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 két modellt hoztunk létre, a `User` és a `Task` modellt, 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 |
| neptun     | varchar(255) | YES  |     | NULL    |                |
| passwd     | 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> describe tasks;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int(11)      | NO   | PRI | NULL    | auto_increment |
| number     | int(11)      | YES  |     | NULL    |                |
| deadline   | date         | YES  |     | NULL    |                |
| url        | varchar(255) | 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.

```
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+"--"+neptun)
    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
Loading development environment (Rails 3.2.12)
irb(main):001:0> u = User.new
=> #<User id: nil, neptun: nil, encrypted_passwd: nil,
    email: nil, created_at: nil, updated_at: nil, salt:
    nil>
irb(main):002:0> u.neptun = 'oweyoa'
=> "oweyoa"
irb(main):003:0> u.passwd = 'titok'
=> "titok"
irb(main):004:0> u.email = 'kovacsg@tmit.bme.hu'
=> "kovacsg@tmit.bme.hu"
irb(main):005:0> u.save
   (0.2ms)  BEGIN
  User Exists (0.5ms)  SELECT 1 AS one FROM `users`
     WHERE `users`.`neptun` = BINARY 'oweyoa' LIMIT 1
  SQL (0.6ms)  INSERT INTO `users` (`created_at`, `
     email`, `encrypted_passwd`, `neptun`, `salt`, `
     updated_at`) VALUES ('2013-04-02 11:04:35', '
     kovacsg@tmit.bme.hu', '33
     abe7f95fc8eb870ae51b122485dcd4e557bad2', 'oweyoa',
      '7cf2fec597e4e14cf418cef11439ca3b5489ad48',
     '2013-04-02 11:04:35')
```

```
   (412.1ms)  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 session
```

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` metódussal, 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 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 neptun 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 neptun kód attribútumnak nemüresnek (`:presence`) és egyedinek kell lennie (`:uniqueness`), a formátuma pontosan hat számból vagy kis- és nagybetűből álló string, és ha ez nem teljesül, akkor visszajelzünk, hogy nem Neptun-kódról van szó. A jelszónak és annak ismétlésének meg kell egyeznie (`:confirmation`), ha az elmentett jelszó nem üres (`password_required?` metódussal vizsgálva). A felhasználónévre még egy olyan megkötést teszünk, hogy az legalább négy, legfeljebb húsz karakter hosszú (`validates_length_of`). Ezeket az ellenőrzéseket a modell osztályban helper metódusokkal tesszük meg.

```
class User < ActiveRecord::Base
  validates :neptun, :uniqueness => true, :presence =>
     true,
    :format => {:with => /\A[a-zA-Z0-9]{6}\z/,
    :message => "Not a Neptun code"}
  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.

```
Loading development environment (Rails 3.2.12)
irb(main):001:0> u = User.new
=> #<User id: nil, neptun: nil, encrypted_passwd: nil,
    email: nil, created_at: nil, updated_at: nil, salt:
    nil>
irb(main):002:0> u.save
   (0.2ms)  BEGIN
```

```
  User Exists (339.6ms)  SELECT 1 AS one FROM `users`
    WHERE `users`.`neptun` IS NULL LIMIT 1
   (0.5ms)  ROLLBACK
=> false
irb(main):003:0> u.neptun = 'a'
=> "a"
irb(main):004:0> u.save
   (0.3ms)  BEGIN
  User Exists (334.3ms)  SELECT 1 AS one FROM `users`
    WHERE `users`.`neptun` = BINARY 'a' LIMIT 1
   (0.1ms)  ROLLBACK
=> false
irb(main):005:0> u.valid?
  User Exists (0.7ms)  SELECT 1 AS one FROM `users`
    WHERE `users`.`neptun` = BINARY 'a' LIMIT 1
=> false
irb(main):006:0> u.errors
=> #<ActiveModel::Errors:0x00000003add2e0 @base=#<User
   id: nil, neptun: "a", encrypted_passwd: nil, email:
   nil, created_at: nil, updated_at: nil, salt: nil>,
   @messages={:neptun=>["Not a Neptun code"], :
   passwd_confirmation=>["can't be blank"]}>
```

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.

```
class UsersController < ApplicationController
  def create
    @user = User.new(params[:user])
    if @user.save
      @current_user = @user
      session[:user] = @user.id
      flash[:notice] = 'Successful registration'
      redirect_to :controller=>say, :action=>'hello'
    else
      flash[:notice] = 'Neptun code is invalid or
         already used'
      render :action=>'new'
```

```
    end
  end
end
```

Ezután módosítsuk az adatbázis sémánkat! Egy felhasználónak több házi feladatra is adhat megoldást, és egy házi feladatra több felhasználó is adhat megoldást, ezért a `users` és az `tasks` táblák között meg kell valósítanunk egy egy-több relációt. Ezt egy explicite létrehozott kapcsolótáblán keresztül valósítjuk meg. Egy felhasználó egy feladatra egy megoldást (`submisssion`) fog beadni, pontosabban a beadott megoldásai közül egy érvényes lesz. Egy felhasználónak több megoldása van, és egy feladatra több megoldás is létezhet, ezért a `submissions` tábla felé mindkét másik modell felől egy egy-több relációnk van. Hozzuk először létre a `Submission` modellt, ami egy string típusú fájlnév, egy ugyancsak string típusú MIME-típus attribútumot, egy-egy idegen kulcsot a `users`, illetve a `tasks` táblákra egész típussal `user_id` és `task_id` néven. A modell létrehozása automatikusan létrehozza számunkra az időpecséteket is, így azokat nem kell külön definiálnunk.

```
rails generate model Submission filename:string mime:
   string user_id:integer task_id:integer
      invoke  active_record
      create    db/migrate/20130402112352
         _create_submissions.rb
      create    app/models/submission.rb
      invoke    test_unit
      create      test/unit/submission_test.rb
      create      test/fixtures/submissions.yml
rake db:migrate
(in /home/kovacsg/gyakorlat)
==  CreateSubmissions: migrating
   ================================================
-- create_table(:submissions)
   -> 0.4936s
==  CreateSubmissions: migrated (0.4941s)
   =======================================
```

Egy felhasználó több feladatra is adhat be megoldást, ezeket a megoldásokat a `submissions` egy-több kapcsolaton (`has_many`) érjük el a modell osztályból, és azokat a feladatokat, amire a felhasználó megoldást adott be, ezen a példányváltozók kereszül (`:through`) érjük el `tasks` néven.

```
class User < ActiveRecord::Base
  has_many :submissions
```

```
  has_many :tasks, :through => :submissions
end
```

A feladat modell ugyanezen gondolatmenet alapján ehhez nagyon hasonlóan néz ki. A feladatra elkészült megoldásokat (`submissions`), illetve a megoldásokat beadó felhasználókat (`users`) láthatjuk.

```
class User < ActiveRecord::Base
  has_many :submissions
  has_many :users, :through => :submissions
end
```

A megoldások modellben a megoldás irányából navigálhatóvá tesszük a `belongs_to` metódussal a megoldást beadó felhasználó felé irányuló kapcsolatot (`user`), illetve elérhetővé tesszük azt a feladatot, amire a megoldás született (`task`).

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

Nézzük meg, hogy működnek-e a modell osztályok közötti kapcsolatok! Írjuk vissza a `tasks_controllerben` az `index`, `show` és `edit` metódusok eredeti változatát, majd hozzunk létre egy feladatot az előző gyakorlaton létrehozott oldalon! Asszociáljuk konzolon ezt a feladatot a felhasználónkhoz, és ehhez a feladathoz! Végül próbáljuk ki a kapcsolatokat!

```
irb(main):001:0> s = Submission.new
=> #<Submission id: nil, filename: nil, mime: nil,
    user_id: nil, task_id: nil, created_at: nil,
    updated_at: nil>
irb(main):002:0> s.user = User.find 1
  User Load (0.6ms)  SELECT `users`.* FROM `users`
    WHERE `users`.`id` = 1 LIMIT 1
=> #<User id: 1, neptun: "aaaaaa", encrypted_passwd: "
    c5494e7484l0b95222b3a732783c491c5d64107a", email: "
    a@bme.hu", created_at: "2013-04-02 10:35:17",
    updated_at: "2013-04-02 11:15:32", salt: "
    b5a4e69021cfdea7f084bdffa53b459ad550cda3">
irb(main):003:0> s.task = Task.find 1
  Task Load (0.1ms)  SELECT `tasks`.* FROM `tasks`
    WHERE `tasks`.`id` = 1 LIMIT 1
```

```
=> #<Task id: 1, number: 1, deadline: "2013-03-04", url
    : "http://", created_at: "2013-04-02 11:31:13",
    updated_at: "2013-04-02 11:31:13">
irb(main):004:0> s.save
    (0.3ms)  BEGIN
  SQL (0.6ms)  INSERT INTO `submissions` (`created_at`,
       `filename`, `mime`, `task_id`, `updated_at`, `
      user_id`) VALUES ('2013-04-02 11:33:19', NULL,
      NULL, 1, '2013-04-02 11:33:19', 1)
    (218.9ms)  COMMIT
=> true
irb(main):001:0> u = User.find :first
  User Load (0.1ms)  SELECT `users`.* FROM `users`
      LIMIT 1
=> #<User id: 1, neptun: "aaaaaa", encrypted_passwd: "
    c5494e748410b95222b3a732783c491c5d64107a", email: "
    a@bme.hu", created_at: "2013-04-02 10:35:17",
    updated_at: "2013-04-02 11:15:32", salt: "
    b5a4e69021cfdea7f084bdffa53b459ad550cda3">
irb(main):002:0> t = Task.find :first
  Task Load (0.1ms)  SELECT `tasks`.* FROM `tasks`
      LIMIT 1
=> #<Task id: 1, number: 1, deadline: "2013-03-04", url
    : "http://", created_at: "2013-04-02 11:31:13",
    updated_at: "2013-04-02 11:31:13">
irb(main):003:0> u.tasks
  Task Load (2.5ms)  SELECT `tasks`.* FROM `tasks`
      INNER JOIN `submissions` ON `tasks`.`id` = `
      submissions`.`task_id` WHERE `submissions`.`
      user_id` = 1
=> [#<Task id: 1, number: 1, deadline: "2013-03-04",
    url: "http://", created_at: "2013-04-02 11:31:13",
    updated_at: "2013-04-02 11:31:13">]
irb(main):004:0> t.submissions
  Submission Load (0.3ms)  SELECT `submissions`.* FROM
      `submissions` WHERE `submissions`.`task_id` = 1
=> [#<Submission id: 1, filename: "hazifeladat", mime:
    "application/zip", user_id: 1, task_id: 1,
    created_at: "2013-04-02 11:33:19", updated_at:
    "2013-04-02 11:33:19">]
irb(main):005:0> t.users
```

```
  User Load (2.5ms)  SELECT `users`.* FROM `users`
    INNER JOIN `submissions` ON `users`.`id` = `
    submissions`.`user_id` WHERE `submissions`.`
    task_id` = 1
=> [#<User id: 1, neptun: "aaaaaa", encrypted_passwd: "
   c5494e748410b95222b3a732783c491c5d64107a", email: "
   a@bme.hu", created_at: "2013-04-02 10:35:17",
   updated_at: "2013-04-02 11:15:32", salt: "
   b5a4e69021cfdea7f084bdffa53b459ad550cda3">]
irb(main):006:0> t.users[0].neptun
=> "aaaaaa"
```

Az adatbázisunkat minjárt fel is tölthetjük kezdeti adatokkal, hogy fejlesztés közben adatbázisból származó adatokkal tudjunk dolgozni. Ezt a `db/seeds.rb` fájlban elhelyezett Ruby metódushívásokkal tudjuk megtenni.

```
u = User.new
u.neptun = 'cccccc'
u.passwd = 'c'
u.email = 'c@bme.hu'
u.save
```

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

```
rake db:seed
```