Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2012. február 28.

Az előző gyakorlaton megkezdett példát folytatjuk a megoldások beadásának megvalósításával. Az előző alkalommal megvalósított nézetek mögé megtervezzük a modelleket és a kontrollereket. Az előző alkalommak két modellt hoztunk létre, a `User` és a `Issue` 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 |
| email      | varchar(255) | YES  |     | NULL    |                |
| password   | varchar(255) | YES  |     | NULL    |                |
| username   | varchar(255) | YES  |     | NULL    |                |
| admin      | tinyint(1)   | YES  |     | 0       |                |
| created_at | datetime     | NO   |     | NULL    |                |
| updated_at | datetime     | NO   |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
mysql> describe issues;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| id          | int(11)      | NO   | PRI | NULL    | auto_increment |
| label       | varchar(255) | YES  |     | NULL    |                |
| description | text         | YES  |     | NULL    |                |
| priority    | int(11)      | YES  |     | NULL    |                |
| deadline    | datetime     | YES  |     | NULL    |                |
| status      | int(11)      | YES  |     | NULL    |                |
| created_at  | datetime     | NO   |     | NULL    |                |
| updated_at  | datetime     | NO   |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
```

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, :password, :encrypted_password
  end
  def down
    rename_column :users, :encrypted_password, :password
    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.

```
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(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
Loading development environment (Rails 3.2.1)
irb(main):001:0> u = User.new
=> #<User id: nil, email: nil, encrypted_password: nil,
    username: nil, admin: false, created_at: nil,
   updated_at: nil, salt: nil>
irb(main):002:0> u.username='admin'
=> "admin"
irb(main):003:0> u.password='titok'
=> "titok"
irb(main):004:0> u.admin=true
=> true
irb(main):005:0> u.email
=> nil
irb(main):006:0> u.email='admin@mail.bme.hu'
=> "admin@mail.bme.hu"
irb(main):007:0> u.save
   (0.2ms)  BEGIN
  SQL (0.7ms)  INSERT INTO `users` (`admin`, `
      created_at`, `email`, `encrypted_password`, `salt
      `, `updated_at`, `username`) VALUES (1,
      '2012-03-28 10:39:28', 'admin@mail.bme.hu', '79547
      dee50e3efd5e37f92cf20647b982069aa7f', '
      c8703e766f06e54c75d2748e2ce0970a4a7234c1',
      '2012-03-28 10:39:28', 'admin')
   (453.9ms)  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 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.

A regisztrációkor az elmentendő felhasználónévnek nemüresnek (`validates_presence_of`) és egyedinek kell lennie (`validates_uniqueness_of`), és a jelszónak és annak ismétlésének meg kell egyeznie (`validates_confirmation_of`), ha az elmentett jelszó nem üres (`password_required?`). 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_presence_of :username
  validates_uniqueness_of :username
  validates_length_of :username, :within => 4..20
  validates_confirmation_of :password, :if =>
     password_required?

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

Konzolon ellenőrizhetjük, hogy elmenthető-e hibás felhasználónévvel egy rekord.

```
Loading development environment (Rails 3.2.1)
irb(main):001:0> u = User.new
=> #<User id: nil, email: nil, encrypted_password: nil,
    username: nil, admin: false, created_at: nil,
   updated_at: nil, salt: nil>
irb(main):002:0> u.username = 'a'
=> "a"
irb(main):003:0> u.save
   (0.2ms)  BEGIN
  User Exists (0.9ms)  SELECT 1 FROM `users` WHERE `
     users`.`username` = BINARY 'a' LIMIT 1
   (1.4ms)  ROLLBACK
=> false
irb(main):004:0>
```

Regisztrárciókor 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 oldalt 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=>'issues', :action=>'
          index'
    else
      flash[:notice] = 'User name already used'
      redirect_to :action=>'new'
    end
  end
end
```

Az aktuális felhasználó példányára a `new` és `create` kivételével minden akció esetén szükségünk van, így azt a `before_filter`-rel minden akció lefutása előtt inicializálni tudjuk. A filterhez a privát `find_user` metódust rendeljük, ami beállítja a `@user` példányváltozó értékét a `session` alapján. A profil szerkesztést ezután e példányváltozón a `update_attributes` metódussal tesszük meg. Sikeres módosítás esetén a kezdőoldalra navigáljuk a felhasználót egy üzenettel, míg sikertelenség esetén újból a profil oldalt jelenítjük meg egy hibaüzenet kíséretében.

```
class UsersController < ApplicationController
  before_filter :find_user, :except => [:new, :create]
  def update
    if @user.update_attributes params[:user]
      flash[:notice] = 'Update successful'
      redirect_to :controller=>'issues', :action=>'
          index'
    else
      flash[:notice] = 'Could not update user profile'
```

```
      redirect_to :action => 'edit', :id=>@user.id
    end
  end

  private
  def find_user
    @user = User.find(params[:id])
  end
end
```

Ezután módosítsuk az adatbázis sémánkat! Egy felhasználónak több teendője is lehet, ezért a `users` és az `issues` táblák között meg kell valósítanunk egy egy-több relációt. Hozzunk létre az `issues` táblában egy idegen kulcsot, amely a `users` tábla rekordjaira hivatkozik!

```
rails generate migration AddUserIdToIssues
```

A migráció során egy egész típusú `user_id` azonosítójú attribútumot adunk az `issues` táblához. A references metódussal ugyanezt érhettük volna el. Az attribútum hozzáadását a Rails mindkét irányba képes migrálni, ezért elég a `change` metódust definiálnunk.

```
class AddUserIdToIssues < ActiveRecord::Migration
  def change
    add_column :issues, :user_id, :integer
  end
end
```

A következő lépés az, hogy az adatbázis táblák közötti kapcsolatot a modell osztályok között is megvalósítjuk. A relációt mindkét irányban navigálhatóvá tesszük.

Az egy-több reláció egy oldalán, a `User` modell oszályban a `has_many` metódussal elérhetővé tesszük a felhasználó teendőinek listáját.

```
class User < ActiveRecord::Base
  has_many :issues
end
```

A `belongs_to` metódust az idegen kulcsot tartalmazó tábla modell osztályába helyezzük el, így a `user` getterrel minden teendőből elérhető az a felhasználó objektum, akihez a teendő tartozik.

```
class Issue < ActiveRecord::Base
  belongs_to :user
end
```

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.

```
Issue.create(:label=>'Teendo1', :description=>"El ne
    feledd", :status=>1, :priority=>5, :deadline=>Time.
    now)
```

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

```
rake db:seed
```

Ellenőrizzük konzolon, hogy az adat bekerült-e az adatbázisba, majd az idegen kulccsal kapcsoljuk össze a felhasználónkat és a teendőnket! Ezután egy tetszőleges felhasználó teendőinek enumerációját elérhetjük a `user.issues` getterrel, ahol a `user` egy példánya a `User` modell osztálynak.

```
Loading development environment (Rails 3.2.1)
irb(main):001:0> i = Issue.find 1
  Issue Load (0.5ms)  SELECT `issues`.* FROM `issues`
    WHERE `issues`.`id` = 1 LIMIT 1
=> #<Issue id: 1, label: "Teendo1", description: "El ne
    feledd", priority: 5, deadline: "2012-03-28
    11:30:17", status: 1, created_at: "2012-03-28
    11:30:17", updated_at: "2012-03-28 11:30:17",
    user_id: nil>
irb(main):002:0> i.user_id = 2
=> 2
irb(main):003:0> i.save
   (0.2ms)  BEGIN
   (0.7ms)  UPDATE `issues` SET `user_id` = 2, `
      updated_at` = '2012-03-28 11:31:21' WHERE `issues
      `.`id` = 1
   (433.2ms)  COMMIT
=> true
irb(main):004:0> u = User.find 2
  User Load (0.1ms)  SELECT `users`.* FROM `users`
    WHERE `users`.`id` = 2 LIMIT 1
=> #<User id: 2, email: "valaki@mail.bme.hu",
    encrypted_password: "
    de071853654dadd06f2f10a019fb2b0a585aed77", username:
     "valaki", admin: false, created_at: "2012-03-28
```

```
    11:09:30", updated_at: "2012-03-28 11:09:30", salt:
    "b725699f2c33ca83b1910e303718d541135de103">
irb(main):005:0> u.issues
  Issue Load (0.4ms)  SELECT `issues`.* FROM `issues`
    WHERE
  `issues`.`user_id` = 2
=> [#<Issue id: 1, label: "Teendo1", description: "El
    ne feledd", priority: 5, deadline: "2012-03-28
    11:30:17", status: 1, created_at: "2012-03-28
    11:30:17", updated_at: "2012-03-28 11:31:21",
    user_id: 2>]
```

Hozzuk létre az új modellt a kommentekről és egyet a csatolmányokról! A kommentet egy felhasználó készíti így a felhasználó idegen kulcsának szerepelnie kell a táblában, és a kommentek tábla több-egy relációban áll a teendők táblával, így a teendőt tábla tekintetében is szükségünk van egy idegen kulcsra. Ezen kívül a komment szövegét és létrehozásának időpontját kell tárolnunk.

Egy teendőhöz több csatolmány is tartozhat, így a csatolmány táblába is fel kell vennünk a teendők tábla vonatkozásában egy idegen kulcsot. A csatolmány magát a fájlrendszeren fogjuk tárolni, így egy string típusú attribútum elég annak megnevezésére.

```
rails generate model Comment comment:text issue_id:
   integer user_id:integer
rails generate model Attachment file:string issue_id:
   integer
```

A modell osztályok között a következő kapcsolatokat kell még definiálnunk. A `Comment` modellbe kettő `belongs_to` metódushívást kell elhelyeznünk, egyet az `Issue` modell felé fennálló több-egy reláció végett, egyet a `User` modell felé fennálló egy-egy reláció végett.

```
class Comment < ActiveRecord::Base
  belongs_to :issue
  belongs_to :user
end
```

A `Attachment` modell egy `belongs_to` deklaráció tartalmaz, amely az `Issue` modell osztályra vonatkozik.

```
class Attachment < ActiveRecord::Base
  belongs_to :issue
end
```

A teendők modell a csatolmányok és a kommentek modell felé is a reláció több oldalán áll, ezért a relációk navigálhatóságai végett hozzáadunk egy-egy `has_many` deklarációt.

```
class Issue < ActiveRecord::Base
  belongs_to :user
  has_many :attachments
  has_many :comments
end
```

A felhasználó modell két úton kapcsolódik a kommentek modellhez, egyrészt a felhasználó által tett kommentek útján, másrészt a felhasználó teendőihez más felhasználók által fűzött kommenttekkel. Mindkét reláció a `Comment` modell oszályra hivatkozik, ami névütközéshet vezethet. Ezt az egyik reláció átnevezésével oldjuk fel.

```
class User < ActiveRecord::Base
  has_many :comments_made, :class_name=>"Comment"
  has_many :comments, :through => :issues
end
```