Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2013. árpilis 2.

Az előző gyakorlaton megkezdett példát folytatjuk a felhasználói sessionök kezelésének megvalósításával, illetve az adatmodell kialakításával. Az előző alkalommak három modellt hoztunk létre, a felhasználók `User` nevű modelljét, a fogadható események `Event` nevű modelljét és a fogadások `Bid` 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    |                |
| email        | varchar(255) | YES  |     | NULL    |                |
| password     | varchar(255) | YES  |     | NULL    |                |
| account      | int(11)      | YES  |     | NULL    |                |
| bank_account | int(11)      | YES  |     | NULL    |                |
| created_at   | datetime     | NO   |     | NULL    |                |
| updated_at   | datetime     | NO   |     | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
8 rows in set (0.00 sec)
mysql> describe events;
+-------------+----------+------+-----+---------+----------------+
| Field       | Type     | Null | Key | Default | Extra          |
+-------------+----------+------+-----+---------+----------------+
| id          | int(11)  | NO   | PRI | NULL    | auto_increment |
| eventtime   | datetime | YES  |     | NULL    |                |
| description | text     | YES  |     | NULL    |                |
| oddstrue    | float    | YES  |     | NULL    |                |
| oddsfalse   | float    | YES  |     | NULL    |                |
| created_at  | datetime | NO   |     | NULL    |                |
| updated_at  | datetime | NO   |     | NULL    |                |
+-------------+----------+------+-----+---------+----------------+
mysql> describe bids;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| id         | int(11)  | NO   | PRI | NULL    | auto_increment |
| bidtime    | datetime | YES  |     | NULL    |                |
| created_at | datetime | NO   |     | NULL    |                |
| updated_at | datetime | NO   |     | NULL    |                |
+------------+----------+------+-----+---------+----------------+
```

```
6 rows in set (0.01 sec)
```

Első 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.

Módosítsuk a `User` modellt, hogy az ne a titkosítatlan, hanem a már titkosított jelszót tárolja el. Ezt egy migráció létrehozásával és alkalmazásával, valamint a modell osztály módosításával tesszük meg.

```
rails generate migration AddSaltToUsers salt:string
```

A migrációban minden egyes felhasználói rekordhoz hozzáadunk egy egyéni a jelszó titkosításához használt kulcsot, a jelszó attribútumot pedig átnevezzük, így az titkosítatlanul nem kerülhet bele az adatbázisba. Mivel az átnevezés migráció nem reverzibilis, az automatikusan generált `change` metódust szétválaszjuk egy `up` és egy `down` metódusra.

```
class AddSaltToUsers < ActiveRecord::Migration
  def up
    add_column :users, :salt, :string
    rename_column :users, :passwd, :encrypted_passwd
  end
  def down
    rename_column :users, :encrypted_passwd, :passwd
    remove_column :users, :salt
  end
end
```

A jelszó attribútumot átneveztük, viszont a felületen továbbra is használjuk, ezért csak a modell osztályra korlátozva elérhetővé újra tesszük, és egyúttal kibővítjük a modell osztály elérhető attribútumait a jelszó példányváltozójának különböző változataival

```
class User < ActiveRecord::Base
  attr_accessible :passwd, :passwd_confirmation, :
    ecrypted_passwd
  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(salt+pass)
  end

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

Hajtsuk végre a migrációt, és ellenőrizzük, hogy valóban titkosítódik-e a jelszó. Láthatjuk közben, hogy a `users` tábla struktúrája módosult.

```
rake db:migrate
rails console
irb(main):001:0> u = User.new
=> #<User id: nil, username: nil, email: nil,
    encrypted_password: nil, account: nil, bank_account:
     nil, created_at: nil, updated_at: nil, salt: nil>
irb(main):002:0> u.username = 'valaki'
=> "valaki"
irb(main):003:0> u.email = 'valaki@mail.bme.hu'
=> "valaki@mail.bme.hu"
irb(main):004:0> u.password = 'titok'
=> "titok"
irb(main):005:0> u.bank_account = '11111111-11111111'
```

```
=> "11111111-11111111"
irb(main):006:0> u.account = 0
=> 0
irb(main):007:0> u
=> #<User id: nil, username: "valaki", email: "
    valaki@mail.bme.hu", encrypted_password: nil,
    account: 0, bank_account: 11111111, created_at: nil,
     updated_at: nil, salt: nil>
irb(main):008:0> u.save
   (0.3ms)  BEGIN
  SQL (15.0ms)  INSERT INTO `users` (`account`, `
      bank_account`, `created_at`, `email`, `
      encrypted_password`, `salt`, `updated_at`, `
      username`) VALUES (0, 11111111, '2013-10-30
      11:36:51', 'valaki@mail.bme.hu', '846
      fad21d841f48cad57e2527434a52e7933e461', '6
      d614e5a81713ef5882fa9c17ec3ae9269f73443',
      '2013-10-30 11:36:51', 'valaki')
   (96.5ms)  COMMIT
=> true
```

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
```

A `session` kontrollerhez nem tartozik nézet, a `loginform`, illetve a `logout` link eseményeit kezeli le. Ellenőrizzül, 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.

```
rails generate controller sessions
```

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 a `find_by_username`[1] metódussal, majd elvégzi a hitelesítést. Bármelyik sikertelensége esetén a visszatérési érték `nil`. A hitelesítés (`authenticated?`

[1]Rails 4-től verb!User.where(username:'user')!

metódus) azt ellenőrzi, hogy a titkosított jelszó attribútum megegyezik-e a jelszó titkosítása által visszaadott értékkel.

```
class User < ActiveRecord::Base
  def self.authenticate(username,password)
    user = find_by_username username
    user && user.authenticated?(password) ? user : nil
  end

  def authenticated?(pass)
    encrypted_password==User.encrypt(pass,salt)
  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
    @current_user = User.authenticate(params[:username
      ], params[:password])
    if @current_user
      session[:user] = @current_user.id
      redirect_to :back
    else
      flash[:notice] = "Invalid␣user␣name␣code␣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.

A regisztrációkor az elmentendő felhasználót még az elmentés előtt validáljuk, a felhasználónév attribútumnak nemüresnek (`:presence`) és egyedinek kell lennie (`:uniqueness`), 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). Ezeket az ellenőrzéseket a modell osztályban helper metódusokkal tesszük meg.

```
class User < ActiveRecord::Base
  validates :username, :uniqueness => true, :presence
    => true,
  validates :passwd, :confirmation => true, :if => :
    password_required?
  def password_required?
    encrypted_password.blank? || !password.blank?
  end
```

```
end
```

Konzolon ellenőrizhetjük, hogy elmenthető-e hibás felhasználónévvel egy 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.

```
irb(main):001:0> u = User.new
=> #<User id: nil, username: nil, email: nil,
    encrypted_password: nil, account: nil, bank_account:
     nil, created_at: nil, updated_at: nil, salt: nil>
irb(main):002:0> u.save
   (0.3ms)  BEGIN
  User Exists (0.7ms)  SELECT 1 AS one FROM `users`
    WHERE `users`.`username` IS NULL LIMIT 1
   (0.5ms)  ROLLBACK
=> false
irb(main):003:0> u.username = 'valaki'
=> "valaki"
irb(main):004:0> u.save
   (0.2ms)  BEGIN
  User Exists (17.7ms)  SELECT 1 AS one FROM `users`
    WHERE `users`.`username` = BINARY 'valaki' LIMIT 1
   (0.2ms)  ROLLBACK
=> false
```

Regisztrációkor a form paraméterei alapján létrehozunk egy új felhasználó objektumot, és megpróbáljuk elmenteni azt az adatbázisba, a sikeres volt, akkor a felhasználót automatikusan bejelentkeztetjük, beállítjuk a `session` értékét, és egy `flash` üzenet kíséretében átirányítjuk a felhasználót a kezdőoldalra. Sikertelen mentés esetén újból a regisztrációs oldal nézetét jelenítjük meg rajta egy hibaüzenettel, ami üzenet kulcsa térjen el a bejelentkezéskor használtától, és legyen `:reg`. Hasonlóan kell módosítanunk az `update` eseménykezelő metódust is.

```
class UsersController < ApplicationController
  def create
    @user = User.new(params[:user])
    if @user.save
      @current_user = @user
      session[:user] = @user.id
      flash[:reg] = 'Successful_registration'
      redirect_to :controller=>bids, :action=>'index'
```

```
    else
      flash[:reg] = 'User name already used'
      redirect_to :back
      #render :action=>'new'
    end
  end
end
```

Ezután módosítsuk az adatbázis sémánkat! Egy felhasználónak több eseményre is fogadhat, és egy eseményre több felhasználó is adhat fel fogadást, ezért a `users` és az `events` táblák között meg kell valósítanunk egy indirekt több-több relációt. Ezt egy explicite létrehozott kapcsolótáblán keresztül valósítjuk meg. Egy felhasználó egy eseményre egy fogadást (`bid`) fog kötni. Egy felhasználónak több fogadása van, és egy eseményre több fogadás is létezhet, ezért a `bids` tábla felé mindkét másik modell felől egy egy-több relációnk van. Módosítsuk a `Bid` modellt kiegészítve azt egy-egy idegen kulccsal a `users`, illetve a `events` táblákra egész típusú `user_id`, illetve `task_id` nevű attribútumokkal.

```
rails generate migration AddUserIdAndEventIdToBid
      invoke  active_record
      create    db/migrate/20131030122015
          _add_user_id_and_event_id_to_bids.rb
rake db:migrate
(in /home/kovacsg/gyakorlat)
```

A migrációban két `add_column` sémamódosítást végzünk fel irányban, az új attribútumok típusa az `users`, illetve az `events` tábla `id` attribútumának típusa kell, hogy legyen. Alternatív módon ugyanerre az eredményre jutunk, ha `user`, illetve `event` néven referencia típusú attribútumot veszünk fel.

```
class AddUserIdAndEventIdToBids < ActiveRecord::
   Migration
  def up
    add_column :bids, :user_id, :integer
    add_column :bids, :event_id, :integer
  end

  def down
    remove_column :bids, :user_id
    remove_column :bids, :event_id
  end
end
```

Ezután hajtsuk végre a migrációt.

```
==  AddUserIdAndEventIdToBids: migrating
    ===================================
-- add_column(:bids, :user_id, :integer)
   -> 0.1321s
-- add_column(:bids, :event_id, :integer)
   -> 0.0085s
==  AddUserIdAndEventIdToBids: migrated (0.1410s)
    ==============================
```

Egy felhasználó több eseményre is adhat fel fogadást, ezeket a fogadásokat a `bids` egy-több kapcsolaton (`has_many`) érjük el a modell osztályból, és azokat az eseményeket, amire a felhasználó fogadást adott fel, ezen a példányváltozók kereszül (`:through`) érjük el `events` néven.

```
class User < ActiveRecord::Base
  has_many :bids
  has_many :event, :through => :bids
end
```

Az események modell ugyanezen gondolatmenet alapján ehhez nagyon hasonlóan néz ki. Az eseményre feladott fogadásokat (`bids`), illetve a fogadást feladó felhasználókat (`users`) láthatjuk.

```
class Event < ActiveRecord::Base
  has_many :bids
  has_many :users, :through => :bids
end
```

A fogadások modellben a megoldás irányából navigálhatóvá tesszük a `belongs_to` metódussal a fogadást feladó felhasználó felé irányuló kapcsolatot (`user`), illetve elérhetővé tesszük azt a eseményt, amire a fogadás született (`task`).

```
class Submission < ActiveRecord::Base
  belongs_to :user
  belongs_to :event
end
```

Nézzük meg, hogy működnek-e a modell osztályok közötti kapcsolatok! Írjuk vissza a `events_controllerben` az `index`, `show` és `edit` metódusok

eredeti változatát, majd hozzunk létre egy fogadást! Asszociáljuk konzolon ezt a feladatot a felhasználónkhoz, és ehhez a feladathoz! Ugyanezeket végezzük el a weboldalakon is! Végül próbáljuk ki a kapcsolatokat!

```
rails console
Loading development environment (Rails 3.2.12)
irb(main):001:0> u = User.find 1
  User Load (0.5ms)  SELECT `users`.* FROM `users`
    WHERE `users`.`id` = 1 LIMIT 1
=> #<User id: 1, username: "valaki", email: "
   valaki@mail.bme.hu", encrypted_password: "846
   fad21d841f48cad57e2527434a52e7933e461", account: 0,
   bank_account: 11111111, created_at: "2013-10-30
   11:36:51", updated_at: "2013-10-30 11:36:51", salt:
   "6d614e5a81713ef5882fa9c17ec3ae9269f73443">
irb(main):002:0> e = Event.new
=> #<Event id: nil, eventtime: nil, description: nil,
   oddstrue: nil, oddsfalse: nil, created_at: nil,
   updated_at: nil>
irb(main):003:0> e.eventtime = Time.now + 2.days
=> 2013-11-01 13:23:43 +0100
irb(main):004:0> e.description = 'Megnovekedett␣
   buszforgalom'
=> "Megnovekedett␣buszforgalom"
irb(main):005:0> e.oddstrue = 1.1
=> 1.1
irb(main):007:0> e.oddsfalse = 9.0
=> 9.0
irb(main):008:0> e
=> #<Event id: nil, eventtime: "2013-11-01 12:23:43",
   description: "Megnovekedett buszforgalom", oddstrue:
    1.1, oddsfalse: 9.0, created_at: nil, updated_at:
   nil>
irb(main):009:0> e.save
   (0.3ms)  BEGIN
  SQL (2.9ms)  INSERT INTO `events` (`created_at`, `
     description`, `eventtime`, `oddsfalse`, `oddstrue
     `, `updated_at`) VALUES ('2013-10-30␣12:24:30', '
     Megnovekedett␣buszforgalom', '2013-11-01␣12:23:43'
     , 9.0, 1.1, '2013-10-30␣12:24:30')
   (384.8ms)  COMMIT
```

```
=> true
irb(main):011:0> b = Bid.new
=> #<Bid id: nil, bidtime: nil, created_at: nil,
    updated_at: nil, user_id: nil, event_id: nil>
irb(main):012:0> b.bidtime = Time.now
=> 2013-10-30 13:24:58 +0100
irb(main):014:0> b.user_id = u.id
=> 1
irb(main):015:0> b.user
  User Load (0.3ms)  SELECT `users`.* FROM `users`
    WHERE `users`.`id` = 1 LIMIT 1
=> #<User id: 1, username: "valaki", email: "
    valaki@mail.bme.hu", encrypted_password: "846
    fad21d841f48cad57e2527434a52e7933e461", account: 0,
    bank_account: 11111111, created_at: "2013-10-30
    11:36:51", updated_at: "2013-10-30 11:36:51", salt:
    "6d614e5a81713ef5882fa9c17ec3ae9269f73443">
irb(main):016:0> b.event_id = e.id
=> 1
irb(main):017:0> b.event
  Event Load (0.9ms)  SELECT `events`.* FROM `events`
    WHERE `events`.`id` = 1 LIMIT 1
=> #<Event id: 1, eventtime: "2013-11-01 12:23:43",
    description: "Megnovekedett buszforgalom", oddstrue:
     1.1, oddsfalse: 9.0, created_at: "2013-10-30
    12:24:30", updated_at: "2013-10-30 12:24:30">
irb(main):018:0> b.user = User.find 2
  User Load (0.2ms)  SELECT `users`.* FROM `users`
    WHERE `users`.`id` = 2 LIMIT 1
=> #<User id: 2, username: "aaaaaa", email: "a@mail.bme
    .hu", encrypted_password: "0
    fe7fddbea516d2d134c9a04b13a99c6e257f046", account:
    nil, bank_account: 22222222, created_at: "2013-10-30
     12:07:14", updated_at: "2013-10-30 12:14:58", salt:
     "3de3f469a4d09cde9e3c430b6ec34414ec927fae">
irb(main):019:0> b
=> #<Bid id: nil, bidtime: "2013-10-30 12:24:58",
    created_at: nil, updated_at: nil, user_id: 2,
    event_id: 1>
irb(main):020:0> b.save
   (0.3ms)  BEGIN
```

```
  SQL (2.2ms)  INSERT INTO `bids` (`bidtime`, `
      created_at`, `event_id`, `updated_at`, `user_id`)
      VALUES ('2013-10-30 12:24:58', '2013-10-30
      12:26:21', 1, '2013-10-30 12:26:21', 2)
    (4.1ms)  COMMIT
=> true
irb(main):021:0> w = User.find 2
  User Load (0.3ms)  SELECT `users`.* FROM `users`
      WHERE `users`.`id` = 2 LIMIT 1
=> #<User id: 2, username: "aaaaaa", email: "a@mail.bme
    .hu", encrypted_password: "0
    fe7fddbea516d2d134c9a04b13a99c6e257f046", account:
    nil, bank_account: 22222222, created_at: "2013-10-30
     12:07:14", updated_at: "2013-10-30 12:14:58", salt:
     "3de3f469a4d09cde9e3c430b6ec34414ec927fae">
irb(main):022:0> w.bids
  Bid Load (0.6ms)  SELECT `bids`.* FROM `bids` WHERE `
      bids`.`user_id` = 2
=> [#<Bid id: 1, bidtime: "2013-10-30 12:24:58",
    created_at: "2013-10-30 12:26:21", updated_at:
    "2013-10-30 12:26:21", user_id: 2, event_id: 1>]
```

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.

```
event = Event.new
event.description = 'Jovo heten lesz ora'
event.oddstrue = 1.01
event.oddsfalse = 100
event.eventtime = Time.now + 1.week
event.save
```

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

```
rake db:seed
```