Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2020. április 7.

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 két modellt hoztunk létre, a felhasználók `User` nevű modelljét és a feladatok `Task` nevű modelljét. A felhasználók modellje a következőképp néz ki az adatbázisban.

```
MariaDB [gyakorlat_development]> show tables;
+-----------------------------------+
| Tables_in_gyakorlat_development |
+-----------------------------------+
| ar_internal_metadata              |
| schema_migrations                 |
| tasks                             |
| users                             |
+-----------------------------------+
4 rows in set (0.000 sec)

MariaDB [gyakorlat_development]> desc users;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| name       | varchar(40)  | YES  |     | NULL    |                |
| neptun     | varchar(6)   | YES  |     | NULL    |                |
| email      | varchar(255) | YES  |     | NULL    |                |
| password   | varchar(20)  | YES  |     | NULL    |                |
| created_at | datetime(6)  | NO   |     | NULL    |                |
| updated_at | datetime(6)  | NO   |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
7 rows in set (0.001 sec)
```

Először növeljük portálunk biztonságát azzal, hogy a jelszavakat nem szövegesen, hanem titkosítva tároljuk. Ez a jelszó mező átnevezéséből és egy új,

a titkosítás során a felhasználó számára egyedi attribútum felvételéből áll. A Rails az új attribútum felvételéhez képes automatikusan invertálható migrációt generálni jól definiált migrációnév esetén. A hozzáadott attribútumnak ilyen esetben a migráció neve után kell szerepelnie. Azonban az attribútum átnevezését magunknak kell majd hozzáadnunk a migrációhoz.

```
kovacsg@debian:~/gyakorlat/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, visszafejthetetlen és egyben olvashatatlan karaktersorozatokra, és stringből olvashatatlan, visszafejthetetlen karaktersorozatot előállító függvényre.

```
kovacsg@debian:~/gyakorlat> rails c
Loading development environment (Rails 6.0.2.1)
irb(main):001:0> SecureRandom
=> SecureRandom
irb(main):002:0> SecureRandom.hex 8
=> "997ce8188acee447"
irb(main):003:0> SecureRandom.hex 16
=> "df70200fb3e7fbd7c0cc23d703ec3d08"
irb(main):004:0> SecureRandom.base64 16
=> "ILyi2bK+ehWwdU/LbcE3Bw=="
irb(main):005:0> Digest::SHA1.hexdigest 'titok'
=> "46ff53e764c4acf97b54db2020573049d2e3dab3"
```

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
== 20200403095521 AddSaltToUsers: migrating
   ==========================================
-- add_column(:users, :salt, :string)
   -> 0.0272s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0352s
== 20200403095521 AddSaltToUsers: migrated (0.0632s)
   =================================
```

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(40)  | YES  |     | NULL    |                |
| neptun             | varchar(6)   | YES  |     | NULL    |                |
| email              | varchar(255) | YES  |     | NULL    |                |
| encrypted_password | varchar(20)  | YES  |     | NULL    |                |
| created_at         | datetime(6)  | NO   |     | NULL    |                |
| updated_at         | datetime(6)  | NO   |     | NULL    |                |
| salt               | varchar(255) | YES  |     | NULL    |                |
+--------------------+--------------+------+-----+---------+----------------+
8 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
  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
```

Hozzunk létre egy felhasználót (1. sor), mentsük el (3. sor), és nézzük meg, hogy működik-e a jelszó titkosítása.

```
kovacsg@debian:~/gyakorlat> rails c
Loading development environment (Rails 6.0.2.1)
irb(main):001:0> u = User.new name:'valaki', email: 'valaki@mail.bme.hu',
    neptun: 'aaaaaa', password: 'titok'
   (0.4ms)  SET NAMES utf8mb4,  @@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: "valaki", neptun: "aaaaaa", email: "valaki@mail.bme
    .hu", encrypted_password: nil, created_at: nil, updated_at: nil, salt:
    nil>
irb(main):003:0> u.save
   (0.2ms)  BEGIN
  User Create (1.2ms)  INSERT INTO `users` (`name`, `neptun`, `email`, `
      encrypted_password`, `created_at`, `updated_at`, `salt`) VALUES ('
```

```
      valaki', 'aaaaaa', 'valaki@mail.bme.hu', '
      bc304359fdff974160c6fbd9230c9c0b089f960e', '2020-04-03 10:21:32.393553
      ', '2020-04-03 10:21:32.393553', 'kduaB2FW8Y3T4Q14VFUDXw==')
   (0.1ms)  ROLLBACK
Traceback (most recent call last):
        1: from (irb):3
ActiveRecord::ValueTooLong (Mysql2::Error: Data too long for column '
    encrypted_password' at row 1)
```

Elsőre nem sikerült, mert 20 karakteres korlátot adtunk meg eredetileg az azóta már átnevezett `password` attribútumak, és az kevés a `hexdigest` kimenetének. Hozzunk létre egy új migrációt, amelyben ezt módosítjuk.

```
kovacsg@debian:~/gyakorlat/db/migrate> rails g migration
    ChangeEncrpytedPasswordLengthInUsers
      invoke  active_record
      create    db/migrate/20200403102224
          _change_encrpyted_password_length_in_users.rb
```

Módosítsuk a titkosított attribútum hosszát. A művelet invertálható, ezért nem kell kettéválasztanunk a `change` metódust.

```
class ChangeEncrpytedPasswordLengthInUsers < ActiveRecord::Migration[6.0]
  def change
    change_column :users, :encrypted_password, :string, limit: 50
  end
end
```

Hozzuk létre ugyanazt a felhasználót (1. sor), mentsük el (2. sor), és nézzük meg, hogy működik-e a jelszó titkosítása.

```
kovacsg@debian:~/gyakorlat> rails c
Loading development environment (Rails 6.0.2.1)
irb(main):001:0> u = User.new name:'valaki', email: 'valaki@mail.bme.hu',
    neptun: 'aaaaaa', password: 'titok'
   (0.3ms)  SET NAMES utf8mb4,  @@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: "valaki", neptun: "aaaaaa", email: "valaki@mail.bme
    .hu", encrypted_password: nil, created_at: nil, updated_at: nil, salt:
    nil>
irb(main):002:0> u.save
   (0.2ms)  BEGIN
  User Create (0.4ms)  INSERT INTO `users` (`name`, `neptun`, `email`, `
      encrypted_password`, `created_at`, `updated_at`, `salt`) VALUES ('
      valaki', 'aaaaaa', 'valaki@mail.bme.hu', '11
      e85f1b108155fad1c5db4e4f7fd61d60252e2d', '2020-04-03 10:25:07.030354',
       '2020-04-03 10:25:07.030354', 'AoAIFj/36rjDFbNi7tFztw==')
   (1.4ms)  COMMIT
=> true
```

Nézzük meg, hogy milyen elemi műveletekkel férhetünk egy vagy több adatbázisbeli rekordhoz hozzá Rails-ből. A `first` (4. sor), `last` (3. sor) és `take` (5. 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 (6. sor), a visszatérési érték `ActiveRecord::Base` objektumról `ActiveRecord::Relation`

objektumra változik, ami tömbként működik, és kaszkádosan végrehajthatók rá a keresési műveletek. Az `all` (8.sor) függvény az modell összes rekordját visszaadja egy tömbben.

```
kovacsg@debian:~/gyakorlat> rails c
Loading development environment (Rails 6.0.2.1)
irb(main):001:0> User.find_by email:'valaki@mail.bme.hu'
   (0.3ms)  SET NAMES utf8mb4,  @@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` = '
      valaki@mail.bme.hu' LIMIT 1
=> #<User id: 2, name: "valaki", neptun: "aaaaaa", email: "valaki@mail.bme.
    hu", encrypted_password: [FILTERED], created_at: "2020-04-03 10:25:07",
    updated_at: "2020-04-03 10:25:07", salt: "AoAIFj/36rjDFbNi7tFztw==">
irb(main):002:0> User.where email:'valaki@mail.bme.hu'
  User Load (1.0ms)  SELECT `users`.* FROM `users` WHERE `users`.`email` = '
      valaki@mail.bme.hu' LIMIT 11
=> #<ActiveRecord::Relation [#<User id: 2, name: "valaki", neptun: "aaaaaa",
     email: "valaki@mail.bme.hu", encrypted_password: [FILTERED], created_at
    : "2020-04-03 10:25:07", updated_at: "2020-04-03 10:25:07", salt: "
    AoAIFj/36rjDFbNi7tFztw==">]>
irb(main):003:0> User.last
  User Load (1.1ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id`
      DESC LIMIT 1
=> #<User id: 2, name: "valaki", neptun: "aaaaaa", email: "valaki@mail.bme.
    hu", encrypted_password: [FILTERED], created_at: "2020-04-03 10:25:07",
    updated_at: "2020-04-03 10:25:07", salt: "AoAIFj/36rjDFbNi7tFztw==">
irb(main):004:0> User.first
  User Load (0.8ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC
       LIMIT 1
=> #<User id: 2, name: "valaki", neptun: "aaaaaa", email: "valaki@mail.bme.
    hu", encrypted_password: [FILTERED], created_at: "2020-04-03 10:25:07",
    updated_at: "2020-04-03 10:25:07", salt: "AoAIFj/36rjDFbNi7tFztw==">
irb(main):005:0> User.take
  User Load (1.0ms)  SELECT `users`.* FROM `users` LIMIT 1
=> #<User id: 2, name: "valaki", neptun: "aaaaaa", email: "valaki@mail.bme.
    hu", encrypted_password: [FILTERED], created_at: "2020-04-03 10:25:07",
    updated_at: "2020-04-03 10:25:07", salt: "AoAIFj/36rjDFbNi7tFztw==">
irb(main):006:0> User.take(2)
  User Load (0.5ms)  SELECT `users`.* FROM `users` LIMIT 2
=> [#<User id: 2, name: "valaki", neptun: "aaaaaa", email: "valaki@mail.bme.
    hu", encrypted_password: [FILTERED], created_at: "2020-04-03 10:25:07",
    updated_at: "2020-04-03 10:25:07", salt: "AoAIFj/36rjDFbNi7tFztw==">]
irb(main):007:0> User.where(email:'valaki@mail.bme.hu').take
  User Load (0.7ms)  SELECT `users`.* FROM `users` WHERE `users`.`email` = '
      valaki@mail.bme.hu' LIMIT 1
=> #<User id: 2, name: "valaki", neptun: "aaaaaa", email: "valaki@mail.bme.
    hu", encrypted_password: [FILTERED], created_at: "2020-04-03 10:25:07",
    updated_at: "2020-04-03 10:25:07", salt: "AoAIFj/36rjDFbNi7tFztw==">
irb(main):008:0> User.all
  User Load (0.7ms)  SELECT `users`.* FROM `users` LIMIT 11
=> #<ActiveRecord::Relation [#<User id: 2, name: "valaki", neptun: "aaaaaa",
     email: "valaki@mail.bme.hu", encrypted_password: [FILTERED], created_at
    : "2020-04-03 10:25:07", updated_at: "2020-04-03 10:25:07", salt: "
    AoAIFj/36rjDFbNi7tFztw==">]>
```

Elsődleges kulcs alapján, amely jellemzően a weboldalon megjelenő azonosító önmaga, a `find` (9. sor) metódussal kereshetünk egy objektumot. Egy objektum törlése is történhet az azonosító alapján. Valamely attribútum ér-

téke 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):009:0> User.find 2
  User Load (0.9ms)  SELECT `users`.* FROM `users` WHERE `users`.`id` = 2
      LIMIT 1
=> #<User id: 2, name: "valaki", neptun: "aaaaaa", email: "valaki@mail.bme.
    hu", encrypted_password: [FILTERED], created_at: "2020-04-03 10:25:07",
    updated_at: "2020-04-03 10:25:07", salt: "AoAIFj/36rjDFbNi7tFztw==">
irb(main):010:0> User.delete 1
  User Destroy (0.4ms)  DELETE FROM `users` WHERE `users`.`id` = 1
=> 0
```

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> 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    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', as: 'logout', via: [:get
      , :delete]
  match 'say/hello', to: 'say#hello', via: [:get], 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.

```
<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 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_user' # 'users'
  get 'users/edit/:id', to: 'users#edit', as: 'edit_profile' # 'users/:id/
      edit'
  put 'users/update/:id', to: 'users#update', as: 'update_profile' # 'users
      /:id'
  get 'users/index', to: 'users#index', as: 'users' # 'users'
  get 'users/show/:id', to: 'users#show', as: 'profile' # 'users/:id'
  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 6.0.2.1)
irb(main):001:0> u = User.new
   (0.4ms)  SET NAMES utf8mb4,  @@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, neptun: nil, email: nil, encrypted_password:
    nil, created_at: nil, updated_at: nil, salt: nil>
irb(main):002:0> u.save
   (0.2ms)  BEGIN
  User Exists? (0.3ms)  SELECT 1 AS one FROM `users` WHERE `users`.`email`
      IS NULL LIMIT 1
   (0.3ms)  ROLLBACK
=> false
irb(main):003:0> u.errors.messages
=> {:name=>["can't be blank"], :email=>["can't be blank"], :neptun=>["can't
    be blank"]}
irb(main):004:0> u.name = 'valaki'
=> "valaki"
irb(main):005:0> u.email = 'valaki@mail.bme.hu'
=> "valaki@mail.bme.hu"
irb(main):006:0> u.neptun = 'bbbbbb'
=> "bbbbbb"
irb(main):007:0> u.valid?
DEPRECATION WARNING: Uniqueness validator will no longer enforce case
    sensitive comparison in Rails 6.1. To continue case sensitive comparison
     on the :email attribute in User model, pass `case_sensitive: true`
    option explicitly to the uniqueness validator. (called from irb_binding
    at (irb):7)
  User Exists? (2.1ms)  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"]}
```

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
- Feladat (`Task`)
- Megoldás (`Solution`)

Ezeket a korábbi gyakorlatokon már létrehoztuk.

Egy megoldás (`Solution`) a létrehozó felhasználó tulajdonában áll, és egy feladathoz tartozik. A `references` típus a migrációban automatikusan létrehozza a `belongs_to` deklarációt. Mindemellett egy megoldáshoz a későbbiekben tartozni fog egy csatolmány is.

```
kovacsg@debian:~/gyakorlat/app/models> rails g model Solution user:
    references task:references
      invoke  active_record
      create    db/migrate/20200403105841_create_solutions.rb
      create    app/models/solution.rb
      invoke    test_unit
      create      test/models/solution_test.rb
      create      test/fixtures/solutions.yml
kovacsg@debian:~/gyakorlat/db> rails db:migrate
== 20200403105841 CreateSolutions: migrating
   ==========================================
-- create_table(:solutions)
   -> 0.0347s
== 20200403105841 CreateSolutions: migrated (0.0351s)
   =================================
```

A létrejött modell osztály egyelőre így jó.

```
class Solution < ApplicationRecord
  belongs_to :user
  belongs_to :task
end
```

A felhasználók modelljét kiegészítjük a kapcsolataival. Egy egy-több kapcsolattal a megoldások felé, és egy, a megoldásokon keresztül indirekt több-több kapcsolatra a feladatok felé.

```
class User < ApplicationRecord
  has_many :solutions
  has_many :tasks, through: :solutions
end
```

A feladatok modelljével hasonlóan járunk el. Ez is egy egy-több kapcsolattal bír a megoldások felé, és egy, a megoldásokon kereszül indirekt több-több kapcsolattal a felhasználók felé.

```
class Task < ApplicationRecord
  has_many :solutions
  has_many :users, through: :solutions
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.

```
User.create name:'admin', email: 'admin@mail.bme.hu', neptun: 'ADMIN0',
    password: 'titok'
for i in 1..6 do
  Task.create title: "Task#{i}", description: "Contents of task description
     #{i}", number: i
end
```

Töltsük be ezeket az adatokat:

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

Rails konzolon ellenőrizhetjük, hogy az adatok létrejöttek-e.

```
kovacsg@debian:~/gyakorlat> rails c
Loading development environment (Rails 6.0.2.1)
irb(main):001:0> User.last
   (0.3ms)  SET NAMES utf8mb4,  @@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` ORDER BY `users`.`id`
      DESC LIMIT 1
```

```
=> #<User id: 3, name: "admin", neptun: "ADMIN0", email: "admin@mail.bme.hu
     ", encrypted_password: [FILTERED], created_at: "2020-04-03 11:12:36",
     updated_at: "2020-04-03 11:12:36", salt: "0s5I396XxC6cMWhMEu1tCg==">
irb(main):002:0> Task.last
  Task Load (0.4ms)  SELECT `tasks`.* FROM `tasks` ORDER BY `tasks`.`id`
     DESC LIMIT 1
=> #<Task id: 7, number: 6, title: "Task6", description: "Contents of task
     description 6", created_at: "2020-04-03 11:12:36", updated_at:
     "2020-04-03 11:12:36">
```