Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2016. április 5.

Az előző gyakorlaton megkezdett példát folytatjuk a felhasználói sessionök kezelésének megvalósításával, illetve az adatmodell kialakításával. Az előző alkalommak négy modellt hoztunk létre, a felhasználók `User` nevű modelljét, a kérdőívek `Questionnaire` nevű modelljét, a kérdések `Question` nevű modelljét és a válaszok `Answer` nevű modelljét. A felhasználók modellje és példányai a következőképp néznek ki az adatbázisban.

```
kovacsg@debian:~/gyakorlat/db$ rails db
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int(11)      | NO   | PRI | NULL    | auto_increment |
| username   | varchar(255) | YES  |     | NULL    |                |
| password   | varchar(255) | YES  |     | NULL    |                |
| email      | varchar(255) | YES  |     | NULL    |                |
| created_at | datetime     | NO   |     | NULL    |                |
| updated_at | datetime     | NO   |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)

mysql> select * from users;
+----+----------+----------+------------------+---------------------+---------------------+

| id | username | password | email            | created_at          |
    updated_at          |
+----+----------+----------+------------------+---------------------+---------------------+

|  1 | Valaki   | titok    | valaki@mail.bme.hu | 2016-09-13 19:35:23 |
    2016-09-13 19:35:23 |
+----+----------+----------+------------------+---------------------+---------------------+

1 row in set (0.00 sec)
```

Először nézzük meg, hogy milyen elemi műveletekkel férhetünk ehhez hozzá Rails-ből. A `all` függvény (1. sor) az modell összes rekordját visszaadja egy tömbben. A `first` (2. sor), `last` (3. sor) és `take` (4. 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. sor).

```
kovacsg@debian:~/gyakorlat/db$ rails c
Loading development environment (Rails 5.0.0.1)
irb(main):001:0> User.all
  User Load (0.4ms)  SELECT `users`.* FROM `users`
=> #<ActiveRecord::Relation [#<User id: 1, username: "Valaki", password: "
    titok", email: "valaki@mail.bme.hu", created_at: "2016-09-13 19:35:23",
    updated_at: "2016-09-13 19:35:23">]>
irb(main):002:0> User.first
  User Load (0.5ms)  SELECT  `users`.* FROM `users` ORDER BY `users`.`id`
     ASC LIMIT 1
=> #<User id: 1, username: "Valaki", password: "titok", email: "valaki@mail.
    bme.hu", created_at: "2016-09-13 19:35:23", updated_at: "2016-09-13
    19:35:23">
irb(main):003:0> User.last
  User Load (1.1ms)  SELECT  `users`.* FROM `users` ORDER BY `users`.`id`
     DESC LIMIT 1
=> #<User id: 1, username: "Valaki", password: "titok", email: "valaki@mail.
    bme.hu", created_at: "2016-09-13 19:35:23", updated_at: "2016-09-13
    19:35:23">
irb(main):004:0> User.take
  User Load (20.4ms)  SELECT  `users`.* FROM `users` LIMIT 1
=> #<User id: 1, username: "Valaki", password: "titok", email: "valaki@mail.
    bme.hu", created_at: "2016-09-13 19:35:23", updated_at: "2016-09-13
    19:35:23">
irb(main):005:0> User.take(2)
  User Load (0.7ms)  SELECT  `users`.* FROM `users` LIMIT 2
=> [#<User id: 1, username: "Valaki", password: "titok", email: "valaki@mail
    .bme.hu", created_at: "2016-09-13 19:35:23", updated_at: "2016-09-13
    19:35:23">]
irb(main):006:0>
```

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/db# rails g migration AddSaltToUsers salt:string
      invoke  active_record
      create    db/migrate/20161025101806_add_salt_to_users.rb
```

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[5.0]
```

```
  #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
    remove_column :users, :salt
    rename_column :users, :encrypted_password, :password
  end
end
```

Ezután hajtsuk végre a migrációt!

```
kovacsg@debian:~/gyakorlat/app/models$ rake db:migrate
(in /home/kovacsg/gyakorlat)
== 20161025101806 AddSaltToUsers: migrating
   ===================================
-- add_column(:users, :salt, :string)
   -> 0.2681s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.1602s
== 20161025101806 AddSaltToUsers: migrated (0.4286s)
   ==========================
```

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.

Nézzük meg, milyen egyéb módunk van hash kulcsként használható hexadecimális vagy Base64 string előállítására Railsben (3-6. sor). 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`.

```
kovacsg@debian:~/gyakorlat/app/models$ rails c
Loading development environment (Rails 5.0.0.1)
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.new_record?
=> true
irb(main):003:0> SecureRandom.hex(8)
=> "9ec1b119cf22ce8b"
irb(main):004:0> SecureRandom.hex(16)
=> "67f4713858cada60d6c24fd86168e376"
irb(main):005:0> SecureRandom.base64(16)
=> "j/CdOf4ydUI+MKyyW+7YjA=="
irb(main):006:0> Digest::SHA1.hexdigest("lalal")
=> "cb46b325ed3192835c8eecc0a4cb1fd5ada015f4"
irb(main):007:0>
```

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.

```
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` példányváltozótót `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(password, salt)
    Digest::SHA1.hexdigest(salt+password)
  end

  def encrypt_password
    return if password.blank?
    if new_record?
      self.salt = Digest::SHA1.hexdigest(Time.now.to_s + self.username)
    end
    self.encrypted_password = User.encrypt(self.password, self.salt)
  end
end
```

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/views/layouts$ rails g controller sessions
    create destroy
      create  app/controllers/sessions_controller.rb
       route  get 'sessions/destroy'
       route  get 'sessions/create'
      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    coffee
      create      app/assets/javascripts/sessions.coffee
      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 vendégmenüben, vagyis a `layouts/_guestmenu.html.erb`-ben a form akciója a `/sessions/create`-re mutat-e, illetve a belépett felhasználó menüjében (`layouts/application.html.erb`) 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.

A következő lépés a felhasználó hitelesítésének megvalósítása, amit a `User` modellben teszünk meg egy osztálymetódussal. A hitelesítés két argumentummal rendelkezik egy felhasználónévvel 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. A vendégfelhasználó menüjében eredetileg felhasználónevet kértünk, módosítsuk, hogy ezután email címet kérjünk.

```
class User < ApplicationRecord
  def authenticated?(password)
    self.encrypted_password == User.encrypt(password, self.salt)
  end

  def self.authenticate(username, password)
    user = User.find_by username: username
    user && user.authenticated?(password) ? user : nil
  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.

```
class SessionsController < ApplicationController
  def create
    @user = User.authenticate(params[:username], params[:password])
    if @user
      session[:user] = @user.id
    else
      flash[:notice] = 'Invalid user name or password'
    end
    redirect_to :back
  end

  def destroy
    reset_session
    flash[:notice] = 'Logged out successfully'
    redirect_to :back
  end
end
```

A flash hashen keresztül értéket adhatunk át a következő HTTP kérésre adott válasz számára. A megjelenítendő üzenet helye legyen az központi nézetben és a `_guestmenu.html.erb`-ben.

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

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
```

Felhasználó létrehozásához és adatainak módosításához szükséges nézeteket már létrehoztuk, valósítsuk meg a formokat kezelő kontroller akciókat. A regisztrációhoz a `users` kontorller `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_to questionnaires_path
    else
      flash[:notice] = "Error while registering"
      redirect_to action: :new
    end
  end

  def update
    if @user.update(user_params)
      flash[:notice] = "Update successful"
    else
      flash[:notice] = "Could not update user data"
    end
    redirect_to :back
  end

  private
    def user_params
      params.require(:user).permit(:username, :email, :password, :
          password_confirmation)
    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'
  post 'users/create', to: 'users#create'
  get 'users/edit/:id', to: 'users#edit', as: 'edit_user'
  put 'users/update/:id', to: 'users#update', as: 'update_user'
  get 'users/show/:id', to:  'users#show', as: 'show_user'
  get 'users/forgotten'
  post 'users/send_forgotten', to: 'users#send_forgotten'
```

Az `id` 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. Mivel több függvény előtt ugyanazt a keresést kellene elvégeznünk, egy privát callback függvényt definiálunk a `before_filter` segítségével, ami minden kontroller esetén lefut, mert azt a kontrollerek ősosztályában, az `ApplicationController`-ben definiáljuk. A keresést csak akkor végezzük el, ha van aktív felhasználói

session.

```
class ApplicationController < ActionController::Base
  before_action :find_user, only: [:edit, :update, :show]

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

A regisztrációkor az elmentendő felhasználót még az elmentés előtt validáljuk, az felhasználónév attribútumnak nemüresnek kell lennie (`:presence`), 4 és 18 közötti számú karakterből kell állnia, és egyedinek (`:uniqueness`) kell lennie , és ha ez nem teljesül, akkor visszajelzünk, hogy nem megfelelő felhasználónévről van szó. Az email címet 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 :username,
    {
      presence: true,
      uniqueness: true,
      length: { in: 4..18 }
    }
  validates :email,
    presence: true
  validates :password, confirmation: true, if: :password_required?

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

Nézzük meg először Rails konzolon, hogy az `encrypt` osztálymetódus képes-e a jelszót titkosítani! Mivel úgy látjuk, hogy működik, hozzunk létre egy felhasználót, állítsuk be az attribútumait és mentsük el az adatbázisba! Mentés után az `id` attribútum inicializálódik az adatbázisbeli értékre, valamint a `salt` és `encrypted_password` attribútumok ember számára értelmezlen szöveggel kerülnek kitöltésre. Végül nézzük meg, hogy a `authenticate` osztálymetódus megtalálja-e a felhasználót email cím és jelszó alapján! Ha igen, akkor a bejelentkezéssel való kísérletezést folytathatjuk a webfelületen.

```
kovacsg@debian:~/gyakorlat/app/models$ rails c
Loading development environment (Rails 5.0.0.1)
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.username='senki'
=> "senki"
irb(main):003:0> u.email='senki@mail.bme.hu'
=> "senki@mail.bme.hu"
irb(main):004:0> u.password = 'titok'
=> "titok"
irb(main):005:0> u.save
   (0.3ms)  BEGIN
  SQL (23.5ms)  INSERT INTO `users` (`username`, `encrypted_password`, `
      email`, `created_at`, `updated_at`, `salt`) VALUES ('senki', '
      faba9189829a2a08075f734fea3bd746a1b05fee', 'senki@mail.bme.hu', '
      2016-10-25 10:41:47', '2016-10-25 10:41:47', '8706
      f50d58976d2888eefa3a66c7e1b0b82667cc')
   (149.2ms)  COMMIT
=> true
kovacsg@debian:~/gyakorlat/app/models$ rails c
Loading development environment (Rails 5.0.0.1)
irb(main):006:0> u
=> #<User id: 2, username: "senki", encrypted_password: "
    faba9189829a2a08075f734fea3bd746a1b05fee", email: "senki@mail.bme.hu",
    created_at: "2016-10-25 10:41:47", updated_at: "2016-10-25 10:41:47",
    salt: "8706f50d58976d2888eefa3a66c7e1b0b82667cc">
irb(main):007:0> u.new_record?
=> false
irb(main):002:0> User.find_by username: 'senki'
  User Load (0.6ms)  SELECT  `users`.* FROM `users` WHERE `users`.`username`
      = 'senki' LIMIT 1
=> #<User id: 2, username: "senki", encrypted_password: "
    faba9189829a2a08075f734fea3bd746a1b05fee", email: "senki@mail.bme.hu",
    created_at: "2016-10-25 10:41:47", updated_at: "2016-10-25 10:41:47",
    salt: "8706f50d58976d2888eefa3a66c7e1b0b82667cc">
irb(main):003:0> User.find 2
  User Load (1.7ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 2
     LIMIT 1
=> #<User id: 2, username: "senki", encrypted_password: "
    faba9189829a2a08075f734fea3bd746a1b05fee", email: "senki@mail.bme.hu",
    created_at: "2016-10-25 10:41:47", updated_at: "2016-10-25 10:41:47",
    salt: "8706f50d58976d2888eefa3a66c7e1b0b82667cc">
irb(main):004:0>
kovacsg@debian:~/gyakorlat/app/models$ rails c
Loading development environment (Rails 5.0.0.1)
irb(main):001:0> User.find_by username: 'lalala'
  User Load (22.4ms)  SELECT  `users`.* FROM `users` WHERE `users`.`username
      ` = 'lalala' LIMIT 1
=> nil
```

Konzolon és a webfelületen ellenőrizhetjük, hogy elmenthető-e létező felhasználónévvel egy új rekord, illetve, hogy a felhasználónév hosszának megsértése hibával jár-e! 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/app/models$ rails c
Loading development environment (Rails 5.0.0.1)
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.username = 'senki'
```

```
=> "senki"
irb(main):003:0> u.save
   (0.2ms)  BEGIN
  User Exists (26.2ms)  SELECT  1 AS one FROM `users` WHERE `users`.`
      username` = BINARY 'senki' LIMIT 1
   (0.2ms)  ROLLBACK
=> false
irb(main):005:0> u.errors
=> #<ActiveModel::Errors:0x00000002f297a0 @base=#<User id: nil, username: "
    senki", encrypted_password: nil, email: nil, created_at: nil, updated_at
    : nil, salt: nil>, @messages={:username=>["has already been taken"], :
    email=>["can't be blank"]}, @details={:username=>[{:error=>:taken, :
    value=>"senki"}], :email=>[{:error=>:blank}]}>
irb(main):006:0> u.errors.messages
=> {:username=>["has already been taken"], :email=>["can't be blank"]}
irb(main):007:0> u.username = 'se'
=> "se"
irb(main):008:0> u.email = 'se@mail.bme.hu'
=> "se@mail.bme.hu"
irb(main):009:0> u.save
   (0.6ms)  BEGIN
  User Exists (0.8ms)  SELECT  1 AS one FROM `users` WHERE `users`.`username
      ` = BINARY 'se' LIMIT 1
   (0.9ms)  ROLLBACK
=> false
irb(main):010:0> u.errors.messages
=> {:username=>["is too short (minimum is 4 characters)"]}
irb(main):011:0>
```

A felhasználói regisztráció 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.

```
kovacsg@debian:~/gyakorlat/app/models$ rails c
Loading development environment (Rails 5.0.0.1)
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.password = 'a'
=> "a"
```

```
irb(main):003:0> u.password_confirmation = 'a'
=> "a"
irb(main):004:0>
kovacsg@debian:~/gyakorlat/app/controllers$ rails c
Loading development environment (Rails 5.0.0.1)
irb(main):001:0> u = User.find 3
  User Load (0.2ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 3
      LIMIT 1
=> #<User id: 3, username: "valaki", encrypted_password: "79
    ebde9d2eb0ba07c2e5424b3865c356b8c597c5", email: "barki@mail.bme.hu",
    created_at: "2016-10-25 11:26:07", updated_at: "2016-10-25 11:37:31",
    salt: "4ade6b2ac91cf0ffa0a997a7f6c9419a02450eff">
irb(main):002:0> u.username= 'senki'
=> "senki"
irb(main):003:0> u.save
   (0.2ms)  BEGIN
  User Exists (22.5ms)  SELECT  1 AS one FROM `users` WHERE `users`.`
      username` = BINARY 'senki' AND (`users`.`id` != 3) LIMIT 1
   (0.6ms)  ROLLBACK
=> false
irb(main):004:0> u.username= 'barki'
=> "barki"
irb(main):005:0> u.save
   (0.8ms)  BEGIN
  User Exists (1.0ms)  SELECT  1 AS one FROM `users` WHERE `users`.`username
      ` = BINARY 'barki' AND (`users`.`id` != 3) LIMIT 1
  SQL (1.8ms)  UPDATE `users` SET `username` = 'barki', `updated_at` = '
      2016-10-25 11:38:19' WHERE `users`.`id` = 3
   (112.5ms)  COMMIT
=> true
irb(main):006:0> u.username= 'valaki'
=> "valaki"
irb(main):007:0> u.save
   (0.4ms)  BEGIN
  User Exists (0.5ms)  SELECT  1 AS one FROM `users` WHERE `users`.`username
      ` = BINARY 'valaki' AND (`users`.`id` != 3) LIMIT 1
  SQL (2.1ms)  UPDATE `users` SET `username` = 'valaki', `updated_at` = '
      2016-10-25 11:38:23' WHERE `users`.`id` = 3
   (4.7ms)  COMMIT
=> true
irb(main):008:0> User.find 1
  User Load (1.0ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 1
      LIMIT 1
=> #<User id: 1, username: "Valaki", encrypted_password: "titok", email: "
    valaki@mail.bme.hu", created_at: "2016-09-13 19:35:23", updated_at:
    "2016-09-13 19:35:23", salt: nil>
irb(main):009:0>
```

A hibaüzenetet testreszabhatjuk a lokalizációs beállításokban:

```
  activerecord:
    errors:
      models:
        user:
          attributes:
            username:
              blank: 'Empty username'
              taken: 'Username already taken'
```

Az előző gyakorlaton létrehoztuk a kérdőívhez kapcsolódó modelleket. Azt látjuk bennük, hogy az idegen kulcsokhoz automatikusan létrejöttek az

ActiveRecord asszociációk, mert a `references` értéket adtuk meg az attribútum típusaként.

```
class Questionnaire < ApplicationRecord
  belongs_to :user
end
class Question < ApplicationRecord
  belongs_to :questionnaire
end
class Answer < ApplicationRecord
  belongs_to :question
end
```

A relációk csak a több-egy irányban navigálhatók alapértelmezés szerint, a fordított irányú navigációhoz fel kell vennünk a `has_many` helpereket.

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

Hozzunk létre egy új kérdőívet (2-4. sor), asszociáljuk az egy létező felhasználóhoz (5-6. sor), és mentsük el az adatbázisunkba (8. sor), majd nézzük meg, hogyan működnek az asszociációk (9-10. sor, és az új konzol 2. sora)!

```
kovacsg@debian:~/gyakorlat/db$ rails c
Loading development environment (Rails 5.0.0.1)
irb(main):002:0> q = Questionnaire.new
=> #<Questionnaire id: nil, title: nil, description: nil, user_id: nil,
    created_at: nil, updated_at: nil>
irb(main):003:0> q.title = 'Ime, az elso'
=> "Ime, az elso"
irb(main):004:0> q.description = "Az elso kerdoivem hosszu-hosszu leirasa"
=> "Az elso kerdoivem hosszu-hosszu leirasa"
irb(main):005:0> q.user_id = 3
=> 3
irb(main):006:0> u = User.find 3
  User Load (0.5ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 3
      LIMIT 1
=> #<User id: 3, username: "valaki", encrypted_password: "79
    ebde9d2eb0ba07c2e5424b3865c356b8c597c5", email: "barki@mail.bme.hu",
    created_at: "2016-10-25 11:26:07", updated_at: "2016-10-25 11:38:23",
    salt: "4ade6b2ac91cf0ffa0a997a7f6c9419a02450eff">
irb(main):007:0> q.user = u
=> #<User id: 3, username: "valaki", encrypted_password: "79
    ebde9d2eb0ba07c2e5424b3865c356b8c597c5", email: "barki@mail.bme.hu",
    created_at: "2016-10-25 11:26:07", updated_at: "2016-10-25 11:38:23",
    salt: "4ade6b2ac91cf0ffa0a997a7f6c9419a02450eff">
irb(main):008:0> q.save
   (0.2ms)  BEGIN
  SQL (26.4ms)  INSERT INTO `questionnaires` (`title`, `description`, `
      user_id`, `created_at`, `updated_at`) VALUES ('Ime, az elso', 'Az elso
      kerdoivem hosszu-hosszu leirasa', 3, '2016-10-25 11:44:28', '
      2016-10-25 11:44:28')
   (3.9ms)  COMMIT
=> true
irb(main):009:0> q = Questionnaire.first
  Questionnaire Load (1.3ms)  SELECT  `questionnaires`.* FROM `
      questionnaires` ORDER BY `questionnaires`.`id` ASC LIMIT 1
```

```
=> #<Questionnaire id: 1, title: "Ime, az elso", description: "Az elso
    kerdoivem hosszu-hosszu leirasa", user_id: 3, created_at: "2016-10-25
    11:44:28", updated_at: "2016-10-25 11:44:28">
irb(main):010:0> q.user
  User Load (0.3ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 3
    LIMIT 1
=> #<User id: 3, username: "valaki", encrypted_password: "79
    ebde9d2eb0ba07c2e5424b3865c356b8c597c5", email: "barki@mail.bme.hu",
    created_at: "2016-10-25 11:26:07", updated_at: "2016-10-25 11:38:23",
    salt: "4ade6b2ac91cf0ffa0a997a7f6c9419a02450eff">
irb(main):011:0>
kovacsg@debian:~/gyakorlat/app/models$ rails c
Loading development environment (Rails 5.0.0.1)
irb(main):001:0> u = User.find 3
  User Load (0.2ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 3
    LIMIT 1
=> #<User id: 3, username: "valaki", encrypted_password: "79
    ebde9d2eb0ba07c2e5424b3865c356b8c597c5", email: "barki@mail.bme.hu",
    created_at: "2016-10-25 11:26:07", updated_at: "2016-10-25 11:38:23",
    salt: "4ade6b2ac91cf0ffa0a997a7f6c9419a02450eff">
irb(main):002:0> u.questionnaires
  Questionnaire Load (0.5ms)  SELECT `questionnaires`.* FROM `questionnaires
    ` WHERE `questionnaires`.`user_id` = 3
=> #<ActiveRecord::Associations::CollectionProxy [#<Questionnaire id: 1,
    title: "Ime, az elso", description: "Az elso kerdoivem hosszu-hosszu
    leirasa", user_id: 3, created_at: "2016-10-25 11:44:28", updated_at:
    "2016-10-25 11:44:28">]>
irb(main):003:0>
```

Az előző gyakorlaton adatbázis nélkül inicializáltuk ezeket a nézeteket. Most már ezt helyre tehetjük a kontrollerben. Töröljük ki a múltkor hozzáadott kódrészleteket, és állítsuk vissza a kikommentezett viselkedést. Ez a `questionnaires_controller` `index` és `set_questionnaire`, a `questions_controller` `index` és `set_question`, az `answers_controller` `index` és `set_answer`, valamint a `users_controller` metódusait érinti.

Az adatbázisunkban szükségünk lesz kezdeti adatokra, például egy adminisztrátorra, akit a `db/seeds.rb` fájlban elhelyezett Ruby metódushívásokkal tudunk megfelvenni. Ide pontosan azon utasításoknak kell szerepelniük, amiket a konzolon kiadnánk adatok rögzítésekor. Ha az adatokat ebben a fájban adjuk meg, akkor az az adatbázis újrainicializálása után mindig helyreállítható lesz. Vegyünk ezen kívül fel még adatokat, amik a nézetek fejlesztésésben játszanak majd szerepet.

```
u = User.create username: 'thomas', email: 'tom@mail.bme.hu', password: 'a',
    encrypted_password: 'a'
```

Az adatokatokat ezek megadása után a következő paranccsal tudjuk betölteni az adatbázisba:

```
rake db:seed
```