Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2017. március 28.

Az előző gyakorlaton megkezdett példát folytatjuk a felhasználói sessionök kezelésének megvalósításával, illetve az adatmodell kialakításával. Az előző alkalommak három modellt hoztunk létre, a felhasználók `User` nevű modelljét, a receptek `Recipe` nevű modelljét és a receptek összetevőinek `Ingredient` nevű modelljét. A felhasználók modellje a következőképp néz ki az adatbázisban.

```
mysql> desc users;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int(11)      | NO   | PRI | NULL    | auto_increment |
| 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    |                |
+------------+--------------+------+-----+---------+----------------+
6 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.

```
kovacsg@debian:~/gyakorlat/db> rails g migration AddSaltToUsers salt:string
      invoke  active_record
      create    db/migrate/20170328101855_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[5.0]
  #def change
  #  reversible do |dir|
  #    dir.up {..}
  #    dir.down {..}
  #  end
  #end
  def up
    add_column :users, :salt, :string
    rename_column :users, :password, :encrypted_password
  end

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

Vegyünk fel két új attribútumot a receptek modellbe a komplexitást, amely egy 1 és 5 közötti szám, és az elkészítés idejét, amely egy percben kifejezett érték, és feltételezhetően három digiten ábrázolható.

```
kovacsg@debian:~/gyakorlat/db/migrate> rails g migration
    AddComptexityAndCookTimeToRecipes complexity:integer{1} cook_time:
    integer{3}
      invoke  active_record
      create    db/migrate/20170328102214
          _add_comptexity_and_cook_time_to_recipes.rb
```

Az automatikusan generált migrációs szkripten nem kell módosítanunk, a Rails invertálni tudja az attribútum hozzáadása migrációt.

```
class AddComptexityAndCookTimeToRecipes < ActiveRecord::Migration[5.0]
  def change
    add_column :recipes, :complexity, :integer, limit: 1
    add_column :recipes, :cook_time, :integer, limit: 3
  end
end
```

Hajtsuk végre a migrációt érvényre juttatandó az adatmodellünk változásait!

```
kovacsg@debian:~/gyakorlat/db/migrate> rails db:migrate
== 20170328101855 AddSaltToUsers: migrating
   ===================================
-- add_column(:users, :salt, :string)
   -> 0.1672s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0191s
== 20170328101855 AddSaltToUsers: migrated (0.1870s)
   ==========================
```

```
== 20170328102214 AddComptexityAndCookTimeToRecipes: migrating
   ================
-- add_column(:recipes, :complexity, :integer, {:limit=>1})
   -> 0.0399s
-- add_column(:recipes, :cook_time, :integer, {:limit=>3})
   -> 0.0438s
== 20170328102214 AddComptexityAndCookTimeToRecipes: migrated (0.0841s)
   =======
```

A `new_record?` függvény azt állapítja meg egy ActiveRecord objektumról, hogy azt elmentettük-e már az adatbázisba, ezt az `id` attribútum ellenőrzésével végzi el, ami alapértelmezés szerint `nil`. A memóriában létrehozott `ActiveRercord` objektumok az első mentés műveletig új rekordnak számítanak, az `id` attribútumuk csak annak hatására inicializálódik.

```
kovacsg@debian:~/gyakorlat/app/controllers> rails c
Loading development environment (Rails 5.0.1)
irb(main):001:0> User.new
=> #<User id: nil, name: nil, encrypted_password: nil, email: nil,
    created_at: nil, updated_at: nil, salt: nil>
irb(main):002:0> u = User.new
=> #<User id: nil, name: nil, encrypted_password: nil, email: nil,
    created_at: nil, updated_at: nil, salt: nil>
irb(main):003:0> u.new_record?
=> true
```

Nézzük meg, hogy milyen elemi műveletekkel férhetünk egy vagy több adatbázisbeli rekordhoz hozzá Rails-ből. A `all` függvény (4. sor) az modell összes rekordját visszaadja egy tömbben. A `first` (5. sor), `last` (6. sor) és `take` (7. sor) rendre az elsődleges kulcs szerinti első, utolsó, és az alapértelmezett rendezés szerinti első rekordot adják vissza. Ezeknek egész paramétert adva a rendezés után visszaadott rekordok számát adhatjuk meg (8. sor).

```
irb(main):004:0> User.all
  User Load (0.8ms)  SELECT `users`.* FROM `users`
=> #<ActiveRecord::Relation [#<User id: 1, name: "Kovacs Gabor",
    encrypted_password: "titok", email: "kovacsg@tmit.bme.hu", created_at:
    "2017-02-28 12:28:16", updated_at: "2017-02-28 12:28:16", salt: nil>]>
irb(main):005:0> User.first
  User Load (2.5ms)  SELECT  `users`.* FROM `users` ORDER BY `users`.`id`
     ASC LIMIT 1
=> #<User id: 1, name: "Kovacs Gabor", encrypted_password: "titok", email: "
    kovacsg@tmit.bme.hu", created_at: "2017-02-28 12:28:16", updated_at:
    "2017-02-28 12:28:16", salt: nil>
irb(main):006:0> User.last
  User Load (1.2ms)  SELECT  `users`.* FROM `users` ORDER BY `users`.`id`
     DESC LIMIT 1
=> #<User id: 1, name: "Kovacs Gabor", encrypted_password: "titok", email: "
    kovacsg@tmit.bme.hu", created_at: "2017-02-28 12:28:16", updated_at:
    "2017-02-28 12:28:16", salt: nil>
irb(main):007:0> User.take
  User Load (2.3ms)  SELECT  `users`.* FROM `users` LIMIT 1
=> #<User id: 1, name: "Kovacs Gabor", encrypted_password: "titok", email: "
    kovacsg@tmit.bme.hu", created_at: "2017-02-28 12:28:16", updated_at:
    "2017-02-28 12:28:16", salt: nil>
```

```
irb(main):008:0> User.take(2)
  User Load (2.1ms)  SELECT  `users`.* FROM `users` LIMIT 2
=> [#<User id: 1, name: "Kovacs Gabor", encrypted_password: "titok",
email: "kovacsg@tmit.bme.hu", created_at: "2017-02-28 12:28:16",
updated_at: "2017-02-28 12:28:16", salt: nil>]
```

Valamely attribútum értéke alapján való keresést többféleképpen végezhetünk. Ezek a metódusok minden esetben egy Ruby tömböt adnak vissza, amely azonban lehet akár üres is, vagy tartalmazhat egyetlen objektumot is. A Rails régi változataiból maradtak vissza `find_by_` kezdetű kereső metódusok (9-11. sor), amelyek tetszőleges attribútumnévvel, illetve az attribútumok tetszőleges sorrendben megadott és `_and_`-del elválasztott utótaggal bővítendők. A paraméter az utótagként megadott paraméter értéke, melyre illeszkedő rekordokat keresni kívánunk az adatbázisban. Ha több attribútumnevet soroltunk fel, akkor a paraméterek száma is nő. Ennek modernebb változata a `find_by` metódus (12-13. sor), amely egy hash paraméterrel rendelkezik, amelyben a kulcsok az attribútumok nevei, a hozzájuk tartozó értékek pedig a keresett értékek. A Rails 4-es verziójától a `where` keresőmetódus (14-16. sor) a javasolt. Ezzel nem csak paraméter egyezésére kereshetünk, hanem egy string paraméterrel általánosabb feltételt is megadhatunk, például egy attribútum értéke és egy szám összehasonlítását. A keresőmetódusok tetszőleges számú alkalommal egymás után láncolhatók (15-16. sor), a `where` által visszaadott tömbből jellemzően a `take` metódussal vesszük ki a keresés eredményét, ha tudjuk, hogy pontosan egy lesz.

```
irb(main):009:0> User.find_by_email 'kovacsg@tmit.bme.hu'
  User Load (0.6ms)  SELECT  `users`.* FROM `users` WHERE `users`.`email` =
      'kovacsg@tmit.bme.hu' LIMIT 1
=> #<User id: 1, name: "Kovacs Gabor", encrypted_password: "titok", email: "
    kovacsg@tmit.bme.hu", created_at: "2017-02-28 12:28:16", updated_at:
    "2017-02-28 12:28:16", salt: nil>
irb(main):010:0> User.find_by_encrypted_password 'titok'
  User Load (0.6ms)  SELECT  `users`.* FROM `users` WHERE `users`.`
      encrypted_password` = 'titok' LIMIT 1
=> #<User id: 1, name: "Kovacs Gabor", encrypted_password: "titok", email: "
    kovacsg@tmit.bme.hu", created_at: "2017-02-28 12:28:16", updated_at:
    "2017-02-28 12:28:16", salt: nil>
irb(main):011:0> User.find_by_encrypted_password_and_email 'titok', '
    kovacsg@tmit.bme.hu'
  User Load (0.5ms)  SELECT  `users`.* FROM `users` WHERE `users`.`
      encrypted_password` = 'titok' AND `users`.`email` = 'kovacsg@tmit.bme.
      hu' LIMIT 1
=> #<User id: 1, name: "Kovacs Gabor", encrypted_password: "titok", email: "
    kovacsg@tmit.bme.hu", created_at: "2017-02-28 12:28:16", updated_at:
    "2017-02-28 12:28:16", salt: nil>
irb(main):012:0> User.find_by encrypted_password: 'titok'
  User Load (0.6ms)  SELECT  `users`.* FROM `users` WHERE `users`.`
      encrypted_password` = 'titok' LIMIT 1
=> #<User id: 1, name: "Kovacs Gabor", encrypted_password: "titok", email: "
    kovacsg@tmit.bme.hu", created_at: "2017-02-28 12:28:16", updated_at:
    "2017-02-28 12:28:16", salt: nil>
irb(main):013:0> User.find_by email: 'kovacsg@tmit.bme.hu'
  User Load (0.2ms)  SELECT  `users`.* FROM `users` WHERE `users`.`email` =
```

```
      'kovacsg@tmit.bme.hu' LIMIT 1
=> #<User id: 1, name: "Kovacs Gabor", encrypted_password: "titok", email: "
    kovacsg@tmit.bme.hu", created_at: "2017-02-28 12:28:16", updated_at:
    "2017-02-28 12:28:16", salt: nil>
irb(main):014:0> User.where(encrypted_password: 'titok')
                          User Load (2.5ms)  SELECT `users`.* FROM `users`
    WHERE `users`.`encrypted_password` = 'titok'
=> #<ActiveRecord::Relation [#<User id: 1, name: "Kovacs Gabor",
    encrypted_password: "titok", email: "kovacsg@tmit.bme.hu", created_at:
    "2017-02-28 12:28:16", updated_at: "2017-02-28 12:28:16", salt: nil>]>
irb(main):015:0> User.where(encrypted_password: 'titok').take
  User Load (0.2ms)  SELECT  `users`.* FROM `users` WHERE `users`.`
      encrypted_password` = 'titok' LIMIT 1
=> #<User id: 1, name: "Kovacs Gabor", encrypted_password: "titok", email: "
    kovacsg@tmit.bme.hu", created_at: "2017-02-28 12:28:16", updated_at:
    "2017-02-28 12:28:16", salt: nil>
irb(main):016:0> User.where(encrypted_password: 't').take
  User Load (0.5ms)  SELECT  `users`.* FROM `users` WHERE `users`.`
      encrypted_password` = 't' LIMIT 1
=> nil
```

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 először meg, milyen egyéb módunk van hash kulcsként használható hexadecimális vagy Base64 string előállítására Railsben (3-6. sor).

```
kovacsg@debian:~/gyakorlat/app/models> rails c
Loading development environment (Rails 5.0.1)
irb(main):001:0> Digest::SHA1.hexdigest('alalal')
=> "9adb36176291b9d19ff67f919a2f81563f159a38"
irb(main):002:0> SecureRandom.hex(16)
=> "b71fea165e0890bdfa8d8aa297bceb1c"
irb(main):004:0> SecureRandom.base64(16)
=> "bacu64R9VS/oFo6yCBLn2A=="
irb(main):005:0>
```

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 < ApplicationRecord
  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` példányváltozótót `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 < ApplicationRecord
  before_save :encrypt_password

  def self.encrypt(pass, salt)
    Digest::SHA1.hexdigest(pass+salt)
  end

  def encrypt_password
    return if password.blank?
    if new_record?
      self.salt = SecureRandom.base64(16)
    end
    self.encrypted_password = User.encrypt self.password, self.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.

```
kovacsg@debian:~/gyakorlat/app/views/layouts> rails g controller sessions
Expected string default value for '--helper'; got true (boolean)
Expected string default value for '--assets'; got true (boolean)
      create  app/controllers/sessions_controller.rb
      invoke  erb
      create    app/views/sessions
      invoke  test_unit
      create    test/controllers/sessions_controller_test.rb
      invoke  helper
      create    app/helpers/sessions_helper.rb
      invoke    test_unit
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/sessions.coffee
      invoke    scss
      create      app/assets/stylesheets/sessions.scss
```

A `sessions` kontrollerhez ezek után nem tartozik nézet, a `login`, illetve a `logout` link eseményeit kezeli le. Ellenőrizzük, hogy a menüben, vagyis a `layouts/_menu.html.erb`-ben a form akciója a `/sessions/create`-re mutat-e, illetve a belépett felhasználó menüjében a `Logout` link a `/sessions/destroy`-ra mutat-e. 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, ki-

jelentkezéskor 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 felhasználói email címmel 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 < ApplicationRecord
  def self.authenticate(email, pass)
    #user = find_by_email email
    user = User.where(email: email).take
    user && user.authenticated?(pass) ? user : nil
  end

  def authenticated?(pass)
    self.encrypted_password == User.encrypt(pass, self.salt)
  end
end
```

A kontrollerünk ezek után a következőképp néz ki. A hitelesítés imént megírt metódusának visszatérési értékét a `user` kontroller példányváltozóhoz rendeljük. Az email címet 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 SessionsController < ApplicationController
  def create
    @user = User.authenticate params[:email], params[:password]
    if @user
      session[:user] = @user.id
      #redirect_back
      redirect_to :back
    else
      flash[:notice] = "Invalid user name or password"
      #redirect_back
      redirect_to :back
    end
  end
```

```
  def destroy
    reset_session
    flash[:notice] = "Logged out successfully"
    #redirect_back
    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` kontroller `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
      session[:user] = @user.id
      flash[:notice] = 'Logged in successfully'
      redirect_to  '/say/hello'
    else
      flash[:notice] = 'Email is already in use'
      redirect_back
    end
  end

  def update
    if @user.update(user_params)
      flash[:notice] = "Update successful"
    else
      flash[:notice] = "Could not update user data"
    end
    redirect_to :back
  end
```

```
  private
    def user_params
      params.require(:user).permit(:username, :email, :password, :
          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/new'
  post 'users/create', to: 'users#create'
  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'
  get 'users/forgotten'
  post 'users/send_forgotten', to: 'users#send_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 minden kontroller esetén lefut, mert azt a kontrollerek ősosztályában, az `ApplicationController`-ben definiáljuk. A keresést csak akkor végezzük el, ha van aktív felhasználói session.

```
class ApplicationController < ActionController::Base
  before_action :find_user, only: [:edit, :update, :show]

  protected
    def find_user
      @user = User.find params[:id]
    end
end
```

A regisztrációkor az elmentendő felhasználót még az elmentés előtt validáljuk, az felhasználónév attribútumnak nemüresnek kell lennie (`:presence`), 4 és 18 közötti számú karakterből kell állnia, és egyedinek (`:uniqueness`) kell lennie , és ha ez nem teljesül, akkor visszajelzünk, hogy nem megfelelő felhasználónévről van szó. Az email címet kötelező megadni a regisztráció során, és nem lehet üres. 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 objek-

tumban a `_confirmation` szuffixű 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 :email,
    presence: true
  validates :password, confirmation: true, if: :password_required?

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

Konzolon és a webfelületen ellenőrizhetjük, hogy elmenthető-e üres névvel névvel és létező email címmel egy új 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, amiket kivezethetünk a nézetekre.

```
kovacsg@debian:~/gyakorlat/app/controllers> rails c
Loading development environment (Rails 5.0.1)
irb(main):001:0> u = User.new name: ''
=> #<User id: nil, name: "", encrypted_password: nil, email: nil, created_at
    : nil, updated_at: nil, salt: nil>
irb(main):002:0> u.valid?
  User Exists (0.5ms)  SELECT  1 AS one FROM `users` WHERE `users`.`email`
     IS NULL LIMIT 1
=> false
irb(main):003:0> u.errors
=> #<ActiveModel::Errors:0x007f24d3cf2f70 @base=#<User id: nil, name: "",
    encrypted_password: nil, email: nil, created_at: nil, updated_at: nil,
    salt: nil>, @messages={:name=>["can't be blank"], :email=>["can't be
    blank"]}, @details={:name=>[{:error=>:blank}], :email=>[{:error=>:blank
    }]}>
irb(main):004:0> u.errors.messages
=> {:name=>["can't be blank"], :email=>["can't be blank"]}
irb(main):005:0> u = User.new name: 'valaki', email: 'kovacsg@tmit.bme.hu'
=> #<User id: nil, name: "valaki", encrypted_password: nil, email: "
    kovacsg@tmit.bme.hu", created_at: nil, updated_at: nil, salt: nil>
irb(main):006:0> u.valid?
  User Exists (1.1ms)  SELECT  1 AS one FROM `users` WHERE `users`.`email` =
     BINARY 'kovacsg@tmit.bme.hu' LIMIT 1
=> false
irb(main):007:0> u.errors.messages
=> {:email=>["has already been taken"]}
irb(main):008:0> u.save
   (0.3ms)  BEGIN
  User Exists (0.7ms)  SELECT  1 AS one FROM `users` WHERE `users`.`email` =
     BINARY 'kovacsg@tmit.bme.hu' LIMIT 1
   (0.2ms)  ROLLBACK
=> false
```

A felhasználói regisztáció nézetén (`new.html.erb`) a hibaüzeneteket egyszerűen megjeleníthetjük a következő kódrészlet segítségével:

```
<h1>Registration</h1>
<%= flash[:notice] %>

  <% if @user.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@user.errors.count, "error") %> prohibited this user
          from being saved:</h2>

      <ul>
      <% @user.errors.full_messages.each do |message| %>
        <li><%= message %></li>
      <% end %>
      </ul>
    </div>
  <% end %>
```

Próbáljunk meg ezután felhasználót felvenni a webfelületen hibás adatokkal, és nézzük meg a hibaüzeneteket! Láthatjuk, hogy ezek a műveletek nem hajtódnak végre, és a Rails megjelöli a hibás beviteli mezőket. Ezután a validációs üzenet is megváltozik, amit konzolon ellenőrizhetünk.

A hibaüzenetet testreszabhatjuk a lokalizációs beállításokban:

```
  activerecord:
    errors:
      models:
        user:
          attributes:
            username:
              blank: 'Empty username'
              taken: 'Username already taken'
```

Az előző gyakorlaton létrehoztuk a receptekhez kapcsolódó modelleket. A `references` típusúként deklarált attribútumok esetén az idegen kulcsokhoz automatikusan létrejöttek az ActiveRecord asszociációk. A receptek és a felhasználók közötti kapcsolatot a receptek modellben az egész típusú `user_id` attribútum adja, amely megfelel a Rails alapértelmezett idegen kulcs elnevezési konvenciójának, és így minden módosítás nélkül használhatuk.

A relációink a következők. Egy felhasználó sok receptet tölthet fel a portálra. Egy recept egy felhasználóhoz tartozik, és sok összetevőből áll. Egy összetevő egy recepthez tartozik. A relációink így a Ruby objektumokon is mindkét irányban navigálhatók.

```
class Recipe < ApplicationRecord
  belongs_to :user
  has_many :ingredients
end
class Ingredient < ApplicationRecord
  belongs_to :recipe
end
class User < ApplicationRecord
  has_many :recipes
end
```

Hozzunk létre konzolon néhány receptet! Először nézzük meg, milyen receptek és összetevők érhetők el az adatbázisban (2-3. sor)! Ezután létrehozuk egy recept típusú objektumot, majd inicializáljuk az attribútumait (3-12. sor). A recept tulajdonosa a 2-es azonosítóval rendelkező felhasználó adatbázisból előkeresett példánya lesz (7. sor), láthatjuk, hogy a `user` attribútumhoz `User` típusú objektum tartozik, de használható az egész típusú `user_id` nevű attribútum is (8-10. sor). A receptet a `save` metódussal elmentve az bekerül az adatbázisba, ahonnan mind a recept maga, mind az asszociált felhasználó előkereshető (14-15. sor). Vegyünk fel összetevőket is, amelyek e recepthez tartoznak (16-29. sor), majd kérdezzük meg a recept összetevőit (30. sor).

```
kovacsg@debian:~/gyakorlat/app/controllers> rails c
Loading development environment (Rails 5.0.1)
irb(main):002:0> Ingredient.all
  Ingredient Load (0.4ms)  SELECT `ingredients`.* FROM `ingredients`
=> #<ActiveRecord::Relation []>
irb(main):003:0> Recipe.all
  Recipe Load (0.5ms)  SELECT `recipes`.* FROM `recipes`
=> #<ActiveRecord::Relation []>
irb(main):004:0> r = Recipe.new
=> #<Recipe id: nil, name: nil, description: nil, user_id: nil, created_at:
    nil, updated_at: nil, complexity: nil, cook_time: nil>
irb(main):005:0> r.name = 'Kakaoscsiga'
=> "Kakaoscsiga"
irb(main):007:0> r.description = 'Keverd össze az osszetevoket, es susd meg'
=> "Keverd össze az osszetevoket, es susd meg"
irb(main):008:0> r.user = User.find 2
  User Load (4.3ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 2
    LIMIT 1
=> #<User id: 2, name: "valaki", encrypted_password: "830
    e0c9636b5edd942e5cd644e9ec674a27ba07b", email: "valaki@mail.bme.hu",
    created_at: "2017-03-28 11:18:01", updated_at: "2017-03-28 11:18:01",
    salt: "VBfy4ZKI+16nWTPukcAAnw==">
irb(main):009:0> r.user_id
=> 2
irb(main):010:0> r.user
=> #<User id: 2, name: "valaki", encrypted_password: "830
    e0c9636b5edd942e5cd644e9ec674a27ba07b", email: "valaki@mail.bme.hu",
    created_at: "2017-03-28 11:18:01", updated_at: "2017-03-28 11:18:01",
    salt: "VBfy4ZKI+16nWTPukcAAnw==">
irb(main):011:0> r.complexity = 4
=> 4
irb(main):012:0> r.cook_time = 30
=> 30
irb(main):013:0> r.save
   (0.3ms)  BEGIN
  SQL (0.6ms)  INSERT INTO `recipes` (`name`, `description`, `user_id`, `
      created_at`, `updated_at`, `complexity`, `cook_time`) VALUES ('
      Kakaoscsiga', 'Keverd össze az osszetevoket, es susd meg', 2, '
      2017-03-28 11:32:14', '2017-03-28 11:32:14', 4, 30)
   (181.5ms)  COMMIT
=> true
irb(main):014:0> Recipe.first
  Recipe Load (0.5ms)  SELECT  `recipes`.* FROM `recipes` ORDER BY `recipes
      `.`id` ASC LIMIT 1
=> #<Recipe id: 1, name: "Kakaoscsiga", description: "Keverd össze az
```

```
    osszetevoket, es susd meg", user_id: 2, created_at: "2017-03-28
    11:32:14", updated_at: "2017-03-28 11:32:14", complexity: 4, cook_time:
    30>
irb(main):015:0> Recipe.first.user
  Recipe Load (0.2ms)  SELECT  `recipes`.* FROM `recipes` ORDER BY `recipes
      `.`id` ASC LIMIT 1
  User Load (0.2ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 2
      LIMIT 1
=> #<User id: 2, name: "valaki", encrypted_password: "830
    e0c9636b5edd942e5cd644e9ec674a27ba07b", email: "valaki@mail.bme.hu",
    created_at: "2017-03-28 11:18:01", updated_at: "2017-03-28 11:18:01",
    salt: "VBfy4ZKI+16nWTPukcAAnw==">
irb(main):016:0> i = Ingredient.new
=> #<Ingredient id: nil, element: nil, amount: nil, unit: nil, created_at:
    nil, updated_at: nil, recipe_id: nil>
irb(main):017:0> i.element = 'viz'
=> "viz"
irb(main):018:0> i.amount = 10
=> 10
irb(main):019:0> i.unit = 'dl'
=> "dl"
irb(main):021:0> i.recipe = r
=> #<Recipe id: 1, name: "Kakaoscsiga", description: "Keverd ossze az
    osszetevoket, es susd meg", user_id: 2, created_at: "2017-03-28
    11:32:14", updated_at: "2017-03-28 11:32:14", complexity: 4, cook_time:
    30>
irb(main):022:0> i.save
   (0.2ms)  BEGIN
  SQL (0.5ms)  INSERT INTO `ingredients` (`element`, `amount`, `unit`, `
      created_at`, `updated_at`, `recipe_id`) VALUES ('viz', 10, 'dl', '
      2017-03-28 11:36:22', '2017-03-28 11:36:22', 1)
   (4.1ms)  COMMIT
=> true
irb(main):023:0> r.save
   (0.4ms)  BEGIN
   (0.3ms)  COMMIT
=> true
irb(main):024:0> i = Ingredient.new
=> #<Ingredient id: nil, element: nil, amount: nil, unit: nil, created_at:
    nil, updated_at: nil, recipe_id: nil>
irb(main):025:0> i.element = 'kakao'
=> "kakao"
irb(main):026:0> i.amount = 10
=> 10
irb(main):027:0> i.unit = 'dkg'
=> "dkg"
irb(main):028:0> i.recipe = r
=> #<Recipe id: 1, name: "Kakaoscsiga", description: "Keverd ossze az
    osszetevoket, es susd meg", user_id: 2, created_at: "2017-03-28
    11:32:14", updated_at: "2017-03-28 11:32:14", complexity: 4, cook_time:
    30>
irb(main):029:0> i.save
   (0.2ms)  BEGIN
  SQL (0.3ms)  INSERT INTO `ingredients` (`element`, `amount`, `unit`, `
      created_at`, `updated_at`, `recipe_id`) VALUES ('kakao', 10, 'dkg', '
      2017-03-28 11:37:10', '2017-03-28 11:37:10', 1)
   (105.0ms)  COMMIT
=> true
irb(main):030:0> r.ingredients
  Ingredient Load (0.5ms)  SELECT `ingredients`.* FROM `ingredients` WHERE `
      ingredients`.`recipe_id` = 1
=> #<ActiveRecord::Associations::CollectionProxy [#<Ingredient id: 1,
```

```
    element: "viz", amount: 10, unit: "dl", created_at: "2017-03-28
    11:36:22", updated_at: "2017-03-28 11:36:22", recipe_id: 1>, #<
    Ingredient id: 2, element: "kakao", amount: 10, unit: "dkg", created_at:
     "2017-03-28 11:37:10", updated_at: "2017-03-28 11:37:10", recipe_id:
    1>]>
```

Az adatbázisunkba kezdeti adatokat reprodukálható módon is bevihetünk anélkül, hogy a Rails konzolt használnánk.A `db/seeds.rb` fájlban elhelyezett Ruby metódushívásokkal tudunk adatokat megfelvenni. Itt pontosan azon utasításoknak kell szerepelniük, amiket a konzolon kiadnánk adatok rögzítésekor. Ha az adatokat ebben a fájban adjuk meg, akkor az az adatbázis újrainicializálása után mindig helyreállítható lesz. Vegyünk ezen kívül fel még adatokat, amik a nézetek fejlesztésésben játszanak majd szerepet.

```
user = User.find 2
r = Recipe.create user: user, complexity: 1, name: 'Kenyer', cook_time: 180,
     description:
'Keverd␣ossze␣a␣vizet␣a␣liszttel,␣es␣susd␣3␣oraig'
Ingredient.create recipe: r, element: 'liszt', amount: 60, unit: 'dkg'
Ingredient.create recipe: r, element: 'viz', amount: 40, unit: 'dl'
```

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

```
rake db:seed
```

Eztán a Rails konzolon megnézhetjük, hogy az seed adatok bekerültek-e az adatbázisba.

```
irb(main):035:0> User.find(2).recipes
  User Load (0.6ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 2
      LIMIT 1
  Recipe Load (0.5ms)  SELECT `recipes`.* FROM `recipes` WHERE `recipes`.`
      user_id` = 2
=> #<ActiveRecord::Associations::CollectionProxy [#<Recipe id: 1, name: "
    Kakaoscsiga", description: "Keverd ossze az osszetevoket, es susd meg",
    user_id: 2, created_at: "2017-03-28 11:32:14", updated_at: "2017-03-28
    11:32:14", complexity: 4, cook_time: 30>, #<Recipe id: 2, name: "Kenyer
    ", description: "Keverd ossze a vizet a liszttel, es susd 3 oraig",
    user_id: 2, created_at: "2017-03-28 11:47:10", updated_at: "2017-03-28
    11:47:10", complexity: 1, cook_time: 180>]>
```

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. Ez a `recipes_controller` fájl metódusait érinti.