Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2015. október 27.

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 szavak `Word` 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    |                |
| birthdate  | datetime     | YES  |     | NULL    |                |
| created_at | datetime     | NO   |     | NULL    |                |
| updated_at | datetime     | NO   |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)

mysql> describe words;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| id          | int(11)      | NO   | PRI | NULL    | auto_increment |
| word        | varchar(255) | YES  |     | NULL    |                |
| description | varchar(255) | YES  |     | NULL    |                |
| lang        | varchar(255) | YES  |     | NULL    |                |
| pronounce   | varchar(255) | YES  |     | NULL    |                |
| created_at  | datetime     | NO   |     | NULL    |                |
| updated_at  | datetime     | NO   |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

Először növeljük portálunk biztonságát 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. Ha már módosítjuk a felhasználók modelljék, akkor adjuk hozzá azt az attribútumot is, amiben a felhasználó típusát tároljuk.

```
kovacsg@debian:~/gyakorlat/db$ rails g migration AddSaltToUsers
    salt:string
      invoke  active_record
      create    db/migrate/20151027111933_add_salt_to_users.rb
```

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 már van adat, fel irányú migráció esetén gondoskodunk kell azok új attribútumainak inicializálásáról.

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

  def down
    rename_column :users, :encrypted_password, :password
    remove_column :users, :salt
    remove_column :users, :type
  end
end
```

Ezután hajtsuk végre a migrációkat!

```
kovacsg@debian:~/gyakorlat/db/migrate$ rake db:migrate
(in /home/kovacsg/gyakorlat)
== 20151027111933 AddSaltToUsers: migrating
   ===================================
-- add_column(:users, :salt, :string)
   -> 0.1455s
-- add_column(:users, :type, :integer)
   -> 0.0322s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0241s
== 20151027111933 AddSaltToUsers: migrated (0.2025s)
   ==========================
```

```
mysql> describe users;
```

```
+-----------------+--------------+------+-----+---------+----------------+
| Field           | Type         | Null | Key | Default | Extra          |
+-----------------+--------------+------+-----+---------+----------------+
| id              | int(11)      | NO   | PRI | NULL    | auto_increment |
| username        | varchar(255) | YES  |     | NULL    |                |
| encrypted_password | varchar(255) | YES  |     | NULL    |                |
| email           | varchar(255) | YES  |     | NULL    |                |
| birthdate       | datetime     | YES  |     | NULL    |                |
| created_at      | datetime     | NO   |     | NULL    |                |
| updated_at      | datetime     | NO   |     | NULL    |                |
| salt            | varchar(255) | YES  |     | NULL    |                |
| type            | int(11)      | YES  |     | NULL    |                |
+-----------------+--------------+------+-----+---------+----------------+
9 rows in set (0.00 sec)
```

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

```
kovacsg@debian:~/gyakorlat/app/models$ rails c
Loading development environment (Rails 4.2.4)
irb(main):001:0> SecureRandom.hex(16)
=> "27d7f29cc216bb00daa1f7132fa2b71a"
irb(main):002:0> SecureRandom.base64(16)
=> "loL5fbQM00W5PqfAURbCLg=="
rb(main):003:0> Digest::SHA1.hexdigest('lalala'+'trallala')
=> "88bfc28e7760c54278390d440642f1a89b271efb"
```

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(salt+pass)
  end

  def encrypt_password
    return if password.blank?
    if new_record?
      self.salt = SecureRandom.base64(8)
    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
```

A `sessions` kontrollerhez nem tartozik nézet, a `login`, illetve a `logout` link eseményeit kezeli le. Ellenőrizzük, hogy a `layouts/_menu.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 `routes.rb` konfigurációs fájlhoz adjuk hozzá a `post 'sessions/create'` és a `get 'sessions/ destroy`, ugyanis bejelentkezéskor HTTP POST üzenetben adatokat is küldünk a szerver felé, kijelentkezéskor pedig HTTP GET üzenet elég.

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)
    self.encrypted_password == User.encrypt(pass, salt)
  end

  def self.authenticate(username, password)
    user = User.where(username: username).take
    user && user.authenticated?(password) ? 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 `menu`-ben 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 `_menu.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.create!(user_params)
    if @user
      @current_user = @user
      session[:user] = @current_user.id
      flash[:notice] = "Successful registration"
    else
      flash[:notice] = "User name in use"
      render action: :new
    end
    redirect_to :back
  end
```

```
  def update
    if @user.update(user_params)
      flash[:notice] = 'Successful update'
      redirect_to :back
    else
      flash[:notice] = @user.errors.messages
      redirect_to :back
    end
  end

  private
    def user_params
      params.require(:user).permit([:username, :password, :
          password_confirmation, :birthdate])
    end
end
```

Több felhasználó számára is elérhetővé szeretnénk tenni a portálunkat, ez lehetséges, mert a felhasználók adatait most már az adatbázisból vesszük elő. Ha egy konkrét felhasználó adait szeretnénk megnézni, szerkeszteni vagy módosítani, akkor a HTTP kérés paramétereként át kell adnunk a felhasználó adatbázisbeli azonosítóját is, hogy a megfelelő felhasználó adatai jelenlenek meg, módosuljanak. Az első módosítás a HTTP kérés paraméterezhetővé tétele, amit a `routes.rb` konfiguráció állományban teszünk meg – magyarázat a következő előadáson.

```
  get 'users/new'
  post 'users/create', as: 'create_user'
  get 'users/edit/:id', to: 'users#edit', as: '
      edit_user'
  put 'users/update/:id', to: 'users#update', as: '
      update_user'
  get 'users/show/:id', to: 'users#show', as: '
      show_user'
  delete 'users/destroy/:id', to: 'users#destroy', as:
      'destroy_user'
  get 'users/index'
  get 'users/forgotten'
```

Az `id` értéke, akárcsak az HTTP kérés összes egyéb paramétere bekerül a `params` hash-be, ahonnan kikereshetjük, ha szükségünk van rá. Ha épp nem új felhasználót hozunk létre, akkor ezt meg kell tennünk. Mivel több függvény előtt ugyanazt a keresést kellene elvégeznünk, egy privát callback függvényt definiálunk a `before_filter` segítségével, ami az `edit`, `update`, `show` és `create` akciók előtt fut majd le.

```
class UsersController < ApplicationController
  before_filter :find_user, :only => [:edit, :update, :
    show, :destroy]
  private
    def find_user
      @user = User.find params[:id]
    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,
    {
      :length => {:in => 4..18, :too_long => "Tul␣hosszu"},
      presence: true,
      uniqueness: true
    }
  validates :password, confirmation: true, if: :password_required
    ?

  def password_required?
    encrypted_password.blank? || !password.blank?
    # self.new_record?
  end
end
```

Nézzük meg az ActiveRecord keresési interfészének gyakran használt függvényeit. Fontos megjegyeznünk, hogy a `where` egy tömböt ad vissza, amiből nekünk mindig ki kell vennünk a keresett elemet, akkor is, ha előre tudjuk azt, hogy az eredményhalmaz szingleton.

```
kovacsg@debian:~/gyakorlat/app/models$ rails c
Loading development environment (Rails 4.2.4)
irb(main):001:0> User.take
```

```
  User Load (1.9ms)  SELECT `users`.* FROM `users`
    LIMIT 1
=> #<User id: 1, username: "valaki", encrypted_password
   : "3ce502090fd731fd8f2790f34de7e231b2f9ecac", email:
    "valaki@mail.bme.hu", birthdate: "1995-10-27
   11:36:04", created_at: "2015-10-27 11:35:00",
   updated_at: "2015-10-27 11:36:17", salt: "v8WPTJUgn+
   k=", type: nil>
irb(main):002:0> User.find 1
  User Load (3.3ms)  SELECT `users`.* FROM `users`
    WHERE `users`.`id` = 1 LIMIT 1
=> #<User id: 1, username: "valaki", encrypted_password
   : "3ce502090fd731fd8f2790f34de7e231b2f9ecac", email:
    "valaki@mail.bme.hu", birthdate: "1995-10-27
   11:36:04", created_at: "2015-10-27 11:35:00",
   updated_at: "2015-10-27 11:36:17", salt: "v8WPTJUgn+
   k=", type: nil>
irb(main):003:0> User.first
  User Load (0.6ms)  SELECT `users`.* FROM `users`
    ORDER BY `users`.`id` ASC LIMIT 1
=> #<User id: 1, username: "valaki", encrypted_password
   : "3ce502090fd731fd8f2790f34de7e231b2f9ecac", email:
    "valaki@mail.bme.hu", birthdate: "1995-10-27
   11:36:04", created_at: "2015-10-27 11:35:00",
   updated_at: "2015-10-27 11:36:17", salt: "v8WPTJUgn+
   k=", type: nil>
irb(main):004:0> User.last
  User Load (0.6ms)  SELECT `users`.* FROM `users`
    ORDER BY `users`.`id` DESC LIMIT 1
=> #<User id: 1, username: "valaki", encrypted_password
   : "3ce502090fd731fd8f2790f34de7e231b2f9ecac", email:
    "valaki@mail.bme.hu", birthdate: "1995-10-27
   11:36:04", created_at: "2015-10-27 11:35:00",
   updated_at: "2015-10-27 11:36:17", salt: "v8WPTJUgn+
   k=", type: nil>
irb(main):005:0> User.last(1)
  User Load (0.7ms)  SELECT `users`.* FROM `users`
    ORDER BY `users`.`id` DESC LIMIT 1
=> [#<User id: 1, username: "valaki",
   encrypted_password: "3
   ce502090fd731fd8f2790f34de7e231b2f9ecac", email: "
```

```
    valaki@mail.bme.hu", birthdate: "1995-10-27
    11:36:04", created_at: "2015-10-27 11:35:00",
    updated_at: "2015-10-27 11:36:17", salt: "v8WPTJUgn+
    k=", type: nil>]
irb(main):007:0> User.where username: 'valaki'
  User Load (2.6ms) SELECT `users`.* FROM `users`
    WHERE `users`.`username` = 'valaki'
=> #<ActiveRecord::Relation [#<User id: 1, username: "
    valaki", encrypted_password: "3
    ce502090fd731fd8f2790f34de7e231b2f9ecac", email: "
    valaki@mail.bme.hu", birthdate: "1995-10-27
    11:36:04", created_at: "2015-10-27 11:35:00",
    updated_at: "2015-10-27 11:36:17", salt: "v8WPTJUgn+
    k=", type: nil>]>
irb(main):008:0> User.where(username: 'valaki').take
  User Load (0.6ms) SELECT `users`.* FROM `users`
    WHERE `users`.`username` = 'valaki' LIMIT 1
=> #<User id: 1, username: "valaki", encrypted_password
    :
"3ce502090fd731fd8f2790f34de7e231b2f9ecac", email:
"valaki@mail.bme.hu", birthdate: "1995-10-27 11:36:04",
    created_at:
"2015-10-27 11:35:00", updated_at: "2015-10-27 11:36:17
    ", salt:
"v8WPTJUgn+k=", type: nil>
```

Konzolon és a webfelületen 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, amiket kivezethetünk a nézetekre.

```
kovacsg@debian:~/gyakorlat/app/models$ rails c
irb(main):001:0> User.all
  User Load (0.4ms) SELECT `users`.* FROM `users`
=> #<ActiveRecord::Relation [#<User id: 1, username: "valaki",
    encrypted_password: "3ce502090fd731fd8f2790f34de7e231b2f9ecac
    ", email: "valaki@mail.bme.hu", birthdate: "1995-10-27
    11:36:04", created_at: "2015-10-27 11:35:00", updated_at:
    "2015-10-27 11:36:17", salt: "v8WPTJUgn+k=", type: nil>]>
irb(main):002:0>
irb(main):003:0* u = User.new
```

```
=> #<User id: nil, username: nil, encrypted_password: nil, email:
     nil, birthdate: nil, created_at: nil, updated_at: nil, salt:
     nil, type: nil>
irb(main):004:0> u.username = 'valaki2'
=> "valaki2"
irb(main):005:0> u.email = 'valaki2@mail.bme.hu'
=> "valaki2@mail.bme.hu"
irb(main):006:0> u.password = 'titok'
=> "titok"
irb(main):007:0> u.save
   (0.3ms)  BEGIN
  User Exists (0.8ms)  SELECT  1 AS one FROM `users` WHERE `users
     `.`username` = BINARY 'valaki2' LIMIT 1
  SQL (0.4ms)  INSERT INTO `users` (`username`, `email`, `salt`,
     `encrypted_password`, `created_at`, `updated_at`) VALUES ('
     valaki2', 'valaki2@mail.bme.hu', 'XnrnMGeB3KM=', '0
     dc8ae7275add537d143eae0d711c9832f7aee16', '2015-10-27
     12:12:11', '2015-10-27 12:12:11')
   (119.1ms)  COMMIT
=> true
irb(main):008:0> u
=> #<User id: 2, username: "valaki2", encrypted_password: "0
    dc8ae7275add537d143eae0d711c9832f7aee16", email: "
    valaki2@mail.bme.hu", birthdate: nil, created_at: "2015-10-27
     12:12:11", updated_at: "2015-10-27 12:12:11", salt: "
    XnrnMGeB3KM=", type: nil>
irb(main):011:0> u = User.last
  User Load (0.2ms)  SELECT  `users`.* FROM `users`  ORDER BY `
     users`.`id` DESC LIMIT 1
=> #<User id: 3, username: "valaki2@mail.bme.hu",
    encrypted_password: "37ac6693fdf9b6ab04192a028787251ffd8e01be
    ", email: nil, birthdate: "2015-10-27 12:11:00", created_at:
    "2015-10-27 12:14:27", updated_at: "2015-10-27 12:14:27",
    salt: "Pe/G3mM+DQM=", type: nil>
irb(main):012:0> u.delete
  SQL (4.6ms)  DELETE FROM `users` WHERE `users`.`id` = 3
=> #<User id: 3, username: "valaki2@mail.bme.hu",
    encrypted_password: "37ac6693fdf9b6ab04192a028787251ffd8e01be
    ", email: nil, birthdate: "2015-10-27 12:11:00", created_at:
    "2015-10-27 12:14:27", updated_at: "2015-10-27 12:14:27",
    salt: "Pe/G3mM+DQM=", type: nil>
irb(main):013:0> u = User.new
=> #<User id: nil, username: nil, encrypted_password: nil, email:
     nil, birthdate: nil, created_at: nil, updated_at: nil, salt:
     nil, type: nil>
irb(main):014:0> u.username = 'a'
=> "a"
irb(main):015:0> u.save
   (0.3ms)  BEGIN
```

```
  User Exists (0.5ms)  SELECT  1 AS one FROM `users` WHERE `users
      `.`username` = BINARY 'a' LIMIT 1
   (0.1ms)  ROLLBACK
=> false
irb(main):016:0> u.errors
=> #<ActiveModel::Errors:0x00000002770658 @base=#<User id: nil,
    username: "a", encrypted_password: nil, email: nil, birthdate
    : nil, created_at: nil, updated_at: nil, salt: nil, type: nil
    >, @messages={:username=>["is too short (minimum is 4
    characters)"]}>
irb(main):017:0> u.errors.messages
=> {:username=>["is too short (minimum is 4 characters)"]}
irb(main):018:0> u.username = 'valaki2'
=> "valaki2"
irb(main):019:0> u.save
   (0.2ms)  BEGIN
  User Exists (0.5ms)  SELECT  1 AS one FROM `users` WHERE `users
      `.`username` = BINARY 'valaki2' LIMIT 1
   (0.4ms)  ROLLBACK
=> false
irb(main):020:0> u.errors.messages
=> {:username=>["has already been taken"]}
irb(main):021:0> u.username = ''
=> ""
irb(main):022:0> u.save
   (0.2ms)  BEGIN
  User Exists (0.7ms)  SELECT  1 AS one FROM `users` WHERE `users
      `.`username` = BINARY '' LIMIT 1
   (0.9ms)  ROLLBACK
=> false
irb(main):023:0> u.errors.messages
=> {:username=>["is too short (minimum is 4 characters)", "can't
    be blank"]}
irb(main):024:0> u.username = 'valaki4'
=> "valaki4"
irb(main):025:0> u.password = 'c'
=> "c"
irb(main):026:0> u.password_confirmation = 'd'
=> "d"
irb(main):027:0> u.save
   (0.8ms)  BEGIN
  User Exists (2.2ms)  SELECT  1 AS one FROM `users` WHERE `users
      `.`username` = BINARY 'valaki4' LIMIT 1
   (0.2ms)  ROLLBACK
=> false
irb(main):028:0> u.errors.messages
=> {:password_confirmation=>["doesn't match Password"]}
irb(main):029:0> Word.first
  Word Load (0.4ms)  SELECT  `words`.* FROM `words`  ORDER BY `
```

```
      words`.`id` ASC LIMIT 1
=> nil
irb(main):030:0> Word.all
  Word Load (2.3ms)  SELECT `words`.* FROM `words`
=> #<ActiveRecord::Relation []>
```

Vegyünk fel tömegesen szavakat az adatbázisunkba, hogy a fejlesztés során legyen mit megjelenítenünk:

```
irb(main):035:0> ('a'..'z').each do |x| Word.create(
    word: x, description: x, lang: 'en') end
```

Ezután módosítsuk az adatbázis sémánkat! Egy új modell definiálunk, a tesztet. Egy tesztet egy személy tölt ki és a szavan rendezetlen listájával áll kapcsolatban, amit stringként tárolunk az adatbázisban. Egy véletlen listát a következő módon generálhatunk, és alakíthatunk át JSON formátumú stringgé, ami alkalmas adatbázisban való tárolásra, és onnan való visszatöltésre.

```
irb(main):037:0> Word.all.sample(20).collect do |x| x.id end
  Word Load (2.1ms)  SELECT `words`.* FROM `words`
=> [26, 23, 20, 7, 24, 16, 25, 3, 14, 11, 10, 19, 1, 21, 22, 2,
    17, 9, 13, 5]
irb(main):038:0> Word.all.sample(20).collect do |x| x.id end.sort
  Word Load (2.5ms)  SELECT `words`.* FROM `words`
=> [1, 2, 4, 5, 6, 7, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
     23, 24, 26]
irb(main):039:0> Word.all.sample(20).collect do |x| x.id end
  Word Load (0.6ms)  SELECT `words`.* FROM `words`
=> [20, 17, 14, 4, 22, 6, 19, 15, 2, 24, 18, 9, 3, 10, 5, 23, 8,
    11, 26, 1]
irb(main):040:0> Word.all.sample(20).collect do |x| x.id end.join
    ('.')
  Word Load (1.2ms)  SELECT `words`.* FROM `words`
=> "11.14.13.22.17.25.26.8.4.19.15.6.20.21.3.18.16.10.9.5"
irb(main):041:0> Word.all.sample(20).collect do |x| x.id end.join
    (',')
  Word Load (0.6ms)  SELECT `words`.* FROM `words`
=> "1,26,14,25,12,8,17,18,13,10,15,23,21,4,7,6,5,16,19,11"
irb(main):042:0> Word.all.sample(20).collect do |x| x.id end.join
    (',').split(',')
  Word Load (0.4ms)  SELECT `words`.* FROM `words`
=> ["9", "14", "26", "18", "12", "6", "3", "10", "19", "24", "7",
    "15", "17", "23", "2", "13", "21", "20", "4", "22"]
irb(main):043:0> Word.all.sample(20).collect do |x| x.id end.
    to_json
  Word Load (0.2ms)  SELECT `words`.* FROM `words`
=> "[14,22,1,13,23,18,5,25,19,10,4,20,11,8,3,2,7,24,26,15]"
```

```
irb(main):044:0> w = Word.all.sample(20).collect do |x| x.id end.
    to_json
  Word Load (2.1ms)  SELECT `words`.* FROM `words`
=> "[13,18,24,26,12,17,23,6,25,8,11,22,15,2,16,14,3,4,7,21]"
irb(main):045:0> JSON.parse w
=> [13, 18, 24, 26, 12, 17, 23, 6, 25, 8, 11, 22, 15, 2, 16, 14,
    3, 4, 7, 21]
```

A tesztek modellje hivatkozik a felhasználóra, a kérdéseket, illetve a megoldásokat egy-egy tömböt tartalmazó stringben tárolja.

```
kovacsg@debian:~/gyakorlat/log$ rails g model Test user
    :references questions:string solution:string
      invoke  active_record
      create    db/migrate/20151027123005_create_tests.
          rb
      create    app/models/test.rb
      invoke    test_unit
      create      test/models/test_test.rb
      create      test/fixtures/tests.yml
kovacsg@debian:~/gyakorlat/log$ rake db:migrate
(in /home/kovacsg/gyakorlat)
== 20151027123005 CreateTests: migrating
    ==========================================
-- create_table(:tests)
    -> 0.1216s
== 20151027123005 CreateTests: migrated (0.1221s)
    =================================
```

```
class CreateTests < ActiveRecord::Migration
  def change
    create_table :tests do |t|
      t.references :user, index: true, foreign_key:
          true
      t.string :questions
      t.string :solution

      t.timestamps null: false
    end
  end
end
```

A teszt modell egy példányának létrehozásakor inicializáljuk a teszt során felteendő kérdések listáját a `before_save` helperrel.

```
class Test < ActiveRecord::Base
  validates :user, presence: true

  before_save :create_seq

  def create_seq
    self.questions = Word.all.sample(20).collect do |x|
        x.id end.to_json
    self.solution = [].to_json
  end
end
```

Az előző gyakorlaton adatbázis nélkül inicializáltuk ezeket a nézeteket. Most már ezt helyre tehetjük a kontrollerben. Töröljük ki a múltkor hozzáadott kódrészleteket, és állítsuk vissza a kikommentezett viselkedést.

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 több tesztet is kitölthet, az egyes szavakkal kapcsolatban viszont nem tartunk fenn kapcsolatot.

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

A tesztek modell kapcsolata automatikusan generálódott a `references` típus használatával a migrációban, nem kell módosítanunk. A szavak modellel fennálló több-több kapcsolatot (egy szó több tesztben szerepelhet) úgy valósítottuk meg, hogy a szavak azonosítóit egy stringgé szerializált tömbben tároljuk, így ezzel sincs most dolgunk.

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

A szavak modellben nem veszünk fel most kapcsolatot.

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 kiad-

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

```
rake db:seed
```