Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2022. április 12.

1. Session, felhasználókezelelés

1.1. Titkosított jelszó

Az előző gyakorlaton megkezdett példát folytatjuk a felhasználói sessionök kezelésének megvalósításával, illetve az adatmodell kialakításával. Az előző alkalommal három modellt hoztunk létre, a felhasználók `User` nevű modelljét, a pártok `Party` nevű modelljét, valamint a jelöltek `Candidate` nevű modelljét. A felhasználók modellje a következőképp néz ki az adatbázisban.

```
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    |                |
| idnum                 | varchar(255) | YES  |     | NULL    |                |
| birthdate             | datetime(6)  | YES  |     | NULL    |                |
| account               | varchar(255) | YES  |     | NULL    |                |
| created_at            | datetime(6)  | NO   |     | NULL    |                |
| updated_at            | datetime(6)  | NO   |     | NULL    |                |
+-----------------------+--------------+------+-----+---------+----------------+
9rows in set (0.001 sec)
```

```
MariaDB [gyakorlat_development]> select * from users;
+----+---------+-------------------+----------+--------+---------------------
| id | name    | email             | password | idnum | birthdate
                    | account           | created_at                  |
    updated_at                  |
+----+---------+-------------------+----------+--------+---------------------

|  1 | Valaki | valaki@mail.bme.hu | titok    | NULL  | 2022-03-22
    00:00:00.000000 | 11111111-11111111 | 2022-03-22 12:25:22.020308 |
    2022-03-22 12:25:22.020308 |
+----+---------+-------------------+----------+--------+---------------------

1 row in set (0.004 sec)
```

Először növeljük portálunk biztonságát azzal, hogy a jelszavakat nem szövegesen, hanem titkosítva tároljuk. Erre használhatjuk a Devise nevű API-t, vagy Rails 7-től a keretrendszer szinten elérhető titkosítást. Az utóbbihoz a

```
rails db:encryption:init
```

parancsot kell futtatnunk, majd a titkosítani kívánt attribútumot a modell fájlban meg kell jelölnünk:

```
class User < ApplicationRecord
  encrypts :password
end
```

A Rails működési logikájával való ismerkedés végett, most ennél bonyolultabb módon oldjuk meg a titkosítást, fapados eszközökkel összerakunk magunknak egyet. Ez először 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/models> rails g migration AddSaltToUsers salt
    :string
      invoke  active_record
      create    db/migrate/20220412102119_add_salt_to_users.rb
```

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

```
irb(main):001:0> SecureRandom.hex 8
=> "24c7cb21c8992103"
irb(main):002:0> SecureRandom.base64 8
=> "vtN3SoHbZq0="
irb(main):004:0> Digest::SHA1.hexdigest 'hello'
=> "aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d"
```

A Rails migrációi szinte kivétel nélkül invertálhatóak, vagyis a Rails képes fel és le irányban végrehajtani azokat akkor is, ha csak a fel irány definiáljuk a `change` metódusban. Ha olyan műveletet hajtanánk végre, amely mégsem az, akkor vagy a `change` metódusban használjuk a `reversible` függvényt, vagy a `change` helyett két függvényt definiálunk `up`, illetve `down` néven. Mivel az adatbázisban már van adat, fel irányú migráció esetén gondoskodunk kell azok új attribútumainak inicializálásáról.

```
class AddSaltToUsers < ActiveRecord::Migration[7.0]
  def change
    add_column :users, :salt, :string
    rename_column :users, :password, :encrypted_password
  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
```

Hajtsuk végre a migrációt érvényre juttatandó az adatmodellünk változásait!

```
kovacsg@debian:~/gyakorlat/db/migrate> rails db:migrate
== 20220412102119 AddSaltToUsers: migrating ===================================
-- add_column(:users, :salt, :string)
   -> 0.0072s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0041s
== 20220412102119 AddSaltToUsers: migrated (0.0120s) ==========================
```

Nézzük meg a migráció eredményét. Az adatbázisban már jelen lévő rekorddal nem tudunk most 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.

```
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    |
    |
| idnum              | varchar(255) | YES  |     | NULL    |
    |
| birthdate          | datetime(6)  | YES  |     | NULL    |
    |
| account            | varchar(255) | YES  |     | NULL    |
    |
| created_at         | datetime(6)  | NO   |     | NULL    |
    |
| updated_at         | datetime(6)  | NO   |     | NULL    |
    |
| salt               | varchar(255) | YES  |     | NULL    |
    |
+--------------------+--------------+------+-----+---------+----------+

10 rows in set (0.001 sec)
```

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 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. Az attribútum csak a memóriában él, az adatbázisba nem kerül ki az értéke.

```
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
  # encrypts :password
```

```
  before_save :encrypt_password #, on: :create

  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
```

Hozzunk létre egy felhasználót (5. sor), mentsük el (6. sor), és nézzük meg, hogy működik-e a jelszó titkosítása.

```
kovacsg@debian:~/gyakorlat> rails c
irb(main):005:0> u = User.new name: 'Senki', email: 'senki@mail.bme.hu',
    password: 'titok', birthdate: Date.today
=>
#<User:0x00005650c59552b0
...
irb(main):006:0> u.save
  TRANSACTION (0.2ms)  BEGIN
  User Exists? (0.4ms)  SELECT 1 AS one FROM `users` WHERE `users`.`email` =
       'senki@mail.bme.hu' LIMIT 1
  User Create (0.4ms)  INSERT INTO `users` (`name`, `email`, `
      encrypted_password`, `idnum`, `birthdate`, `account`, `created_at`, `
      updated_at`, `salt`) VALUES ('Senki', 'senki@mail.bme.hu', '
      e5320e64617d65c2296b535e8464611d91152fac', NULL, '2022-04-14 00:00:00'
      , NULL, '2022-04-14 11:44:37.756824', '2022-04-14 11:44:37.756824', '
      XCQOSARTqEDUicyxKCbpCw==')
  TRANSACTION (1.0ms)  COMMIT
=> true
```

Nézzük meg, hogy milyen elemi műveletekkel férhetünk egy vagy több adatbázisbeli rekordhoz hozzá Rails-ből. A `find` (21. sor) az `id` attribútum értéke alapján keres, és egy rekordhoz tartozó Ruby objektumot ad vissza. A `first` (7. sor), `last` (8. sor) és `take` (9. 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, a visszatérési érték `ActiveRecord::Base` objektumról tömbre változik. Egy attribútum alapján egy rekordot a `find_by` metódussal kereshetünk (10.sor). Az általános megoldás a keresésre a `where` (11. sor), amely esetén a visszatérési érték `ActiveRecord::Base` objektumról `ActiveRecord::Relation` objektumra változik, ami tömbként működik, és kaszkádosan végrehajthatók rá a fenti keresési műveletek, például a `take`, amellyel egy találatot kivehetünk abból. Az `all` függvény az modell összes rekordját visszaadja egy tömbben.

```
kovacsg@debian:~/gyakorlat> rails c
irb(main):007:0> User.first
```

```
 name: "Senki",
 encrypted_password: "[FILTERED]",
  User Load (0.3ms)  SELECT `users`.* FROM `users` LIMIT 1
#<User:0x00007f8b203ad6a0
 name: "Valaki",
 email: "valaki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 idnum: nil,
 birthdate: nil,
 account: "11111111-11111111",
 created_at: Tue, 12 Apr 2022 10:38:17.100987000 UTC +00:00,
 updated_at: Tue, 12 Apr 2022 10:38:17.100987000 UTC +00:00,
 salt: "[FILTERED]">
irb(main):008:0> User.last
 name: "Senki",
 encrypted_password: "[FILTERED]",
  User Load (0.3ms)  SELECT `users`.* FROM `users` LIMIT 1
#<User:0x00007f8b203ad6a0
 name: "Valaki",
 email: "valaki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 idnum: nil,
 birthdate: nil,
 account: "11111111-11111111",
 created_at: Tue, 12 Apr 2022 10:38:17.100987000 UTC +00:00,
 updated_at: Tue, 12 Apr 2022 10:38:17.100987000 UTC +00:00,
 salt: "[FILTERED]">
irb(main):010:0> User.find_by email: 'senki@mail.bme.hu'
  User Load (0.4ms)  SELECT `users`.* FROM `users` WHERE `users`.`email` = '
      senki@mail.bme.hu' LIMIT 1
=>
#<User:0x00005650c5b6c1e8
 id: 4,
 name: "Senki",
 email: "senki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 idnum: nil,
 birthdate: Thu, 14 Apr 2022 00:00:00.000000000 UTC +00:00,
 account: nil,
 created_at: Thu, 14 Apr 2022 11:44:37.756824000 UTC +00:00,
 updated_at: Thu, 14 Apr 2022 11:44:37.756824000 UTC +00:00,
 salt: "[FILTERED]">
irb(main):011:0> User.where(email: 'senki@mail.bme.hu').take
  User Load (4.6ms)  SELECT `users`.* FROM `users` WHERE `users`.`email` = '
      senki@mail.bme.hu' LIMIT 1
=>
#<User:0x00005650c5d56a30
 id: 4,
 name: "Senki",
 email: "senki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 idnum: nil,
 birthdate: Thu, 14 Apr 2022 00:00:00.000000000 UTC +00:00,
 account: nil,
 created_at: Thu, 14 Apr 2022 11:44:37.756824000 UTC +00:00,
 updated_at: Thu, 14 Apr 2022 11:44:37.756824000 UTC +00:00,
 salt: "[FILTERED]">
irb(main):012:0> User.find 4
  User Load (0.4ms)  SELECT `users`.* FROM `users` WHERE `users`.`id` = 4
      LIMIT 1
=>
#<User:0x00005650c5e24e30
```

```
id: 4,
name: "Senki",
email: "senki@mail.bme.hu",
encrypted_password: "[FILTERED]",
idnum: nil,
birthdate: Thu, 14 Apr 2022 00:00:00.000000000 UTC +00:00,
account: nil,
created_at: Thu, 14 Apr 2022 11:44:37.756824000 UTC +00:00,
updated_at: Thu, 14 Apr 2022 11:44:37.756824000 UTC +00:00,
salt: "[FILTERED]">
```

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/models> 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
```

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 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', as: 'logout', via: [:get
      , :delete]
  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, pass)
    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. 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.

```
<p><%= flash[:notice] %></p>
```

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

```
      redirect_back fallback_location: hello_path
    end
  end


  def edit
  end

  def update
    if @user.update(user_params)
      flash[:notice] = 'Successful update'
      redirect_back fallback_location: hello_path
    else
      flash[:notice] = 'Could not update profile'
      redirect_back fallback_location: hello_path
    end
  end

  private
    def user_params
      params.require(:user).permit([:name, :email, :passsword, :
          password_confirmation, :account])
    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] if 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: 'registration'
  post 'users/create', to: 'users#create', as: 'register'
  get 'users/forgotten', to: 'users#forgotten', as: 'forgotten'
  post 'users/send_forgotten', to: 'user#send_forgetten', as: '
      send_forgotten'
  get 'users/edit/:id', to: 'users#edit', as: 'profile'
  put 'users/update/:id', to: 'users#update', as: 'edit_profile'
```

Eztán mind a vendégfelhasználó, mind a bejelentkezett felhasználó menüjében módosítjuk a linkeket.

```
<p>Hello, Valaki!</p>
<p><%= flash[:notice] %></p>

<p><%= link_to "Profile", profile_path %></p>
<p><%= link_to "Parties", parties_path %></p>
<p><%= link_to "Candidates", candidates_path %></p>
<p><%= link_to "Vote", candidates_path %></p>
<p><%= link_to "Logout", logout_path %></p>
```

```
Hello, Guest!
<p><%= flash[:notice] %></p>

<fieldset>
  <legend>Login</legend>
  <%= form_tag login_path, method: :post do %>
    <%= label_tag 'email', 'Email' %>
    <%= text_field_tag 'email', '', size: 14 %>
    <%= label_tag 'password', 'Password' %>
    <%= password_field_tag 'password', '', size: 14 %>
    <%= submit_tag 'Login' %>
  <% end %>
</fieldset>

<%= link_to "Register", '/users/new' %><br/>
<%= link_to "Forgotten_password", '/users/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.

```
irb(main):013:0> u = User.new email: 'valaki@mail.bme.hu'
=>
#<User:0x00005650c5b3fe90
...
irb(main):014:0> u.save
  TRANSACTION (0.1ms)  BEGIN
  User Exists? (0.2ms)  SELECT 1 AS one FROM `users` WHERE `users`.`email` =
       'valaki@mail.bme.hu' LIMIT 1
  TRANSACTION (0.1ms)  ROLLBACK
=> false
irb(main):015:0> u.errors.messages
=> {:name=>["can't be blank"], :email=>["Email already in use"]}
```

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

```
<h1>Registration</h1>

<% if flash[:errors] && flash[:errors].any? %>
  <div id="error_explanation">
    <h2><%= pluralize(flash[:errors].count, 'error') %>prohibited this user
        from being saved</h2>
    <ul>
      <% flash[:errors].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 (`config/locales/en.yml`):

```
en:
  activerecord:
    errors:
      models:
        user:
          attributes:
            email:
              blank: "Cannot be empty"
              taken: "Email already in use"
```

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 választási portálunk adatmodellje a következő elemekből áll:

- Felhasználó, szavazópolgár (`User`)
- Párt (`Party`)
- Jelölt (`Candidate`)

Ezeket a korábbi gyakorlatokon már létrehoztuk.

Egy jelölt (`Candidate`) egy párthoz tartozik. A `references` típus a migrációban automatikusan létrehozta a `belongs_to` deklarációt a jelölt oldalon oldalon. A reláció fordított irányú navigációjának engedélyezése végett módosítjuk a párt modell osztályát.

```
class Party < ApplicationRecord
  has_many :candidates
end
```

Egy párt egy pártlistához tartozik, ezért hozzuk létre ezt az új modellt. A pártlistának egyelőre egy neve van csak, ezt később kibővíthetjük.

```
kovacsg@debian:~/gyakorlat/app/models> rails g model list name:string
      invoke  active_record
      create    db/migrate/20220412112448_create_lists.rb
      create    app/models/list.rb
      invoke    test_unit
      create      test/models/list_test.rb
      create      test/fixtures/lists.yml
kovacsg@debian:~/gyakorlat/app/models> rails db:migrate
== 20220412112448 CreateLists: migrating ======================================
-- create_table(:lists)
   -> 0.0128s
```

```
== 20220412112448 CreateLists: migrated (0.0131s)
   ==========================
```

Egy párt egy pártlistához tartozik, ezért fel kell vennünk egy idegen kulcsot a párt modellbe.

```
kovacsg@debian:~/gyakorlat/app/models> rails g migration AddListToParties
    list:references
      invoke  active_record
      create    db/migrate/20220412112535_add_list_to_parties.rb
```

Az automatikusan generált migráció forráskódja a null érték felvételét kizárja. Azt a feltételt távolítsuk el az opciók közül.

```
class AddListToParties < ActiveRecord::Migration[7.0]
  def change
    add_reference :parties, :list, foreign_key: true
  end
end
```

Hajtsuk végre a migrációt.

```
kovacsg@debian:~/gyakorlat/db/migrate> rails db:migrate
== 20220412112535 AddListToParties: migrating
   ================================
-- add_reference(:parties, :list, {:foreign_key=>true})
   -> 0.0287s
== 20220412112535 AddListToParties: migrated (0.0291s)
   ===========================
```

A felhasználókat lakcímül alapján szavazókörökbe soroljuk. A szavazóköröknek is van címük. Mivel nem akarunk kétféle cím modellt definiálni, ezt polimorfizmussal oldjuk meg. A cím modell álljon a következő attribútumokból: egy string típusú város, egy string típusú utca és házszám, egy egész típusú irányítószám, egy-egy lebegőpontos típusú szélesség és hosszúság koordináta, és egy polimorfikus idegen kulcs. A szavazókörnek legyen neve, ami string típusú.

```
kovacsg@debian:~/gyakorlat> rails g model Address city:string street:string
    zip:integer lat:float lng:float addr:references
      invoke  active_record
      create    db/migrate/20220412113757_create_addresses.rb
      create    app/models/address.rb
      invoke    test_unit
      create      test/models/address_test.rb
      create      test/fixtures/addresses.yml
kovacsg@debian:~/gyakorlat/db/migrate> rails g model district name:string
      invoke  active_record
      create    db/migrate/20220412113906_create_districts.rb
      create    app/models/district.rb
      invoke    test_unit
      create      test/models/district_test.rb
      create      test/fixtures/districts.yml
```

A cím migrációs fájljában az idegen kulcs attribútumnál opcióként megadjuk, hogy az polimorfikus.

```
class CreateAddresses < ActiveRecord::Migration[7.0]
  def change
    create_table :addresses do |t|
      t.string :city
      t.string :street
      t.integer :zip
      t.float :lat
      t.float :lng
      t.references :addr, polymorphic: true

      t.timestamps
    end
  end
  end
```

Hajtsuk végre a migrációt!

```
kovacsg@debian:~/gyakorlat/db/migrate> rails db:migrate
== 20220412113757 CreateAddresses: migrating
   ==========================================
-- create_table(:addresses)
   -> 0.0088s
== 20220412113757 CreateAddresses: migrated (0.0092s)
   =================================

== 20220412113906 CreateDistricts: migrating
   ==========================================
-- create_table(:districts)
   -> 0.0075s
== 20220412113906 CreateDistricts: migrated (0.0078s)
   =================================
```

A felhasználó és a választókörzet címmel rendelkezik, és a cím ezek valamelyikékez tartozik. Ezt a modell osztályokban is meg kell jelenítenünk. A cím modell osztályban a `belongs_to` függvénynek egy opcióban jelenzünk kell, hogy itt polimorfikus kapcsolatról van szó. A felhasználó és a választókörzet modell pedig a cím polimorfikus idegen kulcsára hivatkozik, a tulajdonság neve `address`, viszont az idegen kulcsot `addr` néven vettük fel, ezért ezt meg kell jelenítenünk.

```
class Address < ApplicationRecord
  belongs_to :addr, polymorphic: true
  end
class User < ApplicationRecord
  has_one :address, as: :addr
end
class District < ApplicationRecord
  has_one :address, as: :addr
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 leglas-

sabb 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 tageket.

```
u = User.create username: 'Valaki', email: 'valaki@mail.bme.hu', birthdate:
    Date.today, password: 'titok'
```

Töltsük be ezeket az adatokat:

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