Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2023. október 31.

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                 |
| submissions                       |
| 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(255) | YES  |     | NULL    |                |
| email      | varchar(255) | YES  |     | NULL    |                |
| neptun     | varchar(255) | YES  |     | NULL    |                |
| password   | varchar(255) | 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/config> rails g migration AddSaltToUsers salt:
    string
      invoke  active_record
      create    db/migrate/20231031111940_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/db/migrate> rails c
Loading development environment (Rails 7.0.8)
irb(main):001> SecureRandom
=> SecureRandom
irb(main):002> SecureRandom.hex 8
=> "b8f95828917c8053"
irb(main):003> SecureRandom.hex 16
=> "a70036f9f51ddc95925c8189afbfa397"
irb(main):004> SecureRandom.base64 16
=> "WgjGktfEUFvn2G2dSB7q/g=="
irb(main):005> Digest::SHA1.hexdigest 'titok'
=> "46ff53e764c4acf97b54db2020573049d2e3dab3"
irb(main):006>
```

Mivel az attribútum átnevezése invertálható, a `change` metódust használhatjuk, ellenkező esetben 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
#     reversible do |dir|
#       dir.up { ... }
#       dir.down { ... }
#     end
    add_column :users, :salt, :string
    rename_column :users, :password, :encrypted_password
  end

  def up
#     ...
  end

  def down
#     ...
  end
end
```

Hajtsuk végre a migrációt érvényre juttatandó az adatmodellünk változásait!

```
kovacsg@debian:~/gyakorlat/db/migrate> rails db:migrate
== 20231031111940 AddSaltToUsers: migrating ===================================
-- add_column(:users, :salt, :string)
   -> 0.0195s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0072s
== 20231031111940 AddSaltToUsers: migrated (0.0275s) ==========================
```

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]> select * from users;
+----+---------+-------------------+---------+--------------------+----------------------------+----------------------------+------+
| id | name    | email             | neptun  | encrypted_password |
    created_at                 | updated_at                 | salt |
+----+---------+-------------------+---------+--------------------+----------------------------+----------------------------+------+
|  1 | Valaki  | valaki@mail.bme.hu | aaaaaa  | titok              |
    2023-10-03 11:41:10.695120 | 2023-10-03 11:41:10.695120 | NULL |
+----+---------+-------------------+---------+--------------------+----------------------------+----------------------------+------+
1 row in set (0.000 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.

```
kovacsg@debian:~/gyakorlat/app/helpers> rails c
Loading development environment (Rails 7.0.8)
irb(main):001> u = User.new name: 'Senki', email: 'senki@mail.bme.hu',
    password: 'titok', n
eptun: 'senki0'
=>
#<User:0x00007fb0528abce0
...
irb(main):002> u.password
=> "titok"
irb(main):003> u.id
=> nil
irb(main):004> u.new_record?
=> true
irb(main):005> u.password.blank?
=> false
```

A migrációban minden egyes felhasználói rekordhoz hozzáadtunk egy egyéni a jelszó titkosításához használt random kulcs tárololására használt att-

ribútumot 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 User.encrypt(pass, salt)
        Digest::SHA1.hexdigest pass+salt
    end

    def encrypt_password
        return if self.password.blank?
        if self.new_record?
            self.salt = SecureRandom.base64 16
        end
        self.encrypted_password = User.encrypt(self.password, self.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/app/helpers> rails c
Loading development environment (Rails 7.0.8)
irb(main):001> u = User.new name: 'Senki', email: 'senki@mail.bme.hu',
    password: 'titok', neptun: 'senki0'
=>
#<User:0x00007fea4c422a28
...
irb(main):002> u.save
  TRANSACTION (0.3ms)  BEGIN
  User Create (3.2ms)  INSERT INTO `users` (`name`, `email`, `neptun`, `
      encrypted_password`, `created_at`, `updated_at`, `salt`) VALUES ('
```

```
      Senki', 'senki@mail.bme.hu', 'senki0', '74
      c6c5705eff7ed1478cc28dfbd04f01d2160fa6', '2023-10-31 11:33:23.254829',
       '2023-10-31 11:33:23.254829', 'OoBePyeW7/2IHSPOVd8WCg==')
  TRANSACTION (0.9ms)  COMMIT
=> true
```

Nézzük meg, hogy milyen elemi műveletekkel férhetünk egy vagy több adatbázisbeli rekordhoz hozzá Rails-ből. A `first` (3. sor), `last` (5. sor) és `take` (6. sor) rendre az elsődleges kulcs szerinti első, utolsó, és az alapértelmezett rendezés szerinti első rekordot adják vissza. Ezeknek egész paramétert adva a rendezés után visszaadott rekordok számát adhatjuk meg (7. sor), a visszatérési érték `ActiveRecord::Base` objektumról `ActiveRecord::Relation` objektumra változik, ami tömbként működik, és kaszkádosan végrehajthatók rá a keresési műveletek. Az `all` (14.sor) függvény az modell összes rekordját visszaadja egy tömbben.

```
kovacsg@debian:~/gyakorlat/app/helpers$ rails c
Loading development environment (Rails 7.0.8)
irb(main):001> u = User.find 2
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`id` = 2
      LIMIT 1
=>
#<User:0x00007f413d5e8eb0
...
irb(main):002> u.name
=> "Senki"
irb(main):003> u = User.first
  User Load (0.6ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC
       LIMIT 1
=>
#<User:0x00007f413c6cd4a8
...
irb(main):004> u.name
=> "Valaki"
irb(main):005> User.last
  User Load (0.5ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id`
      DESC LIMIT 1
=>
#<User:0x00007f413d0a0a48
 id: 2,
 name: "Senki",
 email: "senki@mail.bme.hu",
 neptun: "senki0",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 31 Oct 2023 11:33:23.254829000 UTC +00:00,
 updated_at: Tue, 31 Oct 2023 11:33:23.254829000 UTC +00:00,
 salt: "[FILTERED]">
irb(main):006> User.take
  User Load (0.3ms)  SELECT `users`.* FROM `users` LIMIT 1
=>
#<User:0x00007f413cbc42e0
 id: 1,
 name: "Valaki",
 email: "valaki@mail.bme.hu",
 neptun: "aaaaaa",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 03 Oct 2023 11:41:10.695120000 UTC +00:00,
 updated_at: Tue, 03 Oct 2023 11:41:10.695120000 UTC +00:00,
```

```
  salt: nil>
irb(main):007> User.take(2)
  User Load (0.5ms)  SELECT `users`.* FROM `users` LIMIT 2
=>
[#<User:0x00007f413cac77c0
  id: 1,
  name: "Valaki",
  email: "valaki@mail.bme.hu",
  neptun: "aaaaaa",
  encrypted_password: "[FILTERED]",
  created_at: Tue, 03 Oct 2023 11:41:10.695120000 UTC +00:00,
  updated_at: Tue, 03 Oct 2023 11:41:10.695120000 UTC +00:00,
  salt: nil>,
 #<User:0x00007f413cac76f8
  id: 2,
  name: "Senki",
  email: "senki@mail.bme.hu",
  neptun: "senki0",
  encrypted_password: "[FILTERED]",
  created_at: Tue, 31 Oct 2023 11:33:23.254829000 UTC +00:00,
  updated_at: Tue, 31 Oct 2023 11:33:23.254829000 UTC +00:00,
  salt: "[FILTERED]">]
irb(main):008> User.find_by name: "Valaki"
  User Load (0.5ms)  SELECT `users`.* FROM `users` WHERE `users`.`name` = '
      Valaki' LIMIT 1
=>
#<User:0x00007f413d027440
 id: 1,
 name: "Valaki",
 email: "valaki@mail.bme.hu",
 neptun: "aaaaaa",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 03 Oct 2023 11:41:10.695120000 UTC +00:00,
 updated_at: Tue, 03 Oct 2023 11:41:10.695120000 UTC +00:00,
 salt: nil>
irb(main):009> User.where(name: "Valaki")
  User Load (0.6ms)  SELECT `users`.* FROM `users` WHERE `users`.`name` = '
      Valaki'
=>
[#<User:0x00007f413c63ec08
  id: 1,
  name: "Valaki",
  email: "valaki@mail.bme.hu",
  neptun: "aaaaaa",
  encrypted_password: "[FILTERED]",
  created_at: Tue, 03 Oct 2023 11:41:10.695120000 UTC +00:00,
  updated_at: Tue, 03 Oct 2023 11:41:10.695120000 UTC +00:00,
  salt: nil>]
irb(main):010> User.where(name: "Valaki").where(neptun:'aaaaaa')
  User Load (0.7ms)  SELECT `users`.* FROM `users` WHERE `users`.`name` = '
      Valaki' AND `users`.`neptun` = 'aaaaaa'
=>
[#<User:0x00007f413cb53018
  id: 1,
  name: "Valaki",
  email: "valaki@mail.bme.hu",
  neptun: "aaaaaa",
  encrypted_password: "[FILTERED]",
  created_at: Tue, 03 Oct 2023 11:41:10.695120000 UTC +00:00,
  updated_at: Tue, 03 Oct 2023 11:41:10.695120000 UTC +00:00,
  salt: nil>]
irb(main):011> User.where(name: "Valaki").where(neptun:'aaaaaa').take
```

```
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`name` = '
      Valaki' AND `users`.`neptun` = 'aaaaaa' LIMIT 1
=>
#<User:0x00007f413c5dcc88
 id: 1,
 name: "Valaki",
 email: "valaki@mail.bme.hu",
 neptun: "aaaaaa",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 03 Oct 2023 11:41:10.695120000 UTC +00:00,
 updated_at: Tue, 03 Oct 2023 11:41:10.695120000 UTC +00:00,
 salt: nil>
irb(main):012> User.where.not(name: "Valaki").take
  User Load (0.4ms)  SELECT `users`.* FROM `users` WHERE `users`.`name` != '
      Valaki' LIMIT 1
=>
#<User:0x00007f413c69ae90
 id: 2,
 name: "Senki",
 email: "senki@mail.bme.hu",
 neptun: "senki0",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 31 Oct 2023 11:33:23.254829000 UTC +00:00,
 updated_at: Tue, 31 Oct 2023 11:33:23.254829000 UTC +00:00,
 salt: "[FILTERED]">
irb(main):013> User.where("created_at >= ?", Date.new(2023,10,30)).take
  User Load (1.8ms)  SELECT `users`.* FROM `users` WHERE (created_at >= '
      2023-10-30') LIMIT 1
=>
#<User:0x00007f413cc46128
 id: 2,
 name: "Senki",
 email: "senki@mail.bme.hu",
 neptun: "senki0",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 31 Oct 2023 11:33:23.254829000 UTC +00:00,
 updated_at: Tue, 31 Oct 2023 11:33:23.254829000 UTC +00:00,
 salt: "[FILTERED]">
irb(main):014> User.all
  User Load (0.4ms)  SELECT `users`.* FROM `users`
=>
[#<User:0x00007f413caef540
  id: 1,
  name: "Valaki",
  email: "valaki@mail.bme.hu",
  neptun: "aaaaaa",
  encrypted_password: "[FILTERED]",
  created_at: Tue, 03 Oct 2023 11:41:10.695120000 UTC +00:00,
  updated_at: Tue, 03 Oct 2023 11:41:10.695120000 UTC +00:00,
  salt: nil>,
 #<User:0x00007f413caef428
  id: 2,
  name: "Senki",
  email: "senki@mail.bme.hu",
  neptun: "senki0",
  encrypted_password: "[FILTERED]",
  created_at: Tue, 31 Oct 2023 11:33:23.254829000 UTC +00:00,
  updated_at: Tue, 31 Oct 2023 11:33:23.254829000 UTC +00:00,
  salt: "[FILTERED]">]
```

Elsődleges kulcs alapján, amely jellemzően a weboldalon megjelenő azo-

nosító önmaga, a `find` (1. sor) 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 `find_by` (8.sor) helyett `where` keresőmetódus (előző kódblokk 9-13. 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.

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/helpers> 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, ki-

jelentkezéskor 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:  [:
      delete, :get]
  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 User.authenticate(email, pass)
        user = User.where(email: email).first
        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 email paraméter 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
    else
        flash[:notice] = "Invalid email or password"
    end
    redirect_back fallback_location: hello_path
  end

  def destroy
    reset_session
    flash[:notice] = "Logged out successfully"
    redirect_to 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"
        else
            flash[:notice] = "#{@user.errors.messages}"
        end
        redirect_back fallback_location: hello_path
    end

    def update
        if @user.update user_params
            flash[:notice] = "Update successful"
            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, :password, :
                password_confirmation, :neptun])
        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.

```
Rails.application.routes.draw do
  post 'users/create', to: 'users#create', as: 'register'
  get 'users/edit/:id', to: 'users#edit', as: 'edit_user'
  put 'users/update/:id', to: 'users#update', as: 'save_profile'
end
```

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/app/helpers> rails c
Loading development environment (Rails 7.0.8)
irb(main):001> u = User.new name: 'Senki', email: 'senki@mail.bme.hu',
    password: 'titok', n
eptun: 'senki0'
=>
#<User:0x00007f94c63a99b0
...
irb(main):002> u.save
  TRANSACTION (0.2ms)  BEGIN
```

```
  User Exists? (0.3ms)  SELECT 1 AS one FROM `users` WHERE `users`.`email` =
       'senki@mail.bme.hu' LIMIT 1
  TRANSACTION (0.1ms)  ROLLBACK
=> false
irb(main):003> u.valid?
  User Exists? (0.3ms)  SELECT 1 AS one FROM `users` WHERE `users`.`email` =
       'senki@mail.bme.hu' LIMIT 1
=> false
irb(main):004> u.errors.messages
=> {:email=>["has already been taken"]}
irb(main):005> u = User.new
=>
#<User:0x00007f94c5ff3708
...
irb(main):006> u.save
  TRANSACTION (0.2ms)  BEGIN
  User Exists? (0.3ms)  SELECT 1 AS one FROM `users` WHERE `users`.`email`
       IS NULL LIMIT 1
  TRANSACTION (0.1ms)  ROLLBACK
=> false
irb(main):007> u.errors.messages
=> {:name=>["can't be blank"], :email=>["can't be blank"]}
```

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 (`Submission`)

Ezeket a korábbi gyakorlatokon már létrehoztuk.

Egy megoldás (`Submission`) a létrehozó felhasználó tulajdonában áll, és egy feladathoz tartozik. A `references` típus a migrációban automatikusan létrehozta a `belongs_to` deklarációt. Mindemellett egy megoldáshoz a későbbiekben tartozni fog egy csatolmány is, amely részére múltkor definiáltunk egy példányváltozót, amely nem kerül mentésre.

A létrejött modell osztály egyelőre így jó.

```
class Submission < 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 :submissions
  has_many :tasks, through: : submissions
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 : submissions
  has_many :users, through: : submissions
end
```

Nézzük meg konzolon, hogy létrejöttek-e az objektumok közötti kapcsolatok. Vegyünk egy feladatot (9. sor) és egy felhasználót (11. sor). Ez a felhasználó adjon be egy megoldást erre a feladatra (12-13. sor). Nézzük meg a felhasználó megoldásait (14. sor), a feladatra született megoldásokat

(15. sor). Végül a `through` kapcsolatot ellenőrizendő nézzük meg, hogy a felhasználó mely feladatokra adott megoldást (17. sor), illetve a feladatra mely felhasználóktól érkezett megoldás (16. sor).

```
irb(main):009> t = Task.last
  Task Load (0.2ms)  SELECT `tasks`.* FROM `tasks` ORDER BY `tasks`.`id`
      DESC LIMIT 1
=>
#<Task:0x00007f94c5ab6880
...
irb(main):010> t.description
=> "Ez is nehez"
irb(main):011> u = User.last
  User Load (0.4ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id`
      DESC LIMIT 1
=>
#<User:0x00007f94c5f34f10
...
irb(main):012> s = Submission.new user: u, task: t
=>
#<Submission:0x00007f94c6aa20c8
...
irb(main):013> s.save
  TRANSACTION (0.2ms)  BEGIN
  Submission Create (3.5ms)  INSERT INTO `submissions` (`user_id`, `task_id
      `, `created_at`, `updated_at`) VALUES (3, 2, '2023-10-31
      12:37:39.966582', '2023-10-31 12:37:39.966582')
  TRANSACTION (0.6ms)  COMMIT
=> true
irb(main):014> u.submissions
  Submission Load (3.0ms)  SELECT `submissions`.* FROM `submissions` WHERE `
      submissions`.`user_id` = 3
=>
[#<Submission:0x00007f7e43d5ad18
  id: 1,
  user_id: 3,
  task_id: 2,
  created_at: Tue, 31 Oct 2023 12:37:39.966582000 UTC +00:00,
  updated_at: Tue, 31 Oct 2023 12:37:39.966582000 UTC +00:00>]
irb(main):015> t.submissions
  Submission Load (0.6ms)  SELECT `submissions`.* FROM `submissions` WHERE `
      submissions`.`task_id` = 2
=>
[#<Submission:0x00007f7e437c27e8
  id: 1,
  user_id: 3,
  task_id: 2,
  created_at: Tue, 31 Oct 2023 12:37:39.966582000 UTC +00:00,
  updated_at: Tue, 31 Oct 2023 12:37:39.966582000 UTC +00:00>]
irb(main):016> t.users
  User Load (1.6ms)  SELECT `users`.* FROM `users` INNER JOIN `submissions`
      ON `users`.`id` = `submissions`.`user_id` WHERE `submissions`.`task_id
      ` = 2
=>
[#<User:0x00007f7e432512a0
  id: 3,
  name: "valaki2",
  email: "valaki2@mail.bme.hu",
  neptun: "aaaaab",
  encrypted_password: "[FILTERED]",
  created_at: Tue, 31 Oct 2023 12:22:24.036142000 UTC +00:00,
```

```
  updated_at: Tue, 31 Oct 2023 12:22:24.036142000 UTC +00:00,
  salt: "[FILTERED]">]
irb(main):017> u.tasks
  Task Load (0.5ms)  SELECT `tasks`.* FROM `tasks` INNER JOIN `submissions`
      ON `tasks`.`id` = `submissions`.`task_id` WHERE `submissions`.`user_id
      ` = 3
=>
[#<Task:0x00007f7e42fa2838
  id: 2,
  number: 2,
  url: "http://gyakorlat.com/tasks/2",
  description: "Ez_is_nehez",
  deadline: Mon, 06 Nov 2023,
  created_at: Tue, 24 Oct 2023 10:33:55.491456000 UTC +00:00,
  updated_at: Tue, 24 Oct 2023 10:33:55.491456000 UTC +00:00>]
```

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.

```
u = User.new id: 1, name: "Valaki", neptun: 'aaaaaa', email: 'valaki@mail.
    bme.hu', password: 'titok'
u.save
t = Task.new id: 1, number: 1, url: 'http://gyakorlat.com/elsofeladat',
    description: 'De_nehez_az_iskolataska'
t.save
s = Submission.create user: u, task: t
```

Töltsük be ezeket az adatokat:

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

Rails konzolon ellenőrizhetjük, hogy az adatok létrejöttek-e az adatbázisban.

```
MariaDB [gyakorlat_development]> select * from users;
+----+--------+-----------------+---------+------------------------------------------+
| id | name   | email           | neptun  | encrypted_password
                      | created_at                  | updated_at
                  | salt                       |
+----+--------+-----------------+---------+------------------------------------------+

|  1 | Valaki | valaki@mail.bme.hu | aaaaaa |
    f7054522e8da9665bf10e1c94c1cfc745a95da1e | 2023-10-31 12:44:40.710832 |
    2023-10-31 12:44:40.710832 | RyhBMckGkxVwgf/BFGem/Q== |
+----+--------+-----------------+---------+------------------------------------------+

1 row in set (0.001 sec)

MariaDB [gyakorlat_development]> select * from solutions;
```

```
ERROR 1146 (42S02): Table 'gyakorlat_development.solutions' doesn't exist
MariaDB [gyakorlat_development]> select * from submissions;
+----+---------+---------+----------------------------+----------------------------+
| id | user_id | task_id | created_at                 | updated_at                 |
+----+---------+---------+----------------------------+----------------------------+
|  1 |       1 |       1 | 2023-10-31 12:44:40.749250 | 2023-10-31 12:44:40.749250 |
+----+---------+---------+----------------------------+----------------------------+
1 row in set (0.000 sec)

MariaDB [gyakorlat_development]> select * from tasks;
+----+--------+-----------------------------+-------------+------------+----------------------------+----------------------------+
| id | number | url                         | description | deadline   | created_at                 | updated_at                 |
+----+--------+-----------------------------+-------------+------------+----------------------------+----------------------------+
|  1 |      1 | http://gyakorlat.com/elsofeladat | De nehez az iskolataska | NULL | 2023-10-31 12:44:40.725436 | 2023-10-31 12:44:40.725436 |
+----+--------+-----------------------------+-------------+------------+----------------------------+----------------------------+
1 row in set (0.000 sec)
```