Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2018. március 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 és az események `Event` nevű modelljét. Ezek modellje a következőképp néz ki az adatbázisban.

```
MariaDB [gyakorlat_development]> show tables;
+-----------------------------------+
| Tables_in_gyakorlat_development |
+-----------------------------------+
| ar_internal_metadata              |
| events                            |
| schema_migrations                 |
| users                             |
+-----------------------------------+
4 rows in set (0.00 sec)

MariaDB [gyakorlat_development]> desc events;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| id          | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| user_id     | bigint(20)   | YES  | MUL | NULL    |                |
| title       | varchar(255) | YES  |     | NULL    |                |
| description | text         | YES  |     | NULL    |                |
| deadline    | datetime     | YES  |     | NULL    |                |
| priority    | tinyint(4)   | YES  |     | NULL    |                |
| created_at  | datetime     | NO   |     | NULL    |                |
| updated_at  | datetime     | NO   |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
8 rows in set (0.01 sec)

MariaDB [gyakorlat_development]> desc users;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| name       | varchar(255) | YES  |     | NULL    |                |
| email      | varchar(255) | YES  |     | NULL    |                |
| password   | varchar(255) | YES  |     | NULL    |                |
| created_at | datetime     | NO   |     | NULL    |                |
```

```
| updated_at | datetime      | NO   |     | NULL    |                |
+------------+---------------+------+-----+---------+----------------+
6 rows in set (0.01 sec)

MariaDB [gyakorlat_development]> select * from users;
+----+--------+-------------------+----------+---------------------+---------------------+
| id | name   | email             | password | created_at          | updated_at          |
+----+--------+-------------------+----------+---------------------+---------------------+
|  1 | Valaki | valaki@mail.bme.hu | titok    | 2018-02-27 12:31:23 | 2018-02-27 12:31:23 |
+----+--------+-------------------+----------+---------------------+---------------------+
1 row in set (0.00 sec)

MariaDB [gyakorlat_development]> Bye
```

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/migrate$ rails g migration AddSaltToUsers salt
    :string
      invoke  active_record
      create    db/migrate/20180327102232_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[5.1]
  def up
    # def change
    # reversible do |dir|
    #   dir.up
    add_column :users, :salt, :string
    rename_column :users, :password, :encrypted_password
    # end
  end

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

Hajtsuk végre a migrációt érvényre juttatandó az adatmodellünk változásait!

```
kovacsg@debian:~/gyakorlat/db/migrate$ rails db:migrate
== 20180327102232 AddSaltToUsers: migrating
   ===================================
-- add_column(:users, :salt, :string)
   -> 0.0250s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0071s
== 20180327102232 AddSaltToUsers: migrated (0.0324s)
   ==========================
```

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
```

```
kovacsg@debian:~/gyakorlat/db/migrate> rails db
MariaDB [gyakorlat_development]> desc users;
+--------------------+--------------+------+-----+---------+----------------+
| Field              | Type         | Null | Key | Default | Extra          |
+--------------------+--------------+------+-----+---------+----------------+
| id                 | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| email              | varchar(255) | YES  |     | NULL    |                |
| encrypted_password | varchar(255) | YES  |     | NULL    |                |
| name               | varchar(255) | YES  |     | NULL    |                |
| created_at         | datetime     | NO   |     | NULL    |                |
| updated_at         | datetime     | NO   |     | NULL    |                |
| salt               | varchar(255) | YES  |     | NULL    |                |
+--------------------+--------------+------+-----+---------+----------------+
8 rows in set (0.00 sec)
```

Nézzük meg, hogy milyen elemi műveletekkel férhetünk egy vagy több adatbázisbeli rekordhoz hozzá Rails-ből. A `first` (7. sor), `last` (9. sor) és

`take` (6. 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 (4. sor), a visszatérési érték `ActiveRecord::Base` objektumról `ActiveRecord::Relation` objektumra változik, ami tömbként működik, és kaszkádosan végrehajthatók rá a keresési műveletek. Az `all` függvény az modell összes rekordját visszaadja egy tömbben.

```
kovacsg@debian:~/gyakorlat/app/models$ rails c
Loading development environment (Rails 5.1.5)
irb(main):001:0> User.all
   (0.2ms)  SET NAMES utf8,  @@SESSION.sql_mode = CONCAT(CONCAT(@@sql_mode,
       ',STRICT_ALL_TABLES'), ',NO_AUTO_VALUE_ON_ZERO'),  @@SESSION.
       sql_auto_is_null = 0, @@SESSION.wait_timeout = 2147483
  User Load (0.4ms)  SELECT  `users`.* FROM `users` LIMIT 11
=> #<ActiveRecord::Relation [#<User id: 1, name: "Valaki", email: "
    valaki@mail.bme.hu", encrypted_password: "titok", created_at:
    "2018-02-27 12:31:23", updated_at: "2018-02-27 12:31:23", salt: nil>]>
irb(main):005:0> User.take
  User Load (0.7ms)  SELECT  `users`.* FROM `users` LIMIT 1
=> #<User id: 1, name: "Valaki", email: "valaki@mail.bme.hu",
    encrypted_password: "titok", created_at: "2018-02-27 12:31:23",
    updated_at: "2018-02-27 12:31:23", salt: nil>
irb(main):006:0> User.take(2)
  User Load (0.7ms)  SELECT  `users`.* FROM `users` LIMIT 2
=> [#<User id: 1, name: "Valaki", email: "valaki@mail.bme.hu",
    encrypted_password: "titok", created_at: "2018-02-27 12:31:23",
    updated_at: "2018-02-27 12:31:23", salt: nil>]
irb(main):007:0> User.first
  User Load (0.8ms)  SELECT  `users`.* FROM `users` ORDER BY `users`.`id`
      ASC LIMIT 1
=> #<User id: 1, name: "Valaki", email: "valaki@mail.bme.hu",
    encrypted_password: "titok", created_at: "2018-02-27 12:31:23",
    updated_at: "2018-02-27 12:31:23", salt: nil>
irb(main):008:0> User.first(2)
  User Load (0.7ms)  SELECT  `users`.* FROM `users` ORDER BY `users`.`id`
      ASC LIMIT 2
=> [#<User id: 1, name: "Valaki", email: "valaki@mail.bme.hu",
    encrypted_password: "titok", created_at: "2018-02-27 12:31:23",
    updated_at: "2018-02-27 12:31:23", salt: nil>]
irb(main):009:0> User.last
  User Load (0.8ms)  SELECT  `users`.* FROM `users` ORDER BY `users`.`id`
      DESC LIMIT 1
=> #<User id: 1, name: "Valaki", email: "valaki@mail.bme.hu",
    encrypted_password: "titok", created_at: "2018-02-27 12:31:23",
    updated_at: "2018-02-27 12:31:23", salt: nil>
```

Elsődleges kulcs alapján, amely jellemzően a weboldalon megjelenő azonosító önmaga, a `find` metódussal kereshetünk egy objektumot. 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 4-es verziójától a `where` keresőmetódus (10-12. 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, a `where` által visszaadott tömbből jellemzően a `take` (vagy `first`) metódussal vesszük ki a keresés eredményét, ha tudjuk, hogy pontosan egy lesz. Egy attribútum értékeinek egy tömbbe való kigyűjtését a 12. sor mutatja.

```
irb(main):003:0> User.find 1
  User Load (0.4ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 1
      LIMIT 1
=> #<User id: 1, name: "Valaki", email: "valaki@mail.bme.hu",
    encrypted_password: "titok", created_at: "2018-02-27 12:31:23",
    updated_at: "2018-02-27 12:31:23", salt: nil>
irb(main):004:0> u = User.find 1
  User Load (0.6ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 1
      LIMIT 1
=> #<User id: 1, name: "Valaki", email: "valaki@mail.bme.hu",
    encrypted_password: "titok", created_at: "2018-02-27 12:31:23",
    updated_at: "2018-02-27 12:31:23", salt: nil>
irb(main):010:0> User.where(name: "Valaki")
  User Load (0.7ms)  SELECT  `users`.* FROM `users` WHERE `users`.`name` = '
      Valaki' LIMIT 11
=> #<ActiveRecord::Relation [#<User id: 1, name: "Valaki", email: "
    valaki@mail.bme.hu", encrypted_password: "titok", created_at:
    "2018-02-27 12:31:23", updated_at: "2018-02-27 12:31:23", salt: nil>]>
irb(main):011:0> User.where(name: "Valaki").first
  User Load (0.4ms)  SELECT  `users`.* FROM `users` WHERE `users`.`name` = '
      Valaki' ORDER BY `users`.`id` ASC LIMIT 1
=> #<User id: 1, name: "Valaki", email: "valaki@mail.bme.hu",
    encrypted_password: "titok", created_at: "2018-02-27 12:31:23",
    updated_at: "2018-02-27 12:31:23", salt: nil>
irb(main):012:0> User.where(name: "Valaki").collect &:email
  User Load (0.7ms)  SELECT `users`.* FROM `users` WHERE `users`.`name` = '
      Valaki'
=> ["valaki@mail.bme.hu"]
irb(main):013:0> v = User.new
=> #<User id: nil, name: nil, email: nil, encrypted_password: nil,
    created_at: nil, updated_at: nil, salt: nil>
irb(main):014:0> v.name = 'Senki'
=> "Senki"
irb(main):015:0> v.email = 'senki@mail.bme.hu
irb(main):016:0' '
=> "senki@mail.bme.hu\n"
irb(main):017:0> v.email = 'senki@mail.bme.hu'
=> "senki@mail.bme.hu"
irb(main):018:0> v.password = 'titok'
=> "titok"
irb(main):019:0> v
=> #<User id: nil, name: "Senki", email: "senki@mail.bme.hu",
    encrypted_password: nil, created_at: nil, updated_at: nil, salt: nil>
irb(main):020:0> v.save
   (0.6ms)  BEGIN
  SQL (0.3ms)  INSERT INTO `users` (`name`, `email`, `encrypted_password`, `
    created_at`, `updated_at`, `salt`) VALUES ('Senki', 'senki@mail.bme.hu',
    '9a6df7fb0ee3654bf03c7b2ba9384708f7633f5d', '2018-03-27 10:40:08', '
    2018-03-27 10:40:08', 'Q9SGMp9+AQA=')
   (7.9ms)  COMMIT
=> true
irb(main):021:0> v
=> #<User id: 2, name: "Senki", email: "senki@mail.bme.hu",
    encrypted_password: "9a6df7fb0ee3654bf03c7b2ba9384708f7633f5d",
```

```
    created_at: "2018-03-27 10:40:08", updated_at: "2018-03-27 10:40:08",
    salt: "Q9SGMp9+AQA=">
```

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 (2-4. sor).

```
kovacsg@debian:~/gyakorlat/db/migrate$ rails c
Loading development environment (Rails 5.1.5)
irb(main):001:0> SecureRandom
=> SecureRandom
irb(main):002:0> SecureRandom.hex
=> "9868d5bd1e09ceaa40221d499488c843"
irb(main):003:0> SecureRandom.hex(8)
=> "5e3f6956ce33150a"
irb(main):004:0> SecureRandom.base64
=> "PKZHrpGQizDRWiEPZpyqCA=="
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` 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 < 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.hex(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/controllers$ rails g controller sessions
    create destroy
      create  app/controllers/sessions_controller.rb
       route  get 'sessions/destroy'
       route  get 'sessions/create'
      invoke  erb
      create    app/views/sessions
      create    app/views/sessions/create.html.erb
      create    app/views/sessions/destroy.html.erb
      invoke  test_unit
      create    test/controllers/sessions_controller_test.rb
      invoke  helper
      create    app/helpers/sessions_helper.rb
      invoke    test_unit
      invoke  assets
```

A létrehozott `create` és `destroy` nézetekre nincs szükségünk, azokat töröljük. 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, 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 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 = User.where(email: email).first
    user && user.authenticated?(pass) ? user : nil
  end

  def authenticated?(pass)
    self.encrypted_password == User.encrypt(self.salt, pass)
  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
      flash[:notice] = "Logged in successfully"
      session[:user] = @user.id
      #redirect_to :back
      redirect_back(fallback_location: welcome_path)
    else
      flash[:notice] = "Invalid email address or password"
      #redirect_to :back
      redirect_back(fallback_location: welcome_path)
  end

  def destroy
    reset_session
    flash[:notice] = "Logged out successfully"
    redirect_to welcome_path
  end
end
```

A flash hashen keresztül értéket adhatunk át a következő HTTP kérésre adott válasz számára. A megjelenítendő üzenet helye legyen az oldal szerkezetében, vagyis az `application.html.erb`-hez a menü felé hozzáadjuk.

```
<%= 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] = "Successful registration"
      redirect_to welcome_path
    else
      flash[:notice] = "Email address in use or password do not match"
      redirect_back(fallback_location: register_path)
    end
  end

  def update
    @user.update(user_params)
    redirect_back
  end

  private
    def user_params
      params.require(:user).permit([:name, :password, :email, :
          password_confirmation])
    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.

```
  match 'sessions/create', to: 'sessions#create', via: :post, as: 'login'
```

```
  match 'sessions/destroy', to: 'sessions#destroy', via: [:delete, :get], as
      : 'logout'

  get 'users/new', to: 'users#new', as: 'register'
  post 'users/create'

  get 'users/edit/:id', to: 'users#edit', as: 'edit_profile'
  put 'users/update/:id', to: 'users#update', as: 'update_profile'

  get 'users/forgotten'
  post 'users/send_forgotten'

  get 'users/show/:id', to: 'users#show', as: 'profile'

  resources :events
  get 'say/hello', as: :welcome
```

A felhasználó `id` attribútumának 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. Bejelentkezett felhasználó esetén ugyanerre a célra felhasználhatjuk a sessionbel tárolt értéket it. 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 UsersController < ApplicationController
  before_action :find_user, only: [:edit, :update, :show] # except: [:new, :
      create, :forgotten, :send_forgotten]

  private
    def find_user
      @user = User.find session[:user]
    end
end
```

A regisztrációkor az elmentendő felhasználót még az elmentés előtt validáljuk, az emailcím attribútumnak nemüresnek kell lennie (`:presence`), és egyedinek (`:uniqueness`) kell lennie , és ha ez nem teljesül, akkor visszajelzünk, hogy nem megfelelő felhasználónévről van szó. A név mezőt 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 objektumban 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 :email, {
      presence: true,
```

```
      uniqueness: true
    }
  validates :name, 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.

```
irb(main):002:0> u = User.new
=> #<User id: nil, name: nil, email: nil, encrypted_password: nil,
    created_at: nil, updated_at: nil, salt: nil>
irb(main):003:0> u.name = 'Valaki'
=> "Valaki"
irb(main):004:0> u.email = 'valaki@mail.bme.hu'
=> "valaki@mail.bme.hu"
irb(main):005:0> u.password = 'a'
=> "a"
irb(main):006:0> u.encrypted_password = 'a'
=> "a"
irb(main):007:0> u.valid?
  User Exists (1.9ms)  SELECT  1 AS one FROM `users` WHERE `users`.`email` =
       BINARY 'valaki@mail.bme.hu' LIMIT 1
=> false
irb(main):008:0> u.errors.messages
=> {:email=>["has already been taken"]}
irb(main):009:0> u.password = 'b'
=> "b"
irb(main):010:0> u.valid?
  User Exists (0.6ms)  SELECT  1 AS one FROM `users` WHERE `users`.`email` =
       BINARY 'valaki@mail.bme.hu' LIMIT 1
=> false
irb(main):011:0> u.errors.messages
=> {:email=>["has already been taken"]}
```

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:
            mail:
              blank: 'Empty email
              taken: 'Email already used'
```

Az előző gyakorlaton kezdeti adatokkal módosítottuk az események kontrollert. Most távolítsuk el azokat, és írjuk vissza az automatikusan generált kódrészleteket.

Egy esemény több módon is kapcsolatban állhat felhasználókkal. Az események modell szempontjából minden eseménynek van egy tulajdonos felhasználója, és van tetszőleges számú résztvevő felhasználója. E kettő között különbséget kell tennünk. A felhasználók modell szempontjából minden felhasználónak lehet tetszőlegesen sok számú saját eseménye, és tetszőlegesen sok számú esemény résztvevője lehet. Tehát a felhasználók és az események között van egy egy-több és egy több-több kapcsolat. A több-több kapcsolat ábrázolásának egyik módja a kapcsolótábla létrehozása.

```
kovacsg@debian:~/gyakorlat/app/models$ rails g migration
    CreateJoinTableEventsUsers event user
      invoke  active_record
      create    db/migrate/20180327113501_create_join_table_events_users.rb
```

A modell osztályok szintjén ez a következőképp jelenik meg. A `User` modellben felveszünk egy `owned_events` azonosítójú `has_many` kapcsolatot, amely az `Event` modell osztályhoz tartozó `events` táblában található `user_id` attribútumot fogja használni a tulajdonolt események azonosítására. Azon eseményeket, amelyeken a felhasználó részt vesz az `events` azonosítójú `has_and_belongs_to_many` kapcsolaton keresztül érjük el. Egy esemény egy tulajdonos felhasználóhoz tartozik, ezt a kapcsolatot a `owner` azonosítójú attribútum realizálja, amely egy `User` típusú objektumra hivatkozik, amelyet az események tábla `user_id` attribútuma jelöl ki. Egy eseményen sok felhasználó is részt vehet, ezt a kapcsolatot a `users` azonosítójú attribútum valósítja meg, amely a kapcsolótáblára hivatkozik.

```
class Event < ApplicationRecord
  belongs_to :owner, class_name: 'User', foreign_key: 'user_id'
  has_and_belongs_to_many :users
```

```
end
class User < ApplicationRecord
  has_many :owned_events, class_name: "Event", foreign_key: 'user_id'
  has_and_belongs_to_many :events
end
```

Hajtsuk végre a kölcsönzések modell migrációját.

```
kovacsg@debian:~/gyakorlat/db/migrate$ rails db:migrate
== 20180327113501 CreateJoinTableEventsUsers: migrating
   ======================
-- create_join_table(:events, :users)
   -> 0.0214s
== 20180327113501 CreateJoinTableEventsUsers: migrated (0.0217s)
   =============
```

Hozzunk létre konzolon néhány kapcsolatot felhasználók és események között! Keressünk elő egy eseményt (1. sor), és nézzük meg, hogy működik-e az `owner` kapcsolatunk (2. sor). Keressünk elő egy felhasználót (3. sor), és nézzük meg, hogy mely eseményeket tulajdonol (5. sor), illetve mely eseményeken vesz részt (7. sor) – ez üres tömböt ad vissza. Nézzük meg, hogy az előkeresett eseménynek kik a résztvevői (9. sor) – ez is üres tömböt ad vissza. Vegyük fel a felhasználónkat résztvevőnek a saját eseményére (10. sor), majd nézzük meg az esemény résztvevőit (11. sor) és a felhasználó által látogatott eseményeket (12. sor).

```
kovacsg@debian:~/gyakorlat/app/models$ rails c
Loading development environment (Rails 5.1.5)
irb(main):001:0> e = Event.take
   (0.3ms)  SET NAMES utf8,  @@SESSION.sql_mode = CONCAT(CONCAT(@@sql_mode,
       ',STRICT_ALL_TABLES'), ',NO_AUTO_VALUE_ON_ZERO'),  @@SESSION.
       sql_auto_is_null = 0, @@SESSION.wait_timeout = 2147483
  Event Load (0.1ms)  SELECT  `events`.* FROM `events` LIMIT 1
=> #<Event id: 1, user_id: 1, title: "Proba", description: "Zenekari proba,
    kozos enekles", deadline: "2018-03-13 00:00:00", priority: 5, created_at
    : "2018-02-27 12:40:47", updated_at: "2018-02-27 12:40:47">
irb(main):002:0> e.owner
  User Load (0.4ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 1
       LIMIT 1
=> #<User id: 1, name: "Valaki", email: "valaki@mail.bme.hu",
    encrypted_password: "titok", created_at: "2018-02-27 12:31:23",
    updated_at: "2018-02-27 12:31:23", salt: nil>
irb(main):003:0> u = User.find 1
  User Load (0.6ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 1
       LIMIT 1
=> #<User id: 1, name: "Valaki", email: "valaki@mail.bme.hu",
    encrypted_password: "titok", created_at: "2018-02-27 12:31:23",
    updated_at: "2018-02-27 12:31:23", salt: nil>
irb(main):005:0> u.owned_events
  Event Load (3.3ms)  SELECT  `events`.* FROM `events` WHERE `events`.`
       user_id` = 1 LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy [#<Event id: 1, user_id: 1,
     title: "Proba", description: "Zenekari proba, kozos enekles", deadline:
     "2018-03-13 00:00:00", priority: 5, created_at: "2018-02-27 12:40:47",
    updated_at: "2018-02-27 12:40:47">]>
irb(main):007:0> u.events
```

```
  Event Load (0.3ms)  SELECT `events`.* FROM `events` INNER JOIN `
      events_users` ON `events`.`id` = `events_users`.`event_id` WHERE `
      events_users`.`user_id` = 1 LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy []>
irb(main):008:0> e
=> #<Event id: 1, user_id: 1, title: "Proba", description: "Zenekari proba,
    kozos enekles", deadline: "2018-03-13 00:00:00", priority: 5, created_at
    : "2018-02-27 12:40:47", updated_at: "2018-02-27 12:40:47">
irb(main):009:0> e.users
  User Load (0.4ms)  SELECT `users`.* FROM `users` INNER JOIN `events_users
      ` ON `users`.`id` = `events_users`.`user_id` WHERE `events_users`.`
      event_id` = 1 LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy []>
irb(main):010:0> e.users << u
   (0.7ms)  BEGIN
  SQL (0.2ms)  INSERT INTO `events_users` (`event_id`, `user_id`) VALUES (1,
       1)
   (7.8ms)  COMMIT
  User Load (0.4ms)  SELECT `users`.* FROM `users` INNER JOIN `events_users
      ` ON `users`.`id` = `events_users`.`user_id` WHERE `events_users`.`
      event_id` = 1 LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy [#<User id: 1, name: "
    Valaki", email: "valaki@mail.bme.hu", encrypted_password: "titok",
    created_at: "2018-02-27 12:31:23", updated_at: "2018-02-27 12:31:23",
    salt: nil>]>
irb(main):011:0> e.users
  User Load (0.3ms)  SELECT `users`.* FROM `users` INNER JOIN `events_users
      ` ON `users`.`id` = `events_users`.`user_id` WHERE `events_users`.`
      event_id` = 1 LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy [#<User id: 1, name: "
    Valaki", email: "valaki@mail.bme.hu", encrypted_password: "titok",
    created_at: "2018-02-27 12:31:23", updated_at: "2018-02-27 12:31:23",
    salt: nil>]>
irb(main):012:0> u
=> #<User id: 1, name: "Valaki", email: "valaki@mail.bme.hu",
    encrypted_password: "titok", created_at: "2018-02-27 12:31:23",
    updated_at: "2018-02-27 12:31:23", salt: nil>
irb(main):013:0> u.events
  Event Load (0.5ms)  SELECT `events`.* FROM `events` INNER JOIN `
      events_users` ON `events`.`id` = `events_users`.`event_id` WHERE `
      events_users`.`user_id` = 1 LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy [#<Event id: 1, user_id: 1,
    title: "Proba", description: "Zenekari proba, kozos enekles", deadline:
    "2018-03-13 00:00:00", priority: 5, created_at: "2018-02-27 12:40:47",
    updated_at: "2018-02-27 12:40:47">]>
```

Vegyünk fel még egy modellt, ami legyen a cím. Mivel a felhasználónak lehet lakcíme, és az eseménynek lehet helyszíne, ugyanazt a típust újrahasznosíthatjuk, ha polimorfikus kapcsolatot definiálunk. Vegyünk fel először egy cím típust, mely álljon string típusú városnévből, string típusú irányítószámból, string típusú utcanévből, egész típusú házszámból, és string típusú, a címet pontosító tagokat tartalmazó attribútumból. Az automatikusan generált migrációt megtarthatjuk abban a formában.

```
kovacsg@debian:~/gyakorlat/app/models$ rails g model Address city:string zip
    :string street:string number:integer others:string
      invoke  active_record
      create    db/migrate/20180327114256_create_addresses.rb
      create    app/models/address.rb
```

```
      invoke    test_unit
      create      test/models/address_test.rb
      create      test/fixtures/addresses.yml
```

Polimorfikus relációnál egy cím egy valamely címezhető rekordhoz kell tartoznia, vagyis szükségünk van még egy idegen kulcsre is.

```
  kovacsg@debian:~/gyakorlat/app/models$ rails g migration
      AddAddressableToAddress addressable:references
      invoke  active_record
      create    db/migrate/20180327114337_add_addressable_to_address.rb
```

Az automatikusan generált migrációt módosítanunk kell, fel kell tüntetnünk, hogy ez egy polimorfikus kapcsolat.

```
class AddAddressableToAddress < ActiveRecord::Migration[5.1]
  def change
    add_reference :addresses, :addressable, polymorphic: true, index: true
  end
end
```

Hajtsuk végre a migrációt, majd a modell osztályaink között is kialakítjuk a kapcsolatokat. Egy felhasználónak egy lakcíme van, egy eseménynek egy helyszíne van, mindkettőre az `address` attribútummal hivatkozunk a megfelelő modell osztályban, de tudatjuk a Rails-szel, hogy a kapcsolazhoz a `addressable_id` attribútumot kell használnia az `Address` modell osztályhoz tartozó `addresses` táblában. Egy cím az valamely címezhető objektumhoz kapcsolódik, ami lehet felhasználó vagy esemény, ezért az asszociációban jeleznünk kell, hogy az polimorfikus.

```
class Event < ApplicationRecord
  has_one :address, as: :addressable
end
class User < ApplicationRecord
  has_one :address, as: :addressable
end
class Address < ApplicationRecord
  belongs_to :addressable, polymorphic: true
end
```

Az adatbázisunkba adatokat vettünk fel, amikor létrehozunk felhasználót, eseményt, és meghatároztuk a közöttük lévő kapcsolatokat. Ezeket a kezdeti adatokat reprodukálható módon is bevihetjük anélkül, hogy a Rails konzolt használnánk. A `db/seeds.rb` fájlban elhelyezett Ruby metódushívásokkal tudunk adatokat felvenni. 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. Az adatokatokat ezek megadása után a következő paranccsal tudjuk betölteni az adatbázisba:

```
rails db:seed
```