Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2011. október 25.

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 `Quote` 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 |
| 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    |                |
+------------+--------------+------+-----+---------+----------------+
mysql> describe quotes;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int(11)      | NO   | PRI | NULL    | auto_increment |
| quote      | text         | YES  |     | NULL    |                |
| evaluation | tinyint(4)   | YES  |     | NULL    |                |
| votes      | int(11)      | YES  |     | NULL    |                |
| created_at | datetime     | YES  |     | NULL    |                |
| updated_at | datetime     | YES  |     | NULL    |                |
| user_id    | int(11)      | YES  |     | NULL    |                |
| source_id  | int(11)      | YES  |     | 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::SHA2.hexdigest(salt+pass)
  end
```

```
  def encrypt_password
    return if password.blank?
    if new_record?
      self.salt = ActiveSupport::SecureRandom.base64(8)
    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 3.1.0)
irb(main):001:0> u = User.new
=> #<User id: nil, username: nil, encrypted_password:
   nil, email: nil, created_at: nil, updated_at: nil,
   salt: nil>
irb(main):002:0> u.username='Valaki'
=> "Valaki"
irb(main):003:0> u.password='valami'
=> "valami"
irb(main):004:0> u.save
   (0.1ms)  BEGIN
  SQL (0.5ms)  INSERT INTO `users` (`created_at`, `
     email`,
`encrypted_password`, `salt`, `updated_at`, `username`)
    VALUES
('2011-10-25 10:38:40', NULL,
'ef9f54849463ff4a181522f85ba7e0e93ee13e261d422c5
a74617fac713059da', 'tXrcD049kmM=', '2011-10-25
   10:38:40', 'Valaki')
   (1.2ms)  COMMIT
=> true
irb(main):005:0>
```

Ezután rátérhetünk a felhasználói session megvalósítására. Ehhez létrehozzuk a `session` kontrollert, amelynek `create` és `destroy` metódusai léptetik be, illetve ki a felhasználót.

```
rails generate controller session
```

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 `session/create`-re mutat-e, illetve a belépett felhasználó menüjében (`users/menu.html.erb`) a `Logout` link a `session/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. 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
```

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 3.1.0)
irb(main):001:0> u = User.new
=> #<User id: nil, username: nil, encrypted_password:
    nil, email: nil, created_at: nil, updated_at: nil,
    salt: nil>
irb(main):002:0> u.password = 'hello'
=> "hello"
irb(main):003:0> u.save
   (0.2ms)  BEGIN
   (0.2ms)  ROLLBACK
=> false
irb(main):004:0> u.username = 'a'
=> "a"
irb(main):005:0> u.save
   (0.1ms)  BEGIN
   (0.1ms)  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 oldalt jelenítjük meg rajta egy hibaüzenettel.

```
class UsersController < ApplicationController
  def create
    @user = User.new(params[:user])
    if @user.save
```

```
        @current_user = @user
        session[:user] = @current_user.id
        flash[:notice] = 'Successful registration'
        redirect_to :controller=>"say", :action=>"hello"
      else
        flash[:notice] = 'User name already used'
        render :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 => 'say', :action =>'
          hello'
    else
      flash[:notice]='Could not update personal data'
      render :action => 'edit'
    end
  end

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

Ezután módosítsuk az adatbázis sémánkat! Válasszuk le az idézet forrását az idézetről, és tegyük azt külön táblába tekintve egy forrástól több idézet is származhat.

```
rails generate migration
   RemoveSourceAndSubjectAndReleasedAtFromQuotes source
   :string subject:string released_at:datetime
```

Ennek eredménye alább látható, felirányú migráció esetén eltávolítjuk az attribútumokat a táblából, melyeket leirányú migráció esetén visszaadunk. Ilyenkor a táblában lévő rekordok esetén az értékük inicializálatlan marad.

```
class RemoveSourceAndSubjectAndReleasedAtFromQuotes <
   ActiveRecord::Migration
  def up
    remove_column :quotes, :source
    remove_column :quotes, :subject
    remove_column :quotes, :released_at
  end

  def down
    add_column :quotes, :released_at, :datetime
    add_column :quotes, :subject, :string
    add_column :quotes, :source, :string
  end
end
```

Hozzuk létre az új modellt a forrásokról, és hozzunk létre egy modellt az idézetekhez kapcsolódóhoz kapcsolódó kommentekről.

```
rails generate model Comment comment:text
rails generate model Source source:string subject:
   string released_at:datetime
```

A források táblája a forrás nevét stringként, a forrás tárgyát stringként és az idézet elhangzásának időpontját tartalmazza. A forrás nevét 40 karakterben maximáltuk.

```
class CreateSources < ActiveRecord::Migration
  def change
    create_table :sources do |t|
      t.string :source, :limit=>40
      t.string :subject
      t.datetime :released_at
      t.timestamps
    end
  end
```

```
end
```

A kommentek táblája szövegként tartalmaz értékeket a `comment` attribútumban az időpecsétek mellett.

```
class CreateComments < ActiveRecord::Migration
  def change
    create_table :comments do |t|
      t.text :comment
      t.timestamps
    end
  end
end
```

A következő feladatunk a modellek közötti relációk definíciója. Ezt mind az adatbázisban a megfelelő idegen kulcsok létrehozásával, mind a modell osztályokban jelenzünk kell a `has_one`, `has_many`, `belongs_to` és `has_and_belongs_to_many` metódusokkal, amelyek automatikusan létrehozzák a settereket/gettereket.

```
rails generate migration AddUserIdAndSourceIdToQuotes
   user_id:integer source_id:integer
rails generate migration AddUserIdAndQuoteIdToComments
   user_id:integer
quote_id:integer
rake db:migrate
```

A sémát a migrációk így módosítják. A `quotes` táblába felveszünk két egész típusú attribútumot. Az egyik a `users` tábla, a másik a `sources` tábla kulcsát tartalmazza.

```
class AddUserIdAndSourceIdToQuotes < ActiveRecord::
   Migration
  def change
    add_column :quotes, :user_id, :integer
    add_column :quotes, :source_id, :integer
  end
end
```

A `comments` táblába szintén két egész típusú attribútumot veszünk fel. Az egyik a `users` tábla egyedi kulcsait, a másik a `quotes` tábla egyedi kulcsait tartalmazhatja.

```
class AddUserIdAndQuoteIdToComments < ActiveRecord::
   Migration
```

```
  def change
    add_column :comments, :user_id, :integer
    add_column :comments, :quote_id, :integer
  end
end
```

Ezeket a kapcsolatokat a modellek között is felvesszük. Minden egyes idegen kulcshoz egy `belongs_to` metódushívás tartozik. Egy felhasználó sok idézetet oszthat meg `has_many :quotes`, és sok idézethez fűzhet kommentet `has_many :comments`.

```
class User < ActiveRecord::Base
  has_many :quotes
  has_many :comments
end
```

Egy forrásból több idézet is származhat.

```
class Source < ActiveRecord::Base
  has_many :quotes
end
```

Egy idézet egy forrástól származik (`belongs_to :source`), és egy felhasználó osztotta meg (`belongs_to :user`), továbbá sok komment tartozhat hozzá (`has_many :comments`).

```
class Quote < ActiveRecord::Base
  belongs_to :user
  has_many :comments
  belongs_to :source
end
```

Egy komment egy felhasználóhoz és egy idézethez tartozik.

```
class Quote < ActiveRecord::Base
  belongs_to :user
  belongs_to :quote
end
```

Konzolon ellenőrizthetjük, hogy a megfelelő setterek/getterek létre jöttek-e.

```
irb(main):001:0> u = User.find_by_username 'Valaki'
  User Load (0.5ms)  SELECT `users`.* FROM `users`
    WHERE `users`.`username` = 'Valaki' LIMIT 1
```

```
=> #<User id: 2, username: "Valaki", encrypted_password
    : "
    ef9f54849463ff4a181522f85ba7e0e93ee13e261d422c5a746
    ...", email: nil, created_at: "2011-10-25 10:38:40",
     updated_at: "2011-10-25 10:38:40", salt: "
    tXrcD049kmM=">
irb(main):002:0> u.comments
  Comment Load (0.6ms) SELECT `comments`.* FROM `
      comments` WHERE `comments`.`user_id` = 2
=> []
irb(main):003:0> u.quotes
  Quote Load (0.3ms) SELECT `quotes`.* FROM `quotes`
     WHERE `quotes`.`user_id` = 2
=> []
```

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

```
User.create(username:'Akarki', password:'akarmi', email
    :'akarmi@mail.bme.hu')
```

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

```
rake db:seed
```