10-26 10:37:27.429657000 +0000", salt: [FILTERED]>
irb(main):004:0> User.first
  User Load (0.7ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC
      LIMIT 1
=> #<User id: 2, username: "senki", encrypted_password: [FILTERED], email: "
    senki@mail.bme.hu", created_at: "2021-10-26 10:37:27.429657000 +0000",
    ated_at: "2021-10-26 10:37:27.429657000 +0000", salt: [FILTERED]>
irb(main):005:0> User.first(2)
  User Load (0.4ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC
      LIMIT 2
=> [#<User id: 2, username: "senki", encrypted_password: [FILTERED], email:
    "senki@mail.bme.hu", created_at: "2021-10-26 10:37:27.429657000 +0000",
    dated_at: "2021-10-26 10:37:27.429657000 +0000", salt: [FILTERED]>, #<
    User id: 3, username: "valaki", encrypted_password: [FILTERED], email: "
    valakail.bme.hu", created_at: "2021-10-26 11:25:47.407559000 +0000",
    updated_at: "2021-10-26 11:25:47.407559000 +0000", salt: [FILTERED]>]
irb(main):006:0> User.first(2)[1]
  User Load (0.4ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC
      LIMIT 2
=> #<User id: 3, username: "valaki", encrypted_password: [FILTERED], email:
    "valaki@mail.bme.hu", created_at: "2021-10-26 11:25:47.407559000 +0000"
    pdated_at: "2021-10-26 11:25:47.407559000 +0000", salt: [FILTERED]>
irb(main):007:0> User.find_by email: 'senki@mail.bme.hu'
  User Load (0.9ms)  SELECT `users`.* FROM `users` WHERE `users`.`email` = '
      senki@mail.bme.hu' LIMIT 1
=> #<User id: 2, username: "senki", encrypted_password: [FILTERED], email: "
    senki@mail.bme.hu", created_at: "2021-10-26 10:37:27.429657000 +0000",
    ated_at: "2021-10-26 10:37:27.429657000 +0000", salt: [FILTERED]>
irb(main):008:0> User.where(email: 'senki@mail.bme.hu').take
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`email` = '
      senki@mail.bme.hu' LIMIT 1
=> #<User id: 2, username: "senki", encrypted_password: [FILTERED], email: "
    senki@mail.bme.hu", created_at: "2021-10-26 10:37:27.429657000 +0000",
```

```
    updated_at: "2021-10-26 10:37:27.429657000 +0000", salt: [FILTERED]>
irb(main):011:0>
```

1.2. Be- és kijelentkezés

Következő lépésként tegyük rendbe a felhasználói session kezelését, ami a `menu` akcióinak megvalósítását, a jelszó titkosítását, és a titkosított jelszó adatbázisban való eltárolását jelenti első körben. Másodszor pedig a felhasználó regisztráció folyamatát érinti.

Ezután rátérhetünk a felhasználói session megvalósítására. Ehhez létrehozzuk a `sessions` kontrollert, amelynek `create` és `destroy` metódusai léptetik be, illetve ki a felhasználót.

```
kovacsg@debian:~/gyakorlat/app/controllers# rails g controller sessions
    create destroy
Running via Spring preloader in process 25412
      create  app/controllers/sessions_controller.rb
       route  get 'sessions/create'
get 'sessions/destroy'
      invoke  erb
      create    app/views/sessions
      create    app/views/sessions/create.html.erb
      create    app/views/sessions/destroy.html.erb
      invoke  test_unit
      create    test/controllers/sessions_controller_test.rb
      invoke  helper
      create    app/helpers/sessions_helper.rb
      invoke    test_unit
      invoke  assets
      invoke    scss
      create      app/assets/stylesheets/sessions.scss
```

A létrehozott `create` és `destroy` nézetekre nincs szükségünk, azokat töröljük. A `sessions` kontrollerhez ezek után nem tartozik nézet, a `login`, illetve a `logout` link eseményeit kezeli le. Ellenőrizzük, hogy a menüben, vagyis a `layouts/_guest_menu.html.erb`-ben a form akciója a `/sessions/create`-re mutat-e, illetve a belépett felhasználó menüjében a `Logout` link a `/sessions/destroy`-ra mutat-e. Belépéskor, illetve kilépéskor, megpróbálunk az aktuális oldalon maradni. A `routes.rb` konfigurációs fájlhoz adjuk hozzá a `post 'sessions/ create'` és a `get 'sessions/ destroy'`, ugyanis bejelentkezéskor HTTP POST üzenetben adatokat is küldünk a szerver felé, kijelentkezéskor pedig HTTP GET üzenet elég. Valósítsuk ezeket meg, és rendeljük hozzájuk a `login`, illetve a `logout` címkéket. Ez utóbbiak `login_path` és `login_url` azonosítóval segédfüggvényeket hoznak létre, amelyekkel az útvonalra, illetve a teljes URL-re hivatkozjatunk a `login` alias esetén, a `logout` alias hasonlóan működik. Végül nevezzük el a hello világ nézetünket `hello`-nak.

```
Rails.application.routes.draw do
  post 'sessions/create', to: 'sessions#create', as: 'login'
  match 'sessions/destroy', to: 'sessions#destroy', as: 'logout', via: [:get
      , :delete]
  get 'say/hello', to: 'say#hello', as: 'hello'
end
```

A következő lépés a felhasználó hitelesítésének megvalósítása, amit a `User` modellben teszünk meg egy osztálymetódussal. A hitelesítés két argumentummal rendelkezik egy felhasználói email címmel és egy jelszóval, és a sikeresen hitelesített felhasználó objektumával vagy `nil`-lel tér vissza. Először megkeresi a rekordok között a felhasználó azonosítójának megfelelő rekordot, majd elvégzi a hitelesítést. Bármelyik sikertelensége esetén a visszatérési érték `nil`. A hitelesítés (`authenticated?` metódus) azt ellenőrzi, hogy a titkosított jelszó attribútum megegyezik-e a jelszó titkosítása által visszaadott értékkel.

```
class User < ApplicationRecord
  def self.authenticate(email, pass)
    user = User.where(email: email).take
    user && user.authenticated?(pass) ? user : nil
  end

  def authenticated?(pass)
    self.encrypted_password == User.encrypt(pass, self.salt)
  end
end
```

A kontrollerünk ezek után a következőképp néz ki. A hitelesítés imént megírt metódusának visszatérési értékét a `user` kontroller példányváltozóhoz rendeljük. Az email címet és a jelszót a `params` hash-ből vesszük ki a `menu`-ben megadott név alapján. A `params` hash alábbi használata veszélyes lehet, éles rendszerben ne használjuk közvetlenül! A SQL injection támadásokat elkerülendő az aposztrófokat escape-elnünk kell!

Ha a hitelesített felhasználó értéke nem `nil`, akkor a `session` hash `:user` szimbólummal hivatkozott értékének beállítjuk a felhasználó `id` attribútumának értékét, majd visszairányítjuk a felhasználót az előző oldalra. Ellenkező esetben egy hibaüzenetet küldünk a következő oldalnak a `flash` hash-en keresztül, és ugyancsak visszairányítjuk a felhasználót az előző oldalra. Kilépéskor töröljük a `session` hash tartalmát, és egy `flash` üzenettel visszairányítjuk a felhasználót az előző oldalra. Az előző oldal vagy a JavaScript history-ból, vagy a kérés fejrészének `HTTP_REFERER` opciójából határozható meg, ha egyik sem adott, akkor a helló világ oldalra kerükjünk át.

```
class SessionsController < ApplicationController
  def create
    @user = User.authenticate params[:email], params[:password]
    if @user
      session[:user] = @user.id
```

```
      flash[:notice] = 'Successful login'
    else
      flash[:notice] = 'Invalid email address or password'
    end
    redirect_back fallback_location: hello_path
  end

  def destroy
    reset_session
    flash[:notice] = 'Successful logout'
    redirect_back fallback_location: hello_path
  end
end
```

A flash hashen keresztül értéket adhatunk át a következő HTTP kérésre adott válasz számára. A megjelenítendő üzenet helye legyen az oldal szerkezetében a menü felett hozzáadjuk.

```
<p><%= flash[:notice] %></p>
```

Ezek után már el tudjuk dönteni, hogy egy felhasználó mikor van bejelentkezve az előző alkalommal írt alkalmazás szintű helperben. Ha a `session :user` szimbólumhoz tartozó értéke nem üres, akkor a felhasználó be van jelentkezve.

```
module ApplicationHelper
  def logged_in?
    session[:user]
  end
end
```

1.3. Regisztráció, profil szerkesztése

Felhasználó létrehozásához és adatainak módosításához szükséges nézeteket már létrehoztuk, valósítsuk meg a formokat kezelő kontroller akciókat. A regisztrációhoz a `users` kontroller `create` akciója tartozik, a profil módosításához pedig az `update` akció tartozik. Takarítsuk ki az előző gyakorlaton bedrótozott értékeket a kontrollerből! A rekordok biztonsága végett a HTTP kérés paramétereinek lehetséges kulcsait korlátozzunk, és az alapján hozunk létre új felhasználót. Az ellenőrzést a `user_params` privát metódus végzi el.

```
class UsersController < ApplicationController
  def create
    @user = User.new(user_params)
    if @user.save
      session[:user] = @user.id
      flash[:notice] = "Successful registration"
      redirect_back fallback_location: hello_path
    else
      flash[:notice] = @user.errors.messages
      #redirect_back fallback_location: hello_path
      render :new
    end
```

```
  end

  def edit
  end

  def update
    if @user.update(user_params)
      flash[:notice] = "Update successful"
      redirect_back fallback_location: hello_path
    else
      flash[:notice] = "Could not update profile"
      redirect_back fallback_location: hello_path
    end
  end

  private
    def user_params
      params.require(:user).permit([:name, :email, :password, :
          encrypted_password])
    end
end
```

A felhasználó `id` attribútumának értéke, akárcsak az HTTP kérés összes egyéb paramétere bekerül a `params` hash-be, ahonnan kikereshetjük, ha szükségünk van rá. Ha épp nem új felhasználót hozunk létre, akkor ezt meg kell tennünk. Bejelentkezett felhasználó esetén ugyanerre a célra felhasználhatjuk a sessionbel tárolt értéket. A felhasználó azonosító alapján való előrekeresésére több akció esetén is szükségünk van, de nem akarjuk többször ugyanazt a kódrészletet leírni, ezért felhasználjuk a `before_action` függvényt, melynek argumentuma egy függvényazonosító. Mivel több kontroller esetén is szükségünk van a felhasználóra, a keresést a kontrollerek közös ősosztályában tesszük meg. Az a függvény a kontroller összes publikus akciója előtt felut, ha benne van az `only` utáni felsorolásban, vagy nincs benne a `except` utáni felsorolásban az akció azonosítója. Ha egyik felsorolás sincs megadva, akkor mindenképp lefut a függvény. Ez a kódunk karbantarthatóságát javítja.

```
class ApplicationController < ActionController::Base
  before_action :find_user

  private
    def find_user
      if session[:user]
        @user = User.find session[:user]
      end
    end
end
```

Több felhasználó számára is elérhetővé szeretnénk tenni a portálunkat, ez lehetséges, mert a felhasználók adatait most már az adatbázisból vesszük elő. Ha egy konkrét felhasználó adait szeretnénk megnézni, szerkeszteni vagy módosítani, akkor a HTTP kérés paramétereként át kell adnunk a felhasználó adatbázisbeli azonosítóját is, hogy a megfelelő felhasználó adatai jelenlenek

meg, módosuljanak. Az első módosítás a HTTP kérés paraméterezhetővé tétele, amit a `routes.rb` konfiguráció állományban teszünk meg – magyarázat a következő előadáson.

```
  get 'users/new', to: 'users#new', as: 'registration'
  post 'users/create', to: 'users#create', as: 'register'
  get 'users/forgotten', to: 'users#forgotten', as: 'forgotten'
  post 'users/send_forgotten', to: 'user#send_forgetten', as: '
      send_forgotten'
  get 'users/edit/:id', to: 'users#edit', as: 'profile'
  put 'users/update/:id', to: 'users#update', as: 'edit_profile'
```

Eztán mind a vendégfelhasználó, mind a bejelentkezett felhasználó menüjében módosítjuk a linkeket.

```
<p>Hello, <%= @user.username %>!</p>
<p><%= flash[:notice] %></p>

<%= link_to "Profile", profile_path(@user.id) %><br/>
<%= link_to "Add new bee pasture", new_beepasture_path %><br/>
<%= link_to "All beepastures", beepastures_path %><br/>
<%= link_to 'Logout', logout_path %>
```

```
<p>Hello, guest!</p>
<p><%= flash[:notice] %></p>

<fieldset>
  <legend>Login</legend>
  <%= form_tag login_path, method: :post do %>
    <%= label_tag 'email', 'Email' %>
    <%= text_field_tag 'email', '', size: 14 %>
    <%= label_tag 'password', 'Password' %>
    <%= password_field_tag 'password', '', size: 14 %>
    <%= submit_tag 'Login' %>
  <% end %>
</fieldset>

<%= link_to "Register", registration_path %><br/>
<%= link_to "Forgotten password", forgotten_path %>
```

1.4. Felhasználó által megadott adatok ellenőrzése

A regisztrációkor az elmentendő felhasználót még az elmentés előtt validáljuk, az emailcím attribútumnak nemüresnek kell lennie (`:presence`), és egyedinek (`:uniqueness`) kell lennie, és ha ez nem teljesül, akkor visszajelzünk, hogy nem megfelelő felhasználónévről van szó. A név mezőt kötelező megadni a regisztráció során, és nem lehet üres. A jelszónak és annak ismétlésének meg kell egyeznie (`:confirmation`), ha az elmentett jelszó nem üres (`password_required?` metódussal vizsgálva). A `confirmation` opció létrehozza a modell objektumban a `_confirmation` szuffixű settert és gettert, így a kontroller hozzá tudja rendelni ahhoz a formból érkező adatokat. Ezeket

az ellenőrzéseket a modell osztályban validációs helper metódusokkal tesszük meg.

```
class User < ActiveRecord::Base
  validates :name, presence: true
  validates :email, { presence: true, uniqueness: true }
  validates :password, confirmation: true, if: :password_required?

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

Konzolon és a webfelületen ellenőrizhetjük, hogy elmenthető-e üres névvel névvel és létező email címmel egy új rekord! Az `ActiveRecord` példányokról a `valid?` metódussal kérdezhetjük meg, hogy átmennek-e az osztályában definiált validációkon. Ha nem, akkor a hibaüzeneteket az `errors` példányváltozóban érhetjük el, amiket kivezethetünk a nézetekre.

```
kovacsg@debian:~/gyakorlat> rails c
irb(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.valid?
  User Exists? (0.4ms)  SELECT 1 AS one FROM `users` WHERE `users`.`email`
      IS NULL LIMIT 1
=> false
irb(main):003:0> u.errors.messages
=> {:username=>["can't be blank"], :email=>["Empty email"]}
irb(main):004:0> u.email = 'senki@mail.bme.hu'
=> "senki@mail.bme.hu"
irb(main):005:0> u.save
  TRANSACTION (0.6ms)  BEGIN
  User Exists? (0.4ms)  SELECT 1 AS one FROM `users` WHERE `users`.`email` =
      'senki@mail.bme.hu' LIMIT 1
  TRANSACTION (0.3ms)  ROLLBACK
=> false
irb(main):006:0> u.errors.messages
=> {:username=>["can't be blank"], :email=>["Email already in use"]}
```

A felhasználói regisztáció nézetén (`new.html.erb`) a hibaüzeneteket egyszerűen megjeleníthetjük a következő kódrészlet segítségével:

```
<h1>Registration</h1>
<%= flash[:notice] %>

  <% if @user.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@user.errors.count, "error") %> prohibited this user
          from being saved:</h2>

      <ul>
      <% @user.errors.full_messages.each do |message| %>
        <li><%= message %></li>
      <% end %>
      </ul>
    </div>
  <% end %>
```

Próbáljunk meg ezután felhasználót felvenni a webfelületen hibás adatokkal, és nézzük meg a hibaüzeneteket! Láthatjuk, hogy ezek a műveletek nem hajtódnak végre, és a Rails megjelöli a hibás beviteli mezőket. Ezután a validációs üzenet is megváltozik, amit konzolon ellenőrizhetünk.

A hibaüzenetet testreszabhatjuk a lokalizációs beállításokban:

```
  activerecord:
    errors:
      models:
        user:
          attributes:
            mail:
              blank: 'Empty email
              taken: 'Email already used'
```

Az előző gyakorlaton kezdeti adatokkal módosítottuk az események kontrollert. Most távolítsuk el azokat, és írjuk vissza az automatikusan generált kódrészleteket.

2. Az adatmodell kialakítása

2.1. Modellosztályok kapcsolatai

A méhlegelőmegfigyelő portálunk adatmodellje a következő elemekből áll:

- Felhasználó (`User`)
- Méhlegelő (`BeePasture`)
- Megfigyelés (`Sighting`)

Ezeket a korábbi gyakorlatokon már létrehoztuk.

Egy méhlegelőn (`Video`) sok megfigyelés történhet. A `references` típus a migrációban automatikusan létrehozta a `belongs_to` deklarációt a megfigyelés oldalon. A reláció fordított irányú navigációjának engedélyezése végett módosítjuk a felhasználók modell osztályát.

```
class BeePasture < ApplicationRecord
  has_many :sightings
end
```

Ezeket a megfigyeléseket egy-egy felhasználó teszi, és mivel egy felhasználó több megfigyelést is tehet, ez is egy egy-több kapcsolat.

```
class User < ApplicationRecord
  has_many :sightings
end
```

Tehát a megfigyelések osztályban két `belongs_to` fog szerepelni.

```
class Sighting < ApplicationRecord
  belongs_to :beepasture
  belongs_to :user
end
```

A tudományos igényességű megfigyelések miatt mind magáról a méhlegelőről, mind a megfigyelésekről képeket, videókat kell feltöltenünk a portálra, az állítások igazolása végett. Ezért létrehozunk egy új modellt, a csatolmányokat, amely polimorfikusan kapcsolódik e két modellhez. A csatolmányokról megjegyezzük a fájlok eredeti nevét string típusú attribútumban, a fájlok fálrendszeren való elérési útját string típusú attribútumban, a fájl méretét egész számként, és a csatolmány MIME-típusát string típusú attribútumban. A csatolmány egy fiktív modellhez kapcsolódik polimorfikusan

```
kovacsg@debian:~/gyakorlat/app/models# rails g model attachment name:string
    path:string size:integer mime:integer content:references
Running via Spring preloader in process 1665
      invoke  active_record
      create    db/migrate/20211026113918_create_attachments.rb
```

A generált migrációban idegen kulcsot módosítjuk a `polymorphic` opcióval, és az index eltávolításával.

```
class CreateAttachments < ActiveRecord::Migration[6.1]
  def change
    create_table :attachments do |t|
      t.string :name
      t.string :path
      t.integer :size
      t.string :mime
      t.references :content, null: false, polymorphic: true

      t.timestamps
    end
  end
  end
```

Hajtsuk végre a migrációt!

```
kovacsg@debian:~/gyakorlat/db/migrate> rails db:migrate
== 20211026113918 Attachments: migrating
   ======================================
-- create_table(:attachments)
   -> 0.0255s
== 20211026113918 Attachments: migrated (0.0301s)
   =============================
```

A méhlegelők és a megfigyelések modell osztályában felvesszük a kapcsolatot a csatolmányok felé. Mindkét modell egy-több kapcsolatban áll a csatolmányokkal, és a migrációban megadott fiktív idegen kulcsra hivatkozik a kapcsolat.

```
class Beepasture < ApplicationRecord
  has_many :attachments, as: :content
end
```

```
class Sighting < ApplicationRecord
  has_many :attachments, as: :content
end
```

2.2. Kezdeti adatok felvétele

Az adatmodellünk készen van, de nem tudjuk ellenőrizni, hogy jó-e, mert az adatbázisban nincs adatunk. Adatok felvételére több mód is van. A leglassabb a webfelület használata, ennél gyorsabb a Rails konzolon való adatrögzítés. Ha azt szeretnénk, hogy a konzolon felvett adatok reprodukálhatóan meglegyenek, akkor a konzolba írandó utasásokat a `db/seed.rb` fájlban helyezzük el. Vegyük fel tageket.

```
    u = User.create username: 'senki', email: 'senki@mail.bme.hu', password:
        'titok'
```

Töltsük be ezeket az adatokat:

```
kovacsg@debian:~/gyakorlat/db# rails db:seed
```