Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2015. március 31.

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 feladatok `Task` 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    |                |
| name       | varchar(255) | YES  |     | NULL    |                |
| password   | varchar(255) | YES  |     | NULL    |                |
| email      | varchar(255) | YES  |     | NULL    |                |
| created_at | datetime     | NO   |     | NULL    |                |
| updated_at | datetime     | NO   |     | NULL    |                |
| t          | int(11)      | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+


mysql> describe tasks;
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| id           | int(11)      | NO   | PRI | NULL    | auto_increment |
| number       | int(11)      | YES  |     | NULL    |                |
| deadline     | datetime     | YES  |     | NULL    |                |
| submitted_at | datetime     | YES  |     | NULL    |                |
| status       | varchar(255) | YES  |     | NULL    |                |
| created_at   | datetime     | NO   |     | NULL    |                |
| updated_at   | datetime     | NO   |     | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
6 rows in set (0.02 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.

```
rails generate migration AddSaltToUser salt:string
```

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
    rename_column :users, :password, :encrypted_password
    add_column :users, :t, :integer
  end

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

Ezután hajtsuk végre a migrációkat!

```
rake db:migrate
== 20150331101739 AddSaltToUsers: migrating
   ===================================
-- add_column(:users, :salt, :string)
   -> 0.0716s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0453s
-- add_column(:users, :t, :integer)
   -> 0.0533s
== 20150331101739 AddSaltToUsers: migrated (0.1711s)
   ==========================
```

Következő 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. Nézzük meg, milyen egyéb módunk van hash kulcs előállítására Railsben.

```
rails console
irb(main):001:0> Digest::SHA1.hexdigest('')
=> "da39a3ee5e6b4b0d3255bfef95601890afd80709"
irb(main):002:0> SecureRandom.base64(16)
=> "cMp065gxWk5XfKMfTO8lcA=="
irb(main):003:0> SecureRandom.hex(16)
=> "0ec5e3a52081fc5b54d5e176c36d96a9"
```

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 User.encrypt(password,salt)
    Digest::SHA1.hexdigest(password+salt)
  end

  def encrypt_password
    return if password.blank?
    if new_record?
      self.salt = SecureRandom.hex(16)
    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 create destroy
```

A `sessions` kontrollerhez nem tartozik nézet, a `loginform`, 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 fájlban létrejött `get 'sessions/create'` sorban módosítsuk a `get`-et `post`-ra, ugyanis bejelentkezéskor HTTP POST üzenetben adatokat is küldünk a szerver felé.

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,self.salt)
  end

  def User.authenticate(username, password)
    user = User.find_by username: username
    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.new(user_params)
    if @user.save
      @current_user = @user
      session[:user] = @current_user.id
      flash[:notice] = 'Successful registration'
      redirect_to :back
    else
      flash[:notice] = @user.errors.messages
      redirect_to :back
    end
  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, :name, :email, :password_confirmation
      )
    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/edit/:id', :to => 'users#edit'
```

```
post 'users/update/:id', :to => 'users#update'
get 'users/show/:id', :to => 'users#show'
```

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 csak a `new` és `create` akciók előtt nem fut majd le.

```
class UsersController < ApplicationController
  before_filter :find_user, :except => [:new, :create]
  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,
    {
      :presence => true,
      :uniqueness => true,
      :length => {:in => 4..18},
    }
  validates :password, :confirmation => true,
    :if => :password_required?

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

Konzolon 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.

```
rails c
Loading development environment (Rails 4.2.0)
irb(main):001:0> User.all
  User Load (0.5ms)  SELECT `users`.* FROM `users`
=> #<ActiveRecord::Relation [#<User id: 1, username: "valakicsoda
    ", name: "Valaki Csoda", encrypted_password: "titok", email:
    "valakicsoda@mail.bme.hu", created_at: "2015-03-03 12:31:51",
     updated_at: "2015-03-03 12:31:51", salt: nil, t: nil>, #<
    User id: 2, username: "valaki", name: "Vala Ki",
    encrypted_password: "e4f929d6db10f9dec83ef76789ae5c22f2ab5ea3
    ", email: "valaki@mail.bme.hu", created_at: "2015-03-31
    10:35:05", updated_at: "2015-03-31 10:39:36", salt: "8134835
    c9d24d01981cb181cf5068719", t: nil>]>
irb(main):002:0> u = User.new
=> #<User id: nil, username: nil, name: nil, encrypted_password:
    nil, email: nil, created_at: nil, updated_at: nil, salt: nil,
     t: nil>
irb(main):003:0> u.save
   (0.4ms)  BEGIN
  User Exists (0.5ms)  SELECT  1 AS one FROM `users` WHERE `users
      `.`username` IS NULL LIMIT 1
   (0.1ms)  ROLLBACK
=> false
irb(main):004:0> u.valid?
  User Exists (0.5ms)  SELECT  1 AS one FROM `users` WHERE `users
      `.`username` IS NULL LIMIT 1
=> false
irb(main):005:0> u.errors
=> #<ActiveModel::Errors:0x0000000059709f0 @base=#<User id: nil,
    username: nil, name: nil, encrypted_password: nil, email: nil
    , created_at: nil, updated_at: nil, salt: nil, t: nil>,
    @messages={:username=>["can't be blank", "is too short (
    minimum is 4 characters)"]}>
irb(main):006:0> u.errors.messages
=> {:username=>["can't be blank", "is too short (minimum is 4
    characters)"]}
irb(main):007:0> u.username = 'valaki'
=> "valaki"
irb(main):008:0> u.save
   (0.2ms)  BEGIN
  User Exists (0.5ms)  SELECT  1 AS one FROM `users` WHERE `users
      `.`username` = BINARY 'valaki' LIMIT 1
   (1.6ms)  ROLLBACK
=> false
```

```
irb(main):009:0> u.errors
=> #<ActiveModel::Errors:0x000000059709f0 @base=#<User id: nil,
    username: "valaki", name: nil, encrypted_password: nil, email
    : nil, created_at: nil, updated_at: nil, salt: nil, t: nil>,
    @messages={:username=>["has already been taken"]}>
irb(main):010:0> u.errors.messages
=> {:username=>["has already been taken"]}
```

Ezután módosítsuk az adatbázis sémánkat! Egy új modell definiálunk, és módosítjuk a feladatok modellt. A feladat státusz és feltöltési időpontja attribútumai rossz helyen vannak, töröljük őket, viszont szükségünk lesz a feladat leírására, vagy a feladat leírását elérhetővé tevő URL-re.

```
rails generate migration AddUrlToTasks url:string
```

```
class AddUrlToTasks < ActiveRecord::Migration
  def change
    add_column :tasks, :url, :string
    remove_column :tasks, :submitted_at
    remove_column :tasks, :status
  end
end
```

Módosítsuk a feladatok automatikusan generált nézeteit: a `_form.html.erb`-ben, az `index.html.erb`-ben és a `show.html.erb`-ben távolítsuk el a törölt attribútumok HTML elemeit, és adjuk hozzá azok mintájára az `url` attribútum HTML elemét.

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. Ne feledkezzük meg a `task_params` függvényről, ahol módosítanunk kell a HTTP kérésben átküldhető paraméterek halmazát.

Az új modellünk a feladatokra beadott megoldások lesz. A megoldást egy hallgató típusú felhasználó adja be egy feladatra, tehát mindkét másik modell tekintetében szükségünk lesz egy-egy hivatkozásra. A megoldást fájlként tároljuk, a fájlról a nevét és a MIME-típusát jegyezzük fel. Ha a megoldás határidőn túli, akkor jelezzük, hogy kései.

```
rails generate model Submission user:references task:
    references filename:string mime:string late:boolean
```

A létrejövő migrációt nem módosítjuk.

```
class CreateSubmissions < ActiveRecord::Migration
  def change
```

```
    create_table :submissions do |t|
      t.references :user, index: true
      t.references :task, index: true
      t.string :filename
      t.string :mime
      t.boolean :late

      t.timestamps null: false
    end
    add_foreign_key :submissions, :users
    add_foreign_key :submissions, :tasks
  end
end
```

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 megoldást adhat be, és több feladatra is adhat megoldást, amik a megoldásokon keresztül teszün elérhetővé.

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

A megoldások modell kapcsolatai automatikusan generálódtak a `references` típus használatával a migrációban, nem kell módosítanunk.

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

A feladatok modellünk hasonló a felhasználók modellre, egy feladatra sok megoldás születhet, és egy feladatra sok felhasználó adhat megoldást.

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

Nézzük meg, hogy működnek-e a modell osztályok közötti kapcsolatok! Konzolon adjon be egy felhasználó egy feladatra egy megoldást! A gyakor-

laton a regisztráció validációja utáni állapotból folytattuk a megoldás felvételét. Az `u` felhasználónkat nem tudtuk elmenteni az adatbázisban, mert a felhasználónév foglalt volt. A 13-16. sorokban négy különböző módon előrekeressük ugyanazt az objektumot az adatbázisból. A 17. sorban létrehozunk egy új megoldás objektumot, a 18. sorban beállítjuk a beadó felhasználót a `belong_to` után megadott setterrel, a 19. sorban beállítjuk azt a feladatot, amire a megoldás született szintén a `belong_to` után megadott setterrel. A 20. sorban elmentjük a megoldást az adatbázisba, de mivel a felhasználónk nem volt valid, nem kerhetett az adatbázisba, a megoldásunk így nem használható. A 21. sorban elsődleges kulcs alapján keresünk egy felhasználót az adatbázisból, akit a 22. és 23. sorokban hozzárendelünk a megoldáshoz, a két megoldás ekvivalens. A 24. és 26. sorokban elmentjük a változást az adatbázisba, láthatjuk, hogy adatbázisművelet csak az első után történik. A 27-30. sorokban megnézzük a felhasználók és a feladatok asszociációit, amik a 28. és 30. sorok esetében a megoldás modellen keresztül valósulnak meg.

```
rails console
irb(main):002:0> u = User.new
=> #<User id: nil, username: nil, name: nil, encrypted_password:
    nil, email: nil, created_at: nil, updated_at: nil, salt: nil,
     t: nil>
irb(main):007:0> u.username = 'valaki'
=> "valaki"
irb(main):013:0> t = Task.take
  Task Load (0.5ms)  SELECT `tasks`.* FROM `tasks` LIMIT 1
=> #<Task id: 1, number: 1, deadline: "2015-02-27 11:31:00",
    created_at: "2015-03-31 11:32:11", updated_at: "2015-03-31
    11:33:38", url: "http://">
irb(main):014:0> t = Task.first
  Task Load (2.2ms)  SELECT `tasks`.* FROM `tasks`  ORDER BY `
     tasks`.`id` ASC LIMIT 1
=> #<Task id: 1, number: 1, deadline: "2015-02-27 11:31:00",
    created_at: "2015-03-31 11:32:11", updated_at: "2015-03-31
    11:33:38", url: "http://">
irb(main):015:0> t = Task.last
  Task Load (0.6ms)  SELECT `tasks`.* FROM `tasks`  ORDER BY `
     tasks`.`id` DESC LIMIT 1
=> #<Task id: 1, number: 1, deadline: "2015-02-27 11:31:00",
    created_at: "2015-03-31 11:32:11", updated_at: "2015-03-31
    11:33:38", url: "http://">
irb(main):016:0> t = Task.find 1
  Task Load (0.5ms)  SELECT `tasks`.* FROM `tasks` WHERE `tasks
     `.`id` = 1 LIMIT 1
=> #<Task id: 1, number: 1, deadline: "2015-02-27 11:31:00",
    created_at: "2015-03-31 11:32:11", updated_at: "2015-03-31
    11:33:38", url: "http://">
irb(main):017:0> s = Submission.new
```

```
=> #<Submission id: nil, user_id: nil, task_id: nil, filename:
    nil, mime: nil, late: nil, created_at: nil, updated_at: nil>
irb(main):018:0> s.user = u
=> #<User id: nil, username: "valaki", name: nil,
    encrypted_password: nil, email: nil, created_at: nil,
    updated_at: nil, salt: nil, t: nil>
irb(main):019:0> s.task = t
=> #<Task id: 1, number: 1, deadline: "2015-02-27 11:31:00",
    created_at: "2015-03-31 11:32:11", updated_at: "2015-03-31
    11:33:38", url: "http://">
irb(main):020:0> s.save
   (0.3ms)  BEGIN
  User Exists (2.3ms)  SELECT  1 AS one FROM `users` WHERE `users
      `.`username` = BINARY 'valaki' LIMIT 1
  SQL (0.7ms)  INSERT INTO `submissions` (`task_id`, `created_at
      `, `updated_at`) VALUES (1, '2015-03-31 11:36:26.196254', '
      2015-03-31 11:36:26.196254')
   (188.4ms)  COMMIT
=> true
irb(main):021:0> u = User.find 2
  User Load (0.2ms)  SELECT  `users`.* FROM `users` WHERE `users
      `.`id` = 2 LIMIT 1
=> #<User id: 2, username: "valaki", name: "Vala Ki",
    encrypted_password: "e4f929d6db10f9dec83ef76789ae5c22f2ab5ea3
    ", email: "valaki@mail.bme.hu", created_at: "2015-03-31
    10:35:05", updated_at: "2015-03-31 10:39:36", salt: "8134835
    c9d24d01981cb181cf5068719", t: nil>
irb(main):022:0> s.user = u
=> #<User id: 2, username: "valaki", name: "Vala Ki",
    encrypted_password: "e4f929d6db10f9dec83ef76789ae5c22f2ab5ea3
    ", email: "valaki@mail.bme.hu", created_at: "2015-03-31
    10:35:05", updated_at: "2015-03-31 10:39:36", salt: "8134835
    c9d24d01981cb181cf5068719", t: nil>
irb(main):023:0> s.user_id = u.id
=> 2
irb(main):024:0> s.save
   (0.2ms)  BEGIN
  SQL (0.8ms)  UPDATE `submissions` SET `user_id` = 2, `
      updated_at` = '2015-03-31 11:39:03.960160' WHERE `
      submissions`.`id` = 1
   (106.5ms)  COMMIT
=> true
irb(main):025:0> s.user = u
=> #<User id: 2, username: "valaki", name: "Vala Ki",
    encrypted_password: "e4f929d6db10f9dec83ef76789ae5c22f2ab5ea3
    ", email: "valaki@mail.bme.hu", created_at: "2015-03-31
    10:35:05", updated_at: "2015-03-31 10:39:36", salt: "8134835
    c9d24d01981cb181cf5068719", t: nil>
irb(main):026:0> s.save
```

```
   (0.2ms)  BEGIN
   (0.1ms)  COMMIT
=> true
irb(main):027:0> u.submissions
  Submission Load (0.7ms)  SELECT `submissions`.* FROM `
      submissions` WHERE `submissions`.`user_id` = 2
=> #<ActiveRecord::Associations::CollectionProxy [#<Submission id
    : 1, user_id: 2, task_id: 1, filename: nil, mime: nil, late:
    nil, created_at: "2015-03-31 11:36:26", updated_at:
    "2015-03-31 11:39:03">]>
irb(main):028:0> u.tasks
  Task Load (0.6ms)  SELECT `tasks`.* FROM `tasks` INNER JOIN `
      submissions` ON `tasks`.`id` = `submissions`.`task_id`
      WHERE `submissions`.`user_id` = 2
=> #<ActiveRecord::Associations::CollectionProxy [#<Task id: 1,
    number: 1, deadline: "2015-02-27 11:31:00", created_at:
    "2015-03-31 11:32:11", updated_at: "2015-03-31 11:33:38", url
    : "http://">]>
irb(main):029:0> t.submissions
  Submission Load (0.5ms)  SELECT `submissions`.* FROM `
      submissions` WHERE `submissions`.`task_id` = 1
=> #<ActiveRecord::Associations::CollectionProxy [#<Submission id
    : 1, user_id: 2, task_id: 1, filename: nil, mime: nil, late:
    nil, created_at: "2015-03-31 11:36:26", updated_at:
    "2015-03-31 11:39:03">]>
irb(main):030:0> t.users
  User Load (2.8ms)  SELECT `users`.* FROM `users` INNER JOIN `
      submissions` ON `users`.`id` = `submissions`.`user_id`
      WHERE `submissions`.`task_id` = 1
=> #<ActiveRecord::Associations::CollectionProxy [#<User id: 2,
    username: "valaki", name: "Vala Ki", encrypted_password: "
    e4f929d6db10f9dec83ef76789ae5c22f2ab5ea3", email: "
    valaki@mail.bme.hu", created_at: "2015-03-31 10:35:05",
    updated_at: "2015-03-31 10:39:36", salt: "8134835
    c9d24d01981cb181cf5068719", t: nil>]>
```

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

```
rake db:seed
```