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 két modellt hoztunk létre, a felhasználók `User` nevű modelljét, a szavak `Word` nevű modelljét, amelyek a következő táblákat hozták létre az adatbázisban.

```
mysql> describe users;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| id             | int(11)      | NO   | PRI | NULL    | auto_increment |
| email          | varchar(255) | YES  |     | NULL    |                |
| name           | varchar(255) | YES  |     | NULL    |                |
| password       | varchar(255) | YES  |     | NULL    |                |
| birthdate      | datetime     | YES  |     | NULL    |                |
| account_number | varchar(255) | YES  |     | NULL    |                |
| created_at     | datetime     | NO   |     | NULL    |                |
| updated_at     | datetime     | NO   |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
8 rows in set (0.00 sec)

mysql> describe posts;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int(11)      | NO   | PRI | NULL    | auto_increment |
| text       | varchar(255) | YES  |     | NULL    |                |
| author     | int(11)      | YES  |     | NULL    |                |
| user_id    | int(11)      | YES  | MUL | NULL    |                |
| created_at | datetime     | NO   |     | NULL    |                |
| updated_at | datetime     | NO   |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
6 rows in set (0.00 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. Ha már módosítjuk a felhasználók modelljék, akkor adjuk hozzá azt az attribútumot is, amiben a felhasználó típusát tároljuk. Módosítsuk ezen kívül az üzenetek táblájában a `author` attribútumot a konvencióknak megfelelő `author_id` nevűre, valamint hozzunk létre egy kapcsolótáblát a felhasználók közötti barát reláció tárolására.

```
kovacsg@debian:~/gyakorlat/db# rails g migration AddSaltToUsers salt:string
      invoke  active_record
      create    db/migrate/20160405101910_add_salt_to_users.rb
kovacsg@debian:~/gyakorlat/db/migrate# rails g migration RenameAuthorInPosts
      invoke  active_record
      create    db/migrate/20160405102437_rename_author_in_posts.rb
kovacsg@debian:~/gyakorlat/db/migrate# rails g migration
    CreateJoinTableFriendUser user friend
      invoke  active_record
      create    db/migrate/20160405102628_create_join_table_friend_user.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
  #def change
  #  reversible do |dir|
  #  dir.up { add_column ... }
  #  dir.down { ... }
  #  end
  #end
  def up
    add_column :users, :salt, :string
    rename_column :users, :password, :encrypted_password
# Az adatbazisban levo rekordokkal csinaljunk valamit
  end

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

A `posts` táblát módosító migráció:

```
class RenameAuthorInPosts < ActiveRecord::Migration
  def up
    rename_column :posts, :author, :author_id
  end

  def down
    rename_column :posts, :author_id, :author
```

```
  end
end
```

A kapcsolótáblát létrehozó migráció:

```
class CreateJoinTableFriendUser < ActiveRecord::Migration
  def change
    create_join_table :users, :friends do |t|
      # t.index [:user_id, :friend_id]
      # t.index [:friend_id, :user_id]
    end
  end
end
```

Ezután hajtsuk végre a migrációkat!

```
kovacsg@debian:~/gyakorlat/db/migrate# rake db:migrate
(in /home/kovacsg/gyakorlat)
== 20160405101910 AddSaltToUsers: migrating
   ===================================
-- add_column(:users, :salt, :string)
   -> 0.6788s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0867s
== 20160405101910 AddSaltToUsers: migrated (0.7659s)
   ============================

== 20160405102437 RenameAuthorInPosts: migrating
   ===================================
-- rename_column(:posts, :author, :author_id)
   -> 0.0683s
== 20160405102437 RenameAuthorInPosts: migrated (0.0687s)
   ============================

== 20160405102628 CreateJoinTableFriendUser: migrating
   ===================================
-- create_join_table(:users, :friends)
   -> 0.0277s
== 20160405102628 CreateJoinTableFriendUser: migrated (0.0278s)
   ============================
```

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 kulcs előállítására Railsben.

```
kovacsg@debian:~/gyakorlat/app/models# rails c
Loading development environment (Rails 4.2.6)
irb(main):001:0> SecureRandom.hex(16)
=> "5cb87c52c72717c73bba7db38f545b23"
irb(main):002:0> SecureRandom.base64(16)
=> "vtH4bxNxNXoHp/ABlkt+Kw=="
irb(main):003:0> Digest::SHA1.hexdigest('titok')
=> "46ff53e764c4acf97b54db2020573049d2e3dab3"
irb(main):004:0> Digest::SHA1.hexdigest('titok'+SecureRandom.hex(16))
=> "28b778898418abab910f202e6ca2f6621776d7ee"
```

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 < ActiveRecord::Base
  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` példányváltozót `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 < ActiveRecord::Base
  before_save :encrypt_password

  def self.encrypt(pass, salt)
    Digest::SHA2.hexdigest(salt+pass)
  end

  def encrypt_password
    return if password.blank?
    if new_record?
        self.salt = SecureRandom.base64(8)
    end
    self.encrypted_password = User.encrypt(password, 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/config# 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 `layouts/_menu.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 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. A vendégfelhasználó menüjében eredetileg felhasználónevet kértünk, módosítsuk, hogy ezután email címet kérjünk.

```
class User < ActiveRecord::Base
  def authenticated?(pass)
    encrypted_password == User.encrypt(pass, salt)
  end

  def self.authenticate(email, pass)
    user = User.where(email: email).take
    # user = User.find_by username: user
    user && user.authenticated?(pass) ? user : nil
  end
end
```

A kontrollerünk ezek után a következőképp néz ki. A hitelesítés imént megírt metódusának visszatérési értékét a `current_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 ve-

szélyes 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 SessionController < ApplicationController
  def create
    @current_user = User.authenticate(params[:email], params[:password])
    if @current_user
      session[:user] = @current_user.id
      redirect_to :back
    else
      flash[:notice] = "Invalid username or password"
      redirect_to :back
    end
  end

  def destroy
    reset_session
    flash[:notice] = "Logged out successfully"
    redirect_to '/say/hello'
  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 `_menu.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(params_user)
    if @user.save
      @current_user = @user
      session[:user] = @user.id
      flash[:notice] = "Logged in successfully"
      redirect_to action: :show
    else
      flash[:notice] = "Errors occurred"
      render :new
      return
    end
  end

  def update
    if @user.update(user_params)
      flash[:notice] = 'Successful update'
      redirect_to :back
    else
      flash[:notice] = @user.errors.messages
      redirect_to :back
    end
  end

  private
    def params_user
      params.require(:user).permit([:email, :name, :birthdate, :password, :
          password_confirmation, :account_number])
    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', as: 'create_user'
  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'
  delete 'users/destroy/:id', to: 'users#destroy', as: 'destroy_user'
  get 'users/index'
  get 'users/forgotten'
post '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_filter :load_user

  protected
    def load_user
      if session[:user]
        @user = User.find session[:user]
      end
    end
end
```

A regisztrációkor az elmentendő felhasználót még az elmentés előtt validáljuk, az email cím attribútumnak nemüresnek (`:presence`) és egyedinek kell lennie (`:uniqueness`), és ha ez nem teljesül, akkor visszajelzünk, hogy nem megfelelő email címről van szó. 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 :email,
    {
      presence: true,
      uniqueness: true
    }
  validates :name, presence: true
  validates :password, confirmation: true, :if => :password_required?

  def password_required?
    !password.blank? || encrypted_password.blank? # new_record?
  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 4.2.6)
irb(main):012:0> User.encrypt 'titok', 'valami'
=> "3adf7eb2062cef0ac465b7673ca7b7b9df1b3c7267f16fad2d13e8f05e5e8759"

irb(main):003:0> u = User.new email: 'senki@mail.bme.hu', name: 'Sen Ki',
    birthdate: Time.now - 20.years
=> #<User id: nil, email: "senki@mail.bme.hu", name: "Sen Ki",
    encrypted_password: nil, birthdate: "1996-04-05 10:40:47",
    account_number: nil, created_at: nil, updated_at: nil, salt: nil>
irb(main):004:0> u.password = 'titok'
=> "titok"
irb(main):005:0> u.save
   (2.7ms)  BEGIN
  SQL (0.5ms)  INSERT INTO `users` (`email`, `name`, `birthdate`, `salt`, `
      encrypted_password`, `created_at`, `updated_at`) VALUES ('senki@mail.
      bme.hu', 'Sen Ki', '1996-04-05 10:40:47', 'mXQUUNuXbJY=', '
      d729c0ce15b7142826956eeb59e7132c03132a3b27bc84d935eba33dc2c96ec7', '
      2016-04-05 10:41:00', '2016-04-05 10:41:00')
   (67.3ms)  COMMIT
=> true
irb(main):006:0> u
=> #<User id: 2, email: "senki@mail.bme.hu", name: "Sen Ki",
    encrypted_password: "d729c0ce15b7142826956eeb59e7132c03132a3b27bc84d935
    ...", birthdate: "1996-04-05 10:40:47", account_number: nil, created_at:
     "2016-04-05 10:41:00", updated_at: "2016-04-05 10:41:00", salt: "
    mXQUUNuXbJY=">
irb(main):009:0> u.new_record?
=> false
irb(main):010:0> u
=> #<User id: 2, email: "senki@mail.bme.hu", name: "Sen Ki",
    encrypted_password: "d729c0ce15b7142826956eeb59e7132c03132a3b27bc84d935
    ...", birthdate: "1996-04-05 10:40:47", account_number: nil, created_at:
     "2016-04-05 10:41:00", updated_at: "2016-04-05 10:41:00", salt: "
    mXQUUNuXbJY=">
irb(main):002:0> User.authenticate u.email, 'titok'
  User Load (1.0ms)  SELECT `users`.* FROM `users` WHERE `users`.`email` =
      'senki@mail.bme.hu' LIMIT 1
=> #<User id: 2, email: "senki@mail.bme.hu", name: "Sen Ki",
    encrypted_password: "d729c0ce15b7142826956eeb59e7132c03132a3b27bc84d935
    ...", birthdate: "1996-04-05 10:40:47", account_number: nil, created_at:
     "2016-04-05 10:41:00", updated_at: "2016-04-05 10:41:00", salt: "
    mXQUUNuXbJY=">
```

Konzolon és a webfelületen ellenőrizhetjük, hogy elmenthető-e létező email címmel egy rekord, illetve, hogy a jelszó és annak megerősítésének meg kell egyeznie! 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 4.2.6)
irb(main):001:0> u = User.new email: 'senki@mail.bme.hu'
=> #<User id: nil, email: "senki@mail.bme.hu", name: nil, encrypted_password
    : nil, birthdate: nil, account_number: nil, created_at: nil, updated_at:
     nil, salt: nil>
irb(main):002:0> u.valid?
```

```
  User Exists (27.9ms)  SELECT  1 AS one FROM `users` WHERE `users`.`email`
      = BINARY 'senki@mail.bme.hu' LIMIT 1
=> false
irb(main):003:0> u.errors.messages
=> {:email=>["has already been taken"], :name=>["can't be blank"]}
```

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

```
<h1>Register</h1>
  <% if @user.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@user.errors.count, "error") %> prohibited this post
          from being sav
ed:</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. A hibás adatok terjedjenek ki az üres email címre, a foglal email címre, az üres névre és a nem egyező jelszavakra. Láthatjuk, hogy ezek a műveletek nem hajtódnak végre, és a Rails megjelöli a hibás beviteli mezőket.

A hibaüzenetet testreszabhatjuk a lokalizációs beállításokban:

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

Ezután a validációs üzenet is megváltozik:

```
kovacsg@debian:~/gyakorlat/config/locales# rails c
Loading development environment (Rails 4.2.6)
irb(main):001:0> u = User.new email: 'senki@mail.bme.hu'
=> #<User id: nil, email: "senki@mail.bme.hu", name: nil, encrypted_password
    : nil, birthdate: nil, account_number: nil, created_at: nil, updated_at:
     nil, salt: nil>
irb(main):002:0> u.valid?
  User Exists (0.3ms)  SELECT  1 AS one FROM `users` WHERE `users`.`email` =
      BINARY 'senki@mail.bme.hu' LIMIT 1
=> false
irb(main):003:0> u.errors.messages
=> {:email=>["Email already taken"], :name=>["can't be blank"]}
```

Az előző gyakorlaton létrehoztuk az üzenetek (`Post`) modellt. Azt látjuk benne, hogy két idegen kulcsunk van, de mindkettő ugyanazon modell felé, a `User` felé. A két relációt az idegen kulcs alapján különböztetjük meg.

A `author` idegen kulcsot az óra elején módosítottuk, hogy az megfeleljen a konvencióknak. Az modell osztályok közötti reláció az üzenet osztályból a felhasználók osztály felé az alábbi módon hozható létre. A `belongs_to` idegen kulcs jelenlétét jelzi. Az `user` egy-több reláció, és az az alapján lérejövő azonos nevű setter/getter páros az `User` osztály egy példányára fog hivatkozni. Az `author` deklarációja azonos nevű setter/getter párost hoz létre, de itt explicite jeleznünk kell, hogy az `author_id` a `posts` táblában egy idegen kulcs, ami a `users` tábla egy rekordjára hivatkozik. A `Post` modellünk számára írjuk elő, hogy az üzenet szövege nem lehet üres. Az üzenet küldője az aktuálisan bejelenkezett felhasználó lesz, a fogadója pedig az a felhasználó, akinek az oldalán épp tartózkodik a bejelentkezett felhasználó, így azok garantáltan nem vehetnek fel definiálatlan értéket.

```
class Post < ActiveRecord::Base
  belongs_to :author, class_name: 'User', foreign_key: :author_id
  belongs_to :user

  validates :text, presence: true
end
```

Ugyanez a két reláció a `User` modellből nézve a következőképp valósítható meg. A felhasználó üzenőfalán lévő üzeneteket a `posts` egy több reláció enumerációt beállító/visszaadó settere/gettere teszi elérhetővé. A felhasználó által írt üzenetek helyzete – akárcsak a másik modell esetén – kicsit bonyolultabb. Ez is egy-több reláció, hiszen egy felhasználó sok kommentet tehet, de egy komment csak egy felhasználóé, ezért itt is a `has_many` metódust használjuk, azonban mivel ez nem a `Comment` nevű modellre hivatkozik, hanem a `Post` nevűre, azt osztálynév paraméterként nevesítjük, és megadjuk az ahhoz tartozó `posts` táblában lévő idegen kulcs nevét.

```
class User < ActiveRecord::Base
  has_many :posts
  has_many :comments, class_name: "Post", foreign_key: :author_id
end
```

Vegyünk fel egy másik felhasználót az adatbázisunkba a webfelületen, aki egy üzenetet ír a korábban felvett felhasználó számára, és nézzük meg, hogyan működnek ezek a relációkat megvalósító setterek/getterek:

```
kovacsg@debian:~/gyakorlat/app/models# rails c
Loading development environment (Rails 4.2.6)
irb(main):001:0> u = User.find 2
  User Load (0.5ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 2
      LIMIT 1
=> #<User id: 2, email: "senki@mail.bme.hu", name: "Sen Ki",
    encrypted_password: "d729c0ce15b7142826956eeb59e7132c03132a3b27bc84d935
    ...", birthdate: "1996-04-05 10:40:47", account_number: nil, created_at:
     "2016-04-05 10:41:00", updated_at: "2016-04-05 10:41:00", salt: "
    mXQUUNuXbJY=">
irb(main):002:0> v = User.find 3
```

```
  User Load (0.3ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 3
      LIMIT 1
=> #<User id: 3, email: "valaki3@mail.bme.hu", name: "Val Ki III",
    encrypted_password: "a4ee53650096b99fe935ee0d294fe56f2247230ce5fbd217f9
    ...", birthdate: "2011-04-05 00:00:00", account_number: "", created_at:
    "2016-04-05 11:25:29", updated_at: "2016-04-05 11:25:29", salt: "
    zcCj6CKYEdE=">
irb(main):003:0> p = Post.create user: u, author: v, text: 'Itt egy
    bejegyzes'
   (0.2ms)  BEGIN
  SQL (0.5ms)  INSERT INTO `posts` (`user_id`, `author_id`, `text`, `
      created_at`, `updated_at`) VALUES (2, 3, 'Itt egy bejegyzes', '
      2016-04-05 11:33:06', '2016-04-05 11:33:06')
   (180.4ms)  COMMIT
=> #<Post id: 1, text: "Itt egy bejegyzes", author_id: 3, user_id: 2,
    created_at: "2016-04-05 11:33:06", updated_at: "2016-04-05 11:33:06">
irb(main):004:0> p.user
=> #<User id: 2, email: "senki@mail.bme.hu", name: "Sen Ki",
    encrypted_password: "d729c0ce15b7142826956eeb59e7132c03132a3b27bc84d935
    ...", birthdate: "1996-04-05 10:40:47", account_number: nil, created_at:
     "2016-04-05 10:41:00", updated_at: "2016-04-05 10:41:00", salt: "
    mXQUUNuXbJY=">
irb(main):005:0> p.author
=> #<User id: 3, email: "valaki3@mail.bme.hu", name: "Val Ki III",
    encrypted_password: "a4ee53650096b99fe935ee0d294fe56f2247230ce5fbd217f9
    ...", birthdate: "2011-04-05 00:00:00", account_number: "", created_at:
    "2016-04-05 11:25:29", updated_at: "2016-04-05 11:25:29", salt: "
    zcCj6CKYEdE=">
irb(main):006:0> u.posts
  Post Load (23.7ms)  SELECT `posts`.* FROM `posts` WHERE `posts`.`user_id`
      = 2
=> #<ActiveRecord::Associations::CollectionProxy [#<Post id: 1, text: "Itt
    egy bejegyzes", author_id: 3, user_id: 2, created_at: "2016-04-05
    11:33:06", updated_at: "2016-04-05 11:33:06">]>
irb(main):007:0> u.comments
  Post Load (3.1ms)  SELECT `posts`.* FROM `posts` WHERE `posts`.`author_id`
      = 2
=> #<ActiveRecord::Associations::CollectionProxy []>
irb(main):008:0> v.comments
  Post Load (0.6ms)  SELECT `posts`.* FROM `posts` WHERE `posts`.`author_id`
      = 3
=> #<ActiveRecord::Associations::CollectionProxy [#<Post id: 1, text: "Itt
    egy bejegyzes", author_id: 3, user_id: 2, created_at: "2016-04-05
    11:33:06", updated_at: "2016-04-05 11:33:06">]>
```

Ezután hozzuk létre a barát relációt két `User` példány között. Ez egy több-több kapcsolat, egy felhasználónak sok barát felhasználója lehet, és egy barát felhasználó sok felhasználónak lehet a barátja. Az infrastruktúránk, vagyis a kapcsolótáblánk már létezik hozzá `friends_users` néven az adatbázisban. A több-több kapcsolatot vagy `has_many :through` típusú metódussal vagy `has_and_belongs_to_many` metódussal definiáljuk. Mivel itt a kapcsolat nevesítésére nincs szükségünk, ezért az utóbbit választjuk. Egy felhasználónak sok barátja lehet, amit a `friends` reláció és az azonos néven létrejövő setter/getter tesz elérhetővé. További paraméterként megadjuk, hogy a `friends` enumeráció `User` típusú objektumokat tartalmaz, az összerendelést a már említett kapcsolótábla valósítja meg, amiben két idegen kulcs

van, a `user_id` hivatkozik a felhasználóra, és a `friend_id` hivatkozik a baráti felhasználóra.

```
class User < ActiveRecord::Base
  has_and_belongs_to_many :friends,
    class_name: 'User',
    foreign_key: 'friend_id',
    associated_foreign_key: 'user_id',
    join_table: 'friends_users'
```

Nézzük meg ezt a viszonyt konzolon két felhasználó tekintetében. Láthatjuk, hogy a barátnak jelölés művelet nem szimmetrikus, ezért a kapcsolat létrehozásakor, miután mindkét fél jóváhagyta azt, kölcsönösen fel kell vennünk a felhasználókat a másik barátai közé.

```
kovacsg@debian:~/gyakorlat/app/models# rails c
Loading development environment (Rails 4.2.6)
irb(main):001:0> u = User.find 2
  User Load (0.3ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 2
      LIMIT 1
=> #<User id: 2, email: "senki@mail.bme.hu", name: "Sen Ki",
    encrypted_password: "d729c0ce15b7142826956eeb59e7132c03132a3b27bc84d935
    ...", birthdate: "1996-04-05 10:40:47", account_number: nil, created_at:
     "2016-04-05 10:41:00", updated_at: "2016-04-05 10:41:00", salt: "
    mXQUUNuXbJY=">
irb(main):002:0> v = User.find 3
  User Load (0.2ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 3
      LIMIT 1
=> #<User id: 3, email: "valaki3@mail.bme.hu", name: "Val Ki III",
    encrypted_password: "a4ee53650096b99fe935ee0d294fe56f2247230ce5fbd217f9
    ...", birthdate: "2011-04-05 00:00:00", account_number: "", created_at:
    "2016-04-05 11:25:29", updated_at: "2016-04-05 11:25:29", salt: "
    zcCj6CKYEdE=">
irb(main):003:0> u.friends
  User Load (0.5ms)  SELECT `users`.* FROM `users` INNER JOIN `friends_users
      ` ON `users`.`id` = `friends_users`.`user_id` WHERE `friends_users`.`
      friend_id` = 2
=> #<ActiveRecord::Associations::CollectionProxy []>
irb(main):004:0> u.friends << v
   (0.6ms)  BEGIN
  SQL (1.5ms)  INSERT INTO `friends_users` (`friend_id`, `user_id`) VALUES
      (2, 3)
   (42.3ms)  COMMIT
=> #<ActiveRecord::Associations::CollectionProxy [#<User id: 3, email: "
    valaki3@mail.bme.hu", name: "Val Ki III", encrypted_password: "
    a4ee53650096b99fe935ee0d294fe56f2247230ce5fbd217f9...", birthdate:
    "2011-04-05 00:00:00", account_number: "", created_at: "2016-04-05
    11:25:29", updated_at: "2016-04-05 11:25:29", salt: "zcCj6CKYEdE=">]>
irb(main):005:0> u.friends
=> #<ActiveRecord::Associations::CollectionProxy [#<User id: 3, email: "
    valaki3@mail.bme.hu", name: "Val Ki III", encrypted_password: "
    a4ee53650096b99fe935ee0d294fe56f2247230ce5fbd217f9...", birthdate:
    "2011-04-05 00:00:00", account_number: "", created_at: "2016-04-05
    11:25:29", updated_at: "2016-04-05 11:25:29", salt: "zcCj6CKYEdE=">]>
irb(main):006:0> v.friends
  User Load (2.6ms)  SELECT `users`.* FROM `users` INNER JOIN `friends_users
      ` ON `users`.`id` = `friends_users`.`user_id` WHERE `friends_users`.`
      friend_id` = 3
=> #<ActiveRecord::Associations::CollectionProxy []>
```

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 `posts_controller` `set_post` és 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.

```
admin = User.create name: "Adminstrator", email: 'admin@mail.bme.hu'
for i in 1..25 do Post.create(user: User.find(2), author: User.find(3), text
    : "This is the #{i}th post") end
```

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

```
rake db:seed
```