Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2019. április 2.

1. Session, felhasználókezelelés

1.1. Titkosított jelszó

Az előző gyakorlaton megkezdett példát folytatjuk a felhasználói sessionök kezelésének megvalósításával, illetve az adatmodell kialakításával. Az előző alkalommak öt modellt hoztunk létre, a felhasználók `User` nevű modelljét, a topikok `Topic` nevű modelljét, a kommentek `Comment` nevű modelljét, a linkek `Link` nevű modelljét és a csatolmányok `Attachment` nevű modelljét. A felhasználók modellje a következőképp néz ki az adatbázisban.

```
MariaDB [gyakorlat_development]> show tables;
+-----------------------------------+
| Tables_in_gyakorlat_development |
+-----------------------------------+
| ar_internal_metadata              |
| attachments                       |
| comments                          |
| links                             |
| schema_migrations                 |
| subtopics_topics                  |
| topics                            |
| users                             |
+-----------------------------------+
8 rows in set (0.00 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 (0.00 sec)


MariaDB [gyakorlat_development]> select * from users;
+----+---------+-------------------+----------+---------------------+---------------------+


| id | name    | email             | password | created_at          |
    updated_at          |
+----+---------+-------------------+----------+---------------------+---------------------+

|  1 | Senki   | senki2@mail.bme.hu | titok | 2019-02-26 12:28:41 |
    2019-03-19 18:33:15 |
+----+---------+-------------------+----------+---------------------+---------------------+

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/app# rails g migration AddSaltToUsers salt:string
      invoke  active_record
      create    db/migrate/20190319182136_add_salt_to_users.rb
```

Nézzük meg, milyen hash függvények állnak a rendelkezésünkre, amelyek felhasználhatók a titkosítás során. Szükségünk lesz véletlen és egyben olvashatatlan karaktersorozatokra, és stringből olvashatatlan karaktersorozatot előállító függvényre.

```
kovacsg@debian:~/gyakorlat# rails c
Loading development environment (Rails 5.2.1)
irb(main):001:0> SecureRandom
=> SecureRandom
irb(main):002:0> SecureRandom.hex 8
=> "1836f3ffa6479721"
irb(main):003:0> SecureRandom.hex 16
=> "fc89a02752686c040d93c3759964a5ae"
irb(main):004:0> SecureRandom.base64 16
=> "N0jIW99BHYKykuWEG+XzuQ=="
irb(main):006:0> Digest::SHA1.hexdigest 'titok'
=> "46ff53e764c4acf97b54db2020573049d2e3dab3"
```

```
irb(main):007:0>
```

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[5.2]
  # 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
    drop_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
== 20190319182136 AddSaltToUsers: migrating
   ================================
-- add_column(:users, :salt, :string)
   -> 0.0377s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0077s
== 20190319182136 AddSaltToUsers: migrated (0.0456s)
   ===========================
```

Nézzük meg a migráció eredményét. Az adatbázisban már jelen lévő rekorddal nem tudunk mit kezdeni. Ugyan titkosíthatjuk a jelszó attribútumát, de egy le-, majd egy felirányú migrációval az érvénytelen lesz, hiszen a hash egy egyirányú művelet, és így nem visszaállítható. Azt a rekordot meg kell semmisítenünk.

```
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    |
   |
| encrypted_password  | varchar(255)  | YES  |      | NULL    |
   |
| created_at          | datetime      | NO   |      | NULL    |
   |
| updated_at          | datetime      | NO   |      | NULL    |
   |
| salt                | varchar(255)  | YES  |      | NULL    |
   |
+---------------------+---------------+------+------+---------+-----------------+

7 rows in set (0.00 sec)

MariaDB [gyakorlat_development]> select * from users;
+----+----------+--------------------+------------------------------------------+

| id | name     | email              | encrypted_password
                          | created_at           | updated_at            | salt
     |
+----+----------+--------------------+------------------------------------------+

|  1 | Senki    | senki2@mail.bme.hu | titok | 2019-02-26 12:28:41 |
    2019-03-19 18:33:15 | FF41NwIkKyv43mgzbbnNig== |
+----+----------+--------------------+------------------------------------------+

1 row in set (0.00 sec)


MariaDB [gyakorlat_development]> Bye
```

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.

A migrációban minden egyes felhasználói rekordhoz hozzáadtunk egy egyéni a jelszó titkosításához használt random kulcs tárolólására 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 self.new_record?
      self.salt = SecureRandom.base64(16)
    end
    self.encrypted_password = User.encrypt(password, salt)
  end
end
```

Nézzük meg, hogy milyen elemi műveletekkel férhetünk egy vagy több adatbázisbeli rekordhoz hozzá Rails-ből. A `first` (6. sor), `last` (5. 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), 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# rails c
Loading development environment (Rails 5.2.1)
irb(main):001:0> User.find_by email: 'senki@mail.bme.hu'
   (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 (0.3ms)  SELECT  `users`.* FROM `users` WHERE `users`.`email` =
      'senki@mail.bme.hu' LIMIT 1
=> #<User id: 2, name: "Senki", encrypted_password: "
    efb935d3efbc456d8d27a218a4078408bf2888a5", email: "senki@mail.bme.hu",
    neptun: nil, usertype: 0, created_at: "2018-10-30 11:30:40", updated_at:
     "2018-10-30 11:30:40", salt: "Zf3gg8BdfWg=">
irb(main):002:0> User.where email: 'senki@mail.bme.hu'
  User Load (0.4ms)  SELECT  `users`.* FROM `users` WHERE `users`.`email` =
      'senki@mail.bme.hu' LIMIT 11
=> #<ActiveRecord::Relation [#<User id: 2, name: "Senki", encrypted_password
    : "efb935d3efbc456d8d27a218a4078408bf2888a5", email: "senki@mail.bme.hu
    ", neptun: nil, usertype: 0, created_at: "2018-10-30 11:30:40",
```

```
    updated_at: "2018-10-30 11:30:40", salt: "Zf3gg8BdfWg=">]>
irb(main):005:0> User.last
  User Load (0.3ms)  SELECT  `users`.* FROM `users` ORDER BY `users`.`id`
      DESC LIMIT 1
=> #<User id: 2, name: "Senki", encrypted_password: "
    efb935d3efbc456d8d27a218a4078408bf2888a5", email: "senki@mail.bme.hu",
    neptun: nil, usertype: 0, created_at: "2018-10-30 11:30:40", updated_at:
     "2018-10-30 11:30:40", salt: "Zf3gg8BdfWg=">
irb(main):006:0> User.first
  User Load (0.3ms)  SELECT  `users`.* FROM `users` ORDER BY `users`.`id`
      ASC LIMIT 1
=> #<User id: 1, name: "Valaki", encrypted_password: "titok", email: "
    valaki@mail.bme.hu", neptun: "aaaaaa", usertype: 0, created_at:
    "2018-09-25 11:18:40", updated_at: "2018-09-25 11:18:40", salt: nil>
irb(main):007:0> User.take
  User Load (0.3ms)  SELECT  `users`.* FROM `users` LIMIT 1
=> #<User id: 1, name: "Valaki", encrypted_password: "titok", email: "
    valaki@mail.bme.hu", neptun: "aaaaaa", usertype: 0, created_at:
    "2018-09-25 11:18:40", updated_at: "2018-09-25 11:18:40", salt: nil>
irb(main):008:0> User.take(2)
  User Load (0.4ms)  SELECT  `users`.* FROM `users` LIMIT 2
=> [#<User id: 1, name: "Valaki", encrypted_password: "titok", email: "
    valaki@mail.bme.hu", neptun: "aaaaaa", usertype: 0, created_at:
    "2018-09-25 11:18:40", updated_at: "2018-09-25 11:18:40", salt: nil>, #<
    User id: 2, name: "Senki", encrypted_password: "
    efb935d3efbc456d8d27a218a4078408bf2888a5", email: "senki@mail.bme.hu",
    neptun: nil, usertype: 0, created_at: "2018-10-30 11:30:40", updated_at:
     "2018-10-30 11:30:40", salt: "Zf3gg8BdfWg=">]
irb(main):009:0> User.where(email: 'senki@mail.bme.hu').first
  User Load (0.4ms)  SELECT  `users`.* FROM `users` WHERE `users`.`email` =
     'senki@mail.bme.hu' ORDER BY `users`.`id` ASC LIMIT 1
=> #<User id: 2, name: "Senki", encrypted_password: "
    efb935d3efbc456d8d27a218a4078408bf2888a5", email: "senki@mail.bme.hu",
    neptun: nil, usertype: 0, created_at: "2018-10-30 11:30:40", updated_at:
     "2018-10-30 11:30:40", salt: "Zf3gg8BdfWg=">
irb(main):010:0>
```

Elsődleges kulcs alapján, amely jellemzően a weboldalon megjelenő azonosító önmaga, a `find` metódussal kereshetünk egy objektumot. Egy objektum törlése is történhet az azonosító alapján. 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 (előző kódblokk 4. 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.

```
irb(main):014:0> User.find 1
  User Load (0.4ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 1
      LIMIT 1
=> #<User id: 1, name: "Valaki", encrypted_password: "titok", email: "
```

```
    valaki@mail.bme.hu", neptun: "aaaaaa", usertype: 0, created_at:
    "2018-09-25 11:18:40", updated_at: "2018-09-25 11:18:40", salt: nil>
irb(main):015:0> User.delete 1
  User Destroy (10.8ms)  DELETE FROM `users` WHERE `users`.`id` = 1
=> 1
```

Hozzunk létre egy felhasználót, és nézzük meg, hogy működik-e a jelszó titkosítássa.

```
kovacsg@debian:~/gyakorlat# rails c
Loading development environment (Rails 5.2.1)
irb(main):001:0> u = User.new
   (0.4ms)  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 id: nil, name: nil, encrypted_password: nil, email: nil, neptun:
    nil, usertype: nil, created_at: nil, updated_at: nil, salt: nil>
irb(main):002:0> u.name = 'Senki'
=> "Senki"
irb(main):003:0> u.password = 'titok'
=> "titok"
irb(main):004:0> u.email = 'senki@mail.bme.hu'
=> "senki@mail.bme.hu"
irb(main):006:0> u.save
   (0.2ms)  BEGIN
  User Create (0.2ms)  INSERT INTO `users` (`name`, `encrypted_password`, `
      email`, `usertype`, `created_at`, `updated_at`, `salt`) VALUES ('Senki
      ', 'efb935d3efbc456d8d27a218a4078408bf2888a5', 'senki@mail.bme.hu', 0,
       '2018-10-30 11:30:40', '2018-10-30 11:30:40', 'Zf3gg8BdfWg=')
   (7.4ms)  COMMIT
=> true
```

1.2. Be- és kijelentkezés

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.

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# rails g controller sessions create destroy
      create  app/controllers/sessions_controller.rb
       route  get 'sessions/create'
get 'sessions/destroy'
      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
      invoke    coffee
      create      app/assets/javascripts/sessions.coffee
      invoke    scss
      create      app/assets/stylesheets/sessions.scss
```

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/_guest_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 vagy DELETE üzenet elég. Valósítsuk ezeket meg, és rendeljük hozzájuk a `login`, illetve a `logout` címkéket. Ez utóbbiak `login_path` és `login_url` azonosítóval segédfüggvényeket hoznak létre, amelyekkel az útvonalra, illetve a teljes URL-re hivatkozjatunk a `login` alias esetén, a `logout` alias hasonlóan működik. Végül nevezzük el a hello világ nézetünket `hello`-nak.

```
Rails.application.routes.draw do
  post 'sessions/create', to: 'sessions#create', as: 'login'
  match 'sessions/destroy', to: 'sessions#destroy', via: [:get, :delete], as
      : 'logout'

  get 'say/hello', to: 'say#hello', as: 'hello'
end
```

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, password)
    user = User.where(email: email).take
    user && user.authenticated?(password) ? user : nil
  end

  def authenticated?(password)
    self.encrypted_password == User.encrypt(password, 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. Az előző oldal vagy a JavaScript history-ból, vagy a kérés fejrészének `HTTP_REFERER` opciójából határozható meg, ha egyik sem adott, akkor a helló világ oldalra kerükjünk át.

```
class SessionsController < ApplicationController
  def create
    @user = User.authenticate(params[:email], params[:password])
    if @user
      session[:user] = @user.id
      redirect_back fallback_location: hello_path
    else
      flash[:notice] = "Invalid email or password"
      redirect_back fallback_location: hello_path
    end
  end

  def destroy
    reset_session
    flash[:notice] = 'Logged out successfully'
    redirect_back fallback_location: hello_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 a menü felett 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
```

1.3. Regisztráció, profil szerkesztése

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 login'
      redirect_back fallback_location: hello_path
    else
      flash[:notice] = @user.errors.messages
      redirect_back fallback_location: hello_path
    end
  end

  def edit
  end

  def update
    if @user.update(user_params)
      flash[:notice] = "Update successful"
      redirect_back fallback_location: hello_path
    else
      flash[:notice] = "Could not update data"
      redirect_back fallback_location: hello_path
    end
  end

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

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. A felhasználó azonosító alapján való előrekeresésére több akció esetén is szükségünk van, de nem akarjuk többször ugyanazt a kódrészletet leírni, ezért felhasználjuk a `before_action` függvényt, melynek argumentuma egy függvényazonosító. Mivel több kontroller esetén is szükségünk van a felhasználóra, a keresést a kontrollerek közös ősosztályában tesszük meg. Az a függvény a kontroller összes publikus akciója előtt felut,

ha benne van az `only` utáni felsorolásban, vagy nincs benne a `except` utáni felsorolásban az akció azonosítója. Ha egyik felsorolás sincs megadva, akkor mindenképp lefut a függvény. Ez a kódunk karbantarthatóságát javítja.

```
class ApplicationController < ActionController::Base
  before_action :find_user

  private
    def find_user
      @user = User.find session[:user]
    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', to: 'users#new', as: 'register'
  post 'users/create', to: 'users#create', as: 'create'
  get 'users/edit/:id', to: 'users#edit', as: 'profile'
  put 'users/update/:id', to: 'users#update', as: 'update'
  get 'users/forgotten'
  post 'users/send_forgotten'
```

1.4. Felhasználó által megadott adatok ellenőrzése

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 :name, presence: true
  validates :email, { presence: true, uniqueness: 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# rails c
Loading development environment (Rails 5.2.2)
irb(main):001:0> User.new
   (1.9ms)  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 id: nil, name: nil, email: nil, encrypted_password: nil,
    created_at: nil, updated_at: nil, salt: nil>
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.save
   (0.2ms)  BEGIN
  User Exists (0.4ms)  SELECT  1 AS one FROM `users` WHERE `users`.`email`
      IS NULL LIMIT 1
   (0.1ms)  ROLLBACK
=> false
irb(main):004:0> u.errors.messages
=> {:name=>["can't be blank"], :email=>["can't be blank"]}
irb(main):005:0> u.name = 'Valaki'
=> "Valaki"
irb(main):006:0> u.email = 'senki2@mail.bme.hu'
=> "senki2@mail.bme.hu"
irb(main):007:0> u.save
   (0.2ms)  BEGIN
  User Exists (0.3ms)  SELECT  1 AS one FROM `users` WHERE `users`.`email` =
      BINARY 'senki2@mail.bme.hu' LIMIT 1
   (0.2ms)  ROLLBACK
=> false
irb(main):008:0> 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.

2. Az adatmodell kialakítása

2.1. Modellosztályok kapcsolatai

A feladatjavító portálunk adatmodellje a következő elemekből áll:

- Felhasználó (`User`), aki lehet hallgató vagy oktató
- Téma (`Topic`)
- Csatolmány (`Attachment`)
- Link (`Link`)
- Komment (`Comment`)

Ezeket a korábbi gyakorlatokon már létrehoztuk.

Egy téma (`Topic`) a létrehozó felhasználó tulajdonában áll. A `references` típus a migrációban automatikusan létrehozza a `belongs_to` deklarációt. Mindemellett egy témához tartozhat sok csatolmány, komment, illetve link.

```
class Topic < ApplicationRecord
  belongs_to :user
  has_many :comments
  has_many :commenters, through: :comments, source: 'user'
  has_many :attachments
  has_many :links
end
```

Egy link (`Link`) egy témához és a létrehozó felhasználóhoz tartozik, ezek a kapcsolatok már léteznek.

```
class Link < ApplicationRecord
  belongs_to :topic
  belongs_to :user
end
```

Hasonlóak a linkhez, egy csatolmány (`Attachment`) is egy témához és a létrehozó felhasználóhoz tartozik, ezek a kapcsolatok is léteznek.

```
class Attachment < ApplicationRecord
  belongs_to :topic
  belongs_to :user
end
```

Hasonlóak a linkhez, egy komment (`Comment`) is egy témához és a létrehozó felhasználóhoz tartozik, ezek a kapcsolatok is léteznek.

```
class Comment < ApplicationRecord
  belongs_to :topic
  belongs_to :user
end
```

Ha kíváncsiak vagyunk, hogy egy témához kik szóltak hozzá, azt a következő kapcsolattal valósíthatjuk meg a `Topic` modellben. A kommentek ekkor kapcsolótáblaként működik a témák és a felhasználók modellek között. A kommentelők is felhasználó típusúak, ezért ez a kapcsolat nem keverendő össze a téma tulajdonosával.

```
class Topic < ApplicationRecord
  has_many :commenters, through: :comments, source: 'user'
end
```

A témáknak lehetnek altémái is, ezért felveszünk egy új táblát, amely összekapcsolja a `topics` táblát önmagával.

```
kovacsg@debian:~/gyakorlat# rails g migration CreateTopicsSubtopics topics:
    references subtopics:references
```

A migrációban egy kapcsolótáblát hozunk létre a két összekapcsolandó tábla nevének megadásával.

```
class CreateTopicsSubtopics < ActiveRecord::Migration[5.2]
  def change
    create_join_table :subtopics, :topics
  end
end
```

Majd végrehajtjuk a migrációt.

```
kovacsg@debian:~/gyakorlat# rails db:migrate
== 20190319192418 CreateTopicsSubtopics: migrating
   ============================
-- create_join_table(:subtopics, :topics)
   -> 0.0136s
== 20190319192418 CreateTopicsSubtopics: migrated (0.0138s)
   ===================
```

A topikok között több-több kapcsolatot hoztunk létre e kapcsolótáblán keresztül, amelyben az aktuális topikra a `topic_id`, a kapcsolt topikra a `subtopic_id` hivatkozik, és annak a típusa is `Topic`, ami megadandó mert a `subtopic` táblánk valójában nincs.

```
class Topic < ApplicationRecord
  has_and_belongs_to_many :subtopics, class_name: 'Topic', foreign_key: '
      subtopic_id', join_table: 'subtopics_topics',
      assossication_foreign_key: 'topic_id'
end
```

2.2. Kezdeti adatok felvétele

Az adatmodellünk készen van, de nem tudjuk ellenőrizni, hogy jó-e, mert az adatbázisban nincs adatunk. Adatok felvételére több mód is van. A leglassabb a webfelület használata, ennél gyorsabb a Rails konzolon való adatrögzítés. Ha azt szeretnénk, hogy a konzolon felvett adatok reprodukálhatóan meglegyenek, akkor a konzolba írandó utasásokat a `db/seed.rb` fájlban helyezzük el. Vegyük fel topikokat, majd konzolon ellenőrizzük a kapcsolatokat.

```
for i in 1..10 do
  Topic.create title: "Title#{i}", user: User.find(1), contents: "Contents␣
      of␣topic␣#{i}"
end
```

Töltsük be ezeket az adatokat:

```
kovacsg@debian:~/gyakorlat/db# rails db:seed
```

Konzolon végezzünk el néhány műveletet: keressük elő az utolsó témát (1. sor), nézzük meg, hogy milyen altémái vannak (2. sor), rendeljünk két témát altémaként hozzá (3-4. sor), majd nézzünk meg, hogy sikerült-e művelet (5. sor). Ezután egy felhasználóval kommentet fűzetünk ehhez a topikhoz. A 6. sorban látjuk, hogy még nincs kommentünk a témához. A 7. sorban kinézünk magunknak egy felhasználót, akit a 9. sorban létrehozott kommentben felhasználunk. A 10. sorban látjuk, hogy a komment létrejött, és a 12-13. sor pedig megerősíti, hogy a kommentelőnk valóban a kiválasztott felhasználó volt.

```
kovacsg@debian:~/gyakorlat$ rails c
irb(main):001:0> Topic.last
   (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
  Topic Load (0.1ms)  SELECT  `topics`.* FROM `topics` ORDER BY `topics`.`id
      ` DESC LIMIT 1
=> #<Topic id: 12, title: "Title10", user_id: 1, contents: "Contents of
    topic 10", created_at: "2019-03-19 19:22:10", updated_at: "2019-03-19
    19:22:10">

irb(main):002:0> Topic.last.subtopics
  Topic Load (0.2ms)  SELECT  `topics`.* FROM `topics` ORDER BY `topics`.`id
      ` DESC LIMIT 1
  Topic Load (0.4ms)  SELECT  `topics`.* FROM `topics` INNER JOIN `
      subtopics_topics` ON `topics`.`id` = `subtopics_topics`.`topic_id`
      WHERE `subtopics_topics`.`subtopic_id` = 12 LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy []>
```

```
irb(main):003:0> Topic.last.subtopics << Topic.find(8)
  Topic Load (0.2ms)  SELECT `topics`.* FROM `topics` ORDER BY `topics`.`id`
      DESC LIMIT 1
  Topic Load (0.3ms)  SELECT `topics`.* FROM `topics` WHERE `topics`.`id` =
       8 LIMIT 1
   (0.1ms)  BEGIN
  Topic::HABTM_Subtopics Create (7.2ms)  INSERT INTO `subtopics_topics` (`
      subtopic_id`, `topic_id`) VALUES (12, 8)
   (2.1ms)  COMMIT
  Topic Load (0.4ms)  SELECT `topics`.* FROM `topics` INNER JOIN `
      subtopics_topics` ON `topics`.`id` = `subtopics_topics`.`topic_id`
      WHERE `subtopics_topics`.`subtopic_id` = 12 LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy [#<Topic id: 8, title: "
    Title6", user_id: 1, contents: "Contents of topic 6", created_at:
    "2019-03-19 19:22:10", updated_at: "2019-03-19 19:22:10">]>

irb(main):004:0> Topic.last.subtopics << Topic.find(9)
  Topic Load (0.6ms)  SELECT `topics`.* FROM `topics` ORDER BY `topics`.`id`
      DESC LIMIT 1
  Topic Load (0.3ms)  SELECT `topics`.* FROM `topics` WHERE `topics`.`id` =
       9 LIMIT 1
   (0.1ms)  BEGIN
  Topic::HABTM_Subtopics Create (0.9ms)  INSERT INTO `subtopics_topics` (`
      subtopic_id`, `topic_id`) VALUES (12, 9)
   (3.3ms)  COMMIT
  Topic Load (0.4ms)  SELECT `topics`.* FROM `topics` INNER JOIN `
      subtopics_topics` ON `topics`.`id` = `subtopics_topics`.`topic_id`
      WHERE `subtopics_topics`.`subtopic_id` = 12 LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy [#<Topic id: 8, title: "
    Title6", user_id: 1, contents: "Contents of topic 6", created_at:
    "2019-03-19 19:22:10", updated_at: "2019-03-19 19:22:10">, #<Topic id:
    9, title: "Title7", user_id: 1, contents: "Contents of topic 7",
    created_at: "2019-03-19 19:22:10", updated_at: "2019-03-19 19:22:10">]>

irb(main):005:0> Topic.last.subtopics
  Topic Load (0.2ms)  SELECT `topics`.* FROM `topics` ORDER BY `topics`.`id`
      DESC LIMIT 1
  Topic Load (0.1ms)  SELECT `topics`.* FROM `topics` INNER JOIN `
      subtopics_topics` ON `topics`.`id` = `subtopics_topics`.`topic_id`
      WHERE `subtopics_topics`.`subtopic_id` = 12 LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy [#<Topic id: 8, title: "
    Title6", user_id: 1, contents: "Contents of topic 6", created_at:
    "2019-03-19 19:22:10", updated_at: "2019-03-19 19:22:10">, #<Topic id:
    9, title: "Title7", user_id: 1, contents: "Contents of topic 7",
    created_at: "2019-03-19 19:22:10", updated_at: "2019-03-19 19:22:10">]>

irb(main):006:0> Topic.last.comments
  Topic Load (0.2ms)  SELECT `topics`.* FROM `topics` ORDER BY `topics`.`id`
      DESC LIMIT 1
  Comment Load (0.3ms)  SELECT `comments`.* FROM `comments` WHERE `comments
      `.`topic_id` = 12 LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy []>

irb(main):007:0> Topic.last.user
  Topic Load (0.2ms)  SELECT `topics`.* FROM `topics` ORDER BY `topics`.`id`
      DESC LIMIT 1
  User Load (0.1ms)  SELECT `users`.* FROM `users` WHERE `users`.`id` = 1
      LIMIT 1
=> #<User id: 1, name: "Senki", email: "senki2@mail.bme.hu",
    encrypted_password: "ae8538707f4712236534daa1cba7a4456692bdd06",
    created_at: "2019-02-26 12:28:41", updated_at: "2019-03-19 18:33:15",
    salt: "FF41NwIkKyv43mgzbbnNig==">
```

```
irb(main):008:0> Comment.new
=> #<Comment id: nil, topic_id: nil, user_id: nil, comment: nil, created_at:
     nil, updated_at: nil>
irb(main):009:0> Comment.create topic: Topic.last, user: User.last, comment:
     "Jo reggelt"
  Topic Load (0.2ms)  SELECT `topics`.* FROM `topics` ORDER BY `topics`.`id
     ` DESC LIMIT 1
  User Load (0.3ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id`
     DESC LIMIT 1
   (0.2ms)  BEGIN
  Comment Create (6.7ms)  INSERT INTO `comments` (`topic_id`, `user_id`, `
     comment`, `created_at`, `updated_at`) VALUES (12, 2, 'Jo reggelt', '
     2019-03-19 19:35:36', '2019-03-19 19:35:36')
   (2.3ms)  COMMIT
=> #<Comment id: 1, topic_id: 12, user_id: 2, comment: "Jo reggelt",
    created_at: "2019-03-19 19:35:36", updated_at: "2019-03-19 19:35:36">

irb(main):010:0> Topic.last.comments
  Topic Load (0.2ms)  SELECT `topics`.* FROM `topics` ORDER BY `topics`.`id
     ` DESC LIMIT 1
  Comment Load (0.3ms)  SELECT `comments`.* FROM `comments` WHERE `comments
     `.`topic_id` = 12 LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy [#<Comment id: 1, topic_id:
     12, user_id: 2, comment: "Jo reggelt", created_at: "2019-03-19
    19:35:36", updated_at: "2019-03-19 19:35:36">]>

irb(main):012:0> Topic.last.comments.last.user
  Topic Load (0.2ms)  SELECT `topics`.* FROM `topics` ORDER BY `topics`.`id
     ` DESC LIMIT 1
  Comment Load (0.3ms)  SELECT `comments`.* FROM `comments` WHERE `comments
     `.`topic_id` = 12 ORDER BY `comments`.`id` DESC LIMIT 1
  User Load (0.2ms)  SELECT `users`.* FROM `users` WHERE `users`.`id` = 2
     LIMIT 1
=> #<User id: 2, name: "Valaki", email: "valaki@mail.bme.hu",
    encrypted_password: "d7c23def9c7706ca521a3eb308ad1897fdaaa380",
    created_at: "2019-03-19 19:07:42", updated_at: "2019-03-19 19:07:42",
    salt: "N/yhVsyatITn/aLZY7yXKQ==">

irb(main):013:0> Topic.last.commenters
   (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
  Topic Load (0.1ms)  SELECT `topics`.* FROM `topics` ORDER BY `topics`.`id
     ` DESC LIMIT 1
  User Load (0.3ms)  SELECT `users`.* FROM `users` INNER JOIN `comments` ON
     `users`.`id` = `comments`.`user_id` WHERE `comments`.`topic_id` = 12
     LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy [#<User id: 2, name: "
    Valaki", email: "valaki@mail.bme.hu", encrypted_password: "
    d7c23def9c7706ca521a3eb308ad1897fdaaa380", created_at: "2019-03-19
    19:07:42", updated_at: "2019-03-19 19:07:42", salt: "N/yhVsyatITn/
    aLZY7yXKQ==">]>
```