Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2014. október 29.

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 mérkőzések `Game` 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 |
| username   | varchar(255) | YES  |     | NULL    |                |
| password   | varchar(255) | YES  |     | NULL    |                |
| email      | varchar(255) | YES  |     | NULL    |                |
| created_at | datetime     | YES  |     | NULL    |                |
| updated_at | datetime     | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
9 rows in set (0.00 sec)

mysql> describe games;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int(11)      | NO   | PRI | NULL    | auto_increment |
| player1    | int(11)      | YES  |     | NULL    |                |
| player2    | int(11)      | YES  |     | NULL    |                |
| result     | varchar(255) | YES  |     | NULL    |                |
| created_at | datetime     | YES  |     | NULL    |                |
| updated_at | datetime     | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
6 rows in set (0.01 sec)
```

Ezután módosítsuk az adatbázis sémánkat! Két új modell definiálunk. A játék lépéseit tartalmazó modellt, és a kommentek modelljét. Egy lépést egy felhasználó tesz egy játékban egy adott időpontban, ezt a modellben úgy tudjuk megjeleníteni, hogy a lépés tábla hivatkozik egy felhasználóra, egy játékra, megadja magát a lépést, és a Rails hozzáteszi a megtételének

időpontját. A komment egy polimorfikus modell lesz, mert kommentet rendelhetünk egy lépéshez, egy játékhoz és egy felhasználóhoz is, ez utóbbi a közösségi portálunk üzenőfala. A komment modellben meg kell adnunk a kommentet tevő felhasználót, a komment időpontját, a komment szövegét és a kommentelt objektumot. Mind a lépés, mind a komment egy nevesített több-több reláció a felhasználók és a játékok modellek között.

```
rails generate model Move game:references move:string
   user:references
rails generate model Comment user:references comment:
   text commentable:references
```

A lépések migrációból távolítsuk el az automatikusan generált indexeket, a kommentek migrációben pedig jelezzük a polimorfikus tulajdonságot a `commentable` attribútum esetén.

```
class CreateMoves < ActiveRecord::Migration
  def change
    create_table :moves do |t|
      t.references :game
      t.string :move
      t.references :user

      t.timestamps
    end
  end
end
```

```
class CreateComments < ActiveRecord::Migration
  def change
    create_table :comments do |t|
      t.references :user
      t.text :comment
      t.references :commentable, polymorphic: true

      t.timestamps
    end
  end
end
```

A barát reláció megvalósítására szükségünk lesz egy kapcsolótáblára a felhasználók modell és önmaga között. Kapcsolótáblát migrációval tudha-

tunk létrehozni. Ha a migrációnkat jól megadott néven hozzuk létre, a Rails automatikusan generálja a migráció törzsét is.

```
rails generate migration CreateJoinTableFriendUser user
    friend
```

Nézzük meg a létrehozott migrációt! Ügyeljünk arra, hogy a kapcsolótábla nevében az összekapcsolt táblák nevei ábécé rendben szereplen majd.

```
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
```

Az utolsó módosításunk a portálunk biztonságát növeli 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.

```
rails generate migration AddSaltToUser salt:string
```

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 nincs még adat, a létező rekordokat fel irányú migráció esetén nem kell inicializálnunk.

```
class AddSaltToUser < ActiveRecord::Migration
  def up
    add_column :users, :salt, :string
    rename_column :users, :password, :
       encrypted_password
  end
```

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

Ezután hajtsuk végre a migrációkat!

```
rake db:migrate
```

A következő feladatunk az adatbázisbeli táblák közötti relációk leképezése modell osztályok közötti kapcsolatokra. Vegyük sorba a modell oszályokat, és valósítsuk meg a kapcsolatokat!

Kezdjük a felhasználók modellel! Egy felhasználó sok játékot játszhat is lehet, e kapcsolat megvalósítása céljából már felvettünk két idegen kulcsot a `game` táblába az egész típusú `player1` és `player2` attribútumokkal. Ez egy kettő-több kapcsolatot definiál, amit a modell osztályokban a `has_many`, illetve a `belongs_to` metódusok által generált setterek és getterek segítségével elérhetővé tehetünk. Egy felhasználó sok lépést tehet, ez egy egy-több kapcsolat. Egy felhasználó üzenőfalára sok komment érkezhet. Mivel a beérkezett kommentek egy polimorfikus kapcsolaton keresztül valósul meg, a `Comment` modell attribútumának nevére az `as` hash paraméteren kereszül kell hivatkoznunk. Egy felhasználó a kommentek modellel nem csak a kapott kommenteken keresztül áll kapcsolatban, hanem egy-több kapcsolat van egy kommentelő felhasználó és a tett kommentek között, ami a `Comment` osztályon belüli `user` attribútumon keresztül valósul meg. Mivel ennek a neve is `comments` lehetne, de azt az előző kapcsolatra már elhasználtuk, meg kell adnunk egy tetszőleges nevet a kapcsolat felhasználó felőli oldalán, ami legyen `comments_made`, és mivel ez a `Comment` modellre hivatkozik `class_name` attribútumként meg kell adunk az osztály nevét. Végül a barát viszony jelzésére kapcsolatot kell teremtenünk a felhasználó modell példányai között. E kapcsolat számossága több-több, amit a `has_and_belongs_to_many` metódussal valósítunk meg. A kapcsolat egyik oldát nevezzük `users`-nek, a másikat `friends`-nek, de ez utóbbi típusa is az `User`. A kapcsolótábla neve `friends_users`, benne a felhasználó kulcsa `user_id`, a barátok azonosítói pedig `friend_id`.

```
class User < ActiveRecord::Base
  has_many :moves
  has_many :games
  has_many :comments, as: :commentable
  has_many :comments_made, class_name: 'Comment'
```

```
  has_and_belongs_to_many :friends,
    class_name: 'User',
    foreign_key: 'user_id',
    association_foreign_key: 'friend_id',
    join_table: 'friends_users'
end
```

A játék modellben két egy-több kapcsolat van a felhasználók modell irányába. A Rails a `player1`, illetve a `player2` attribútumnevek alapján nem képesek kitalálni a hivatkozott modell osztály nevét, ezért azt a a `class_name` kulcshoz tartozó opcióban át kell adnunk. A táblában az idegen kulcs neve a `player1`, és mivel a Rails az idegen kulcsokhoz a a `_id` posztfixet próbálja illeszteni, explicite meg kell mondanuk az idegen kulcs nevét a a `foreign_key` kulcshoz tartozó értékként. A játékhoz tartozó kommentek tekintetében járjunk el pontosan úgy, mint ahogy azt a felhasználók modellben tettük. Még egy kapcsolatunk van ezen kívül: egy játék sok lépésből áll, ez egy egy-több kapcsolat.

```
class Game < ActiveRecord::Base
  belongs_to :player1, class_name: 'User', foreign_key:
      'player1'
  belongs_to :player2, class_name: 'User', foreign_key:
      'player2'
  has_many :moves
  has_many :comments, as: :commentable
end
```

A lépések osztály kapcsolatait a Rails kitöltötte nekünk. Ide is felvehetjük a kommentek kapcsolatot.

```
class Move < ActiveRecord::Base
  belongs_to :game
  belongs_to :user
  has_many :comments, as: :commentable
end
```

A kommentek modellből nézve is navigálhatóvá tesszük a felhasználó modellel fennálló, a felhasználó által tett kommenteket megadó egy-több kapcsolatot relációt a `belongs_to` metódussal. A kommentelt objektumok polimorfikus kapcsolat, ezt a modellben is jeleznünk kell. Ez plusz egy attribútumot fog felvenni a táblában, ami a kommentelt objektum osztályának nevét fogja tartalmazni.

```
class Comment < ActiveRecord::Base
  belongs_to :user
  belongs_to :commentable, polymorphic: true
end
```

Nézzük meg, hogy működnek-e a modell osztályok közötti kapcsolatok! Konzolon hozzunk létre két új felhasználót, ellenőrizzük a barát viszonyt. Hozzunk létre egy játékot a két felhasználó között, tegyen az egyik felhasználó egy lépést, a másik felhasználó pedig kommentelje a lépést! Közben nézzük meg a különböző `ActiveRecord` keresési függvényeket: az egy rekord rendezett vagy rendezetlen lekérdezését elvégző `first`, `take`, `last` függvényeket, a `find_by` és `where` szűrőfeltételeinak használatát!

```
rails console
irb(main):001:0> u = User.first
  User Load (0.3ms)  SELECT  `users`.* FROM `users`
    ORDER BY `users`.`id` ASC LIMIT 1
=> #<User id: 1, username: "valaki", encrypted_password
   : "37c29faba0e0bf832d786221a2694c0a43171209", email:
    "valaki@mail.bme.hu", created_at: "2014-10-29
   12:24:03", updated_at: "2014-10-29 12:24:03", salt:
   "5265a7feeb5a41a9691450b2fe8026323af1849c">
irb(main):002:0> u = User.last
  User Load (0.6ms)  SELECT  `users`.* FROM `users`
    ORDER BY `users`.`id` DESC LIMIT 1
=> #<User id: 1, username: "valaki", encrypted_password
   : "37c29faba0e0bf832d786221a2694c0a43171209", email:
    "valaki@mail.bme.hu", created_at: "2014-10-29
   12:24:03", updated_at: "2014-10-29 12:24:03", salt:
   "5265a7feeb5a41a9691450b2fe8026323af1849c">
irb(main):003:0> u = User.take
  User Load (0.6ms)  SELECT  `users`.* FROM `users`
    LIMIT 1
=> #<User id: 1, username: "valaki", encrypted_password
   : "37c29faba0e0bf832d786221a2694c0a43171209", email:
    "valaki@mail.bme.hu", created_at: "2014-10-29
   12:24:03", updated_at: "2014-10-29 12:24:03", salt:
   "5265a7feeb5a41a9691450b2fe8026323af1849c">
irb(main):004:0> u
=> #<User id: 1, username: "valaki", encrypted_password
   : "37c29faba0e0bf832d786221a2694c0a43171209", email:
    "valaki@mail.bme.hu", created_at: "2014-10-29
```

```
    12:24:03", updated_at: "2014-10-29 12:24:03", salt:
    "5265a7feeb5a41a9691450b2fe8026323af1849c">
irb(main):005:0> v = User.create username: 'senki',
    password: 'titok', email: 'senki@mail.bme.hu'
    (0.3ms)  BEGIN
  User Exists (0.6ms)  SELECT  1 AS one FROM `users`
      WHERE `users`.`username` = BINARY 'senki' LIMIT 1
  SQL (0.3ms)  INSERT INTO `users` (`created_at`, `
      email`, `encrypted_password`, `salt`, `updated_at
      `, `username`) VALUES ('2014-10-29 12:28:11', '
      senki@mail.bme.hu', '1
      d57626f2c015fab1d257c8f7768c699925721e7', '1
      f9b92c5eb7cb0de308757264de7754575716395', '
      2014-10-29 12:28:11', 'senki')
    (405.1ms)  COMMIT
=> #<User id: 2, username: "senki", encrypted_password:
    "1d57626f2c015fab1d257c8f7768c699925721e7", email:
    "senki@mail.bme.hu", created_at: "2014-10-29
    12:28:11", updated_at: "2014-10-29 12:28:11", salt:
    "1f9b92c5eb7cb0de308757264de7754575716395">
irb(main):006:0> g = Game.new player1: u, player2: v
=> #<Game id: nil, player1: 1, player2: 2, result: nil,
     created_at: nil, updated_at: nil>
irb(main):007:0> g.save
    (0.3ms)  BEGIN
  SQL (0.4ms)  INSERT INTO `games` (`created_at`, `
      player1`, `player2`, `updated_at`) VALUES ('
      2014-10-29 12:28:59', 1, 2, '2014-10-29 12:28:59')
    (4.5ms)  COMMIT
=> true
irb(main):009:0> g.player1
=> #<User id: 1, username: "valaki", encrypted_password
    : "37c29faba0e0bf832d786221a2694c0a43171209", email:
     "valaki@mail.bme.hu", created_at: "2014-10-29
    12:24:03", updated_at: "2014-10-29 12:24:03", salt:
    "5265a7feeb5a41a9691450b2fe8026323af1849c">
irb(main):010:0> u.friends
  User Load (0.8ms)  SELECT `users`.* FROM `users`
      INNER JOIN `friends_users` ON `users`.`id` = `
      friends_users`.`friend_id` WHERE `friends_users`.`
      user_id` = 1
```

```
=> #<ActiveRecord::Associations::CollectionProxy []>
irb(main):011:0> u.friends << v
   (0.3ms)  BEGIN
  SQL (0.4ms)  INSERT INTO `friends_users` (`friend_id
     `, `user_id`) VALUES (2, 1)
   (391.1ms)  COMMIT
=> #<ActiveRecord::Associations::CollectionProxy [#<
   User id: 2, username: "senki", encrypted_password:
   "1d57626f2c015fab1d257c8f7768c699925721e7", email: "
   senki@mail.bme.hu", created_at: "2014-10-29
   12:28:11", updated_at: "2014-10-29 12:28:11", salt:
   "1f9b92c5eb7cb0de308757264de77545757163 95">]>
irb(main):012:0> u.friends
=> #<ActiveRecord::Associations::CollectionProxy [#<
   User id: 2, username: "senki", encrypted_password:
   "1d57626f2c015fab1d257c8f7768c699925721e7", email: "
   senki@mail.bme.hu", created_at: "2014-10-29
   12:28:11", updated_at: "2014-10-29 12:28:11", salt:
   "1f9b92c5eb7cb0de308757264de77545757163 95">]>
irb(main):013:0> v.friends
  User Load (0.8ms)  SELECT `users`.* FROM `users`
     INNER JOIN `friends_users` ON `users`.`id` = `
     friends_users`.`friend_id` WHERE `friends_users`.`
     user_id` = 2
=> #<ActiveRecord::Associations::CollectionProxy []>
irb(main):014:0> v.friends << u
   (0.3ms)  BEGIN
  SQL (0.3ms)  INSERT INTO `friends_users` (`friend_id
     `, `user_id`) VALUES (1, 2)
   (427.7ms)  COMMIT
=> #<ActiveRecord::Associations::CollectionProxy [#<
   User id: 1, username: "valaki", encrypted_password:
   "37c29faba0e0bf832d786221a2694c0a43171209", email: "
   valaki@mail.bme.hu", created_at: "2014-10-29
   12:24:03", updated_at: "2014-10-29 12:24:03", salt:
   "5265a7feeb5a41a9691450b2fe8026323af1849c">]>
irb(main):015:0> c = Comment.new user:v, comment: "
   Meghulyultel?", commentable: u
=> #<Comment id: nil, user_id: 2, comment: "
   Meghulyultel?", commentable_id: 1, commentable_type:
    "User", created_at: nil, updated_at: nil>
```

```
irb(main):016:0> c.save
   (0.4ms)  BEGIN
  SQL (2.1ms)  INSERT INTO `comments` (`comment`, `
      commentable_id`, `commentable_type`, `created_at`,
       `updated_at`, `user_id`) VALUES ('Meghulyultel?',
       1, 'User', '2014-10-29 12:33:21', '2014-10-29
      12:33:21', 2)
   (4.0ms)  COMMIT
=> true
irb(main):017:0> u.comments
  Comment Load (0.5ms)  SELECT `comments`.* FROM `
      comments`  WHERE `comments`.`commentable_id` = 1
      AND `comments`.`commentable_type` = 'User'
=> #<ActiveRecord::Associations::CollectionProxy [#<
   Comment id: 1, user_id: 2, comment: "Meghulyultel?",
    commentable_id: 1, commentable_type: "User",
   created_at: "2014-10-29 12:33:21", updated_at:
   "2014-10-29 12:33:21">]>
irb(main):018:0> v.comments_made
  Comment Load (0.6ms)  SELECT `comments`.* FROM `
      comments`  WHERE `comments`.`user_id` = 2
=> #<ActiveRecord::Associations::CollectionProxy [#<
   Comment id: 1, user_id: 2, comment: "Meghulyultel?",
    commentable_id: 1, commentable_type: "User",
   created_at: "2014-10-29 12:33:21", updated_at:
   "2014-10-29 12:33:21">]>
irb(main):019:0> m = Move.create game:g, user:u, move:
   'e5'
   (0.2ms)  BEGIN
  SQL (0.3ms)  INSERT INTO `moves` (`created_at`, `
      game_id`, `move`, `updated_at`, `user_id`) VALUES
      ('2014-10-29 12:35:39', 1, 'e5', '2014-10-29
      12:35:39', 1)
   (400.3ms)  COMMIT
=> #<Move id: 1, game_id: 1, move: "e5", user_id: 1,
   created_at: "2014-10-29 12:35:39", updated_at:
   "2014-10-29 12:35:39">
irb(main):020:0> g.moves
  Move Load (0.6ms)  SELECT `moves`.* FROM `moves`
      WHERE `moves`.`game_id` = 1
=> #<ActiveRecord::Associations::CollectionProxy [#<
```

```
    Move id: 1, game_id: 1, move: "e5", user_id: 1,
    created_at: "2014-10-29 12:35:39", updated_at:
    "2014-10-29 12:35:39">]>
irb(main):021:0> m.game
=> #<Game id: 1, player1: 1, player2: 2, result: nil,
    created_at: "2014-10-29 12:28:59", updated_at:
    "2014-10-29 12:28:59">
```

Következő lépésként tegyük rendbe a felhasználói session kezelését, ami a `loginform` kontroller 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.

```
rails console
irb(main):001:0> Digest::SHA1.hexdigest('titok')
=> "46ff53e764c4acf97b54db2020573049d2e3dab3"
irb(main):002:0> Digest::SHA2.hexdigest('titok')
=> "5be2bcf5718118eaeab4fe7ae57543262082a8fce89420a5fc4799d99af2f161"
irb(main):003:0> SecureRandom.hex(8)
=> "576befbceb133a3e"
irb(main):004:0> SecureRandom.hex(16)
=> "a599d8adfc255eb81e6256ae2ccad050"
irb(main):005:0> SecureRandom.base64(8)
=> "FR27gRzi1V4="
```

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::SHA1.hexdigest(pass+salt)
  end

  def encrypt_password
    return if password.blank?
    if new_record?
      self.salt = Digest::SHA1.hexdigest(Time.now.to_s
        + email)
    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.

```
rails generate controller sessions create destroy
```

A `sessions` kontrollerhez nem tartozik nézet, a `loginform`, illetve a `logout` link eseményeit kezeli le. Ellenőrizzük, hogy a `layouts/_loginform.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. A létrejött nézeteket töröljük, nincs szükség önálló nézetre belépéskor, illetve kilépéskor, megpróbálunk az aktuális oldalon maradni.

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.

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

  def self.authenticate(name,pass)
    user = User.where(username: name).first
    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. A felhasználónevet és a jelszót a `params` hash-ből vesszük ki a `loginform`-ban 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 SessionController < ApplicationController
  def create
    @user = User.authenticate(params[:username], params
        [:password])
    if @user
     session[:user] = @user.id
     redirect_to :back
    else
     flash[:notice] = "Invalid user name or password"
     redirect_to :back
    end
  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 `_loginform.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
      @current_user = @user
      session[:user] = @user.id
      flash[:error] = 'Successful registration'
      redirect_to :action => :show
    else
      flash[:error] = 'User name already used'
      render action: :new
    end
  end

  private
```

```
    def user_params
      params.require(:user).permit(:password, :username
          , :email, :password_confirmation)
    end
end
```

A regisztrációkor az elmentendő felhasználót még az elmentés előtt validáljuk, a felhasználónév attribútumnak nemüresnek (`:presence`) és egyedinek kell lennie (`:uniqueness`), valamint hossza 4 és 14 köze kell esnie, és ha ez nem teljesül, akkor visszajelzünk, hogy nem megfelelő felhasználónévről 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` posztfixű 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,
    length: {in: 4..14, too_long: 'Too long username'},
    uniqueness: true
  validates :password,
    confirmation: true, :if => :password_required?

  def password_required?
    encrypted_password.blank? || !password.blank?
  end
end
```

Konzolon ellenőrizhetjük, hogy elmenthető-e létező vagy túl rövid felhasználónévvel 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.

Az adatbázisunkat mindjárt fel is tölthetjük kezdeti adatokkal, hogy fejlesztés közben adatbázisból származó adatokkal tudjunk dolgozni. Ezt a `db/seeds.rb` fájlban elhelyezett Ruby metódushívásokkal tudjuk megtenni. Ide pontosan azon utasításoknak kell szerepelniük, amiket a konzolon kiadtunk. Az adatokatokat ezek megadása után a következő paranccsal tudjuk betölteni az adatbázisba:

```
rake db:seed
```