Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2022. október 25.

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 négy modellt hoztunk létre, a felhasználók `User` nevű modelljét, a tárgyak `Subject` nevű modelljét, a szemeszterek `Semester` nevű modelljét, valamint a kurzusok `Course` nevű modelljét. A felhasználók modellje a következőképp néz ki az adatbázisban.

```
MariaDB [pluto_development]> desc users;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| name       | varchar(255) | YES  |     | NULL    |                |
| email      | varchar(255) | YES  |     | NULL    |                |
| password   | varchar(255) | YES  |     | NULL    |                |
| pluto      | 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. 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:~/pluto> rails g migration AddSaltToUsers salt:string
      invoke  active_record
      create    db/migrate/20221018102006_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:~/pluto> rails c
Loading development environment (Rails 7.0.4)
irb(main):001:0> SecureRandom.hex 8
=> "d71793cd8e648e20"
irb(main):002:0> SecureRandom.base64 8
=> "z2U+qZnwJOY="
irb(main):003:0> Digest::SHA1.hexdigest 'titok'
=> "46ff53e764c4acf97b54db2020573049d2e3dab3"
```

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
    change_column :users, :name, :string, null: false#, index: true
    rename_column :users, :password, :encrypted_password
#     reversible do |dir|
#       dir.up { }
#       dir.down { }
#     end
  end

#   def up
#   end
#
#   def down
#   end
end
```

Hajtsuk végre a migrációt érvényre juttatandó az adatmodellünk változásait!

```
kovacsg@debian:~/pluto> rails db:migrate
== 20221018102006 AddSaltToUsers: migrating
   ===================================
-- add_column(:users, :salt, :string)
   -> 0.0089s
-- change_column(:users, :name, :string, {:null=>false})
   -> 0.0064s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0027s
== 20221018102006 AddSaltToUsers: migrated (0.0186s)
   ==========================
```

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 [pluto_development]> desc users;
+--------------------+--------------+------+-----+---------+----------------+
| Field              | Type         | Null | Key | Default | Extra          |
+--------------------+--------------+------+-----+---------+----------------+
| id                 | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| name               | varchar(255) | NO   |     | NULL    |                |
| email              | varchar(255) | YES  |     | NULL    |                |
| encrypted_password | varchar(255) | YES  |     | NULL    |                |
| pluto              | varchar(255) | 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)

MariaDB [pluto_development]> select * from users;
+----+------+---------------+--------------------+-------+----------------------------+----------------------------+------+
| id | name | email         | encrypted_password | pluto | created_at                 | updated_at                 | salt |
+----+------+---------------+--------------------+-------+----------------------------+----------------------------+------+
|  1 | Senki | senki@mail.bme.hu | titok          | aaaaaa | 2022-09-27 11:44:24.300391 | 2022-09-27 11:44:24.300391 | 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.

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, és a name attribútumot módosítottuk, hogy ne vehesse fel a NULL értéket.

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
    attr_accessor :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 self.password, self.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` az `id` attribútum értéke alapján keres, és egy rekordhoz tartozó Ruby objektumot ad vissza. A `first` (9. sor), `last` (10. sor) és `take` (11. 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. Az általános megoldás a keresésre a `where` (12. 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. Ha nemegyezésre keresünk, akkor a `where.not` (16. sor) függvényt használhatjuk. Kisebb, nagyobb relációkra csak sztrigben megadott feltétellel (15. sor) kereshetünk. Az `all` (8. sor) függvény az modell összes rekordját visszaadja egy tömbben.

```
kovacsg@debian:~/pluto> rails c
irb(main):008:0> User.all
  User Load (0.3ms)  SELECT `users`.* FROM `users`
=>
[#<User:0x000055c06c30c5c8
  id: 1,
  name: "Senki",
  email: "senki@mail.bme.hu",
  encrypted_password: "[FILTERED]",
  pluto: "aaaaaa",
  created_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
  updated_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
```

```
  salt: nil>]
irb(main):009:0> User.first
  User Load (0.3ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC
       LIMIT 1
=>
#<User:0x000055c06c160c88
 id: 1,
 name: "Senki",
 email: "senki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 pluto: "aaaaaa",
 created_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
 updated_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
 salt: nil>
irb(main):010:0> User.take
  User Load (0.3ms)  SELECT `users`.* FROM `users` LIMIT 1
=>
#<User:0x000055c06be067b8
 id: 1,
 name: "Senki",
 email: "senki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 pluto: "aaaaaa",
 created_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
 updated_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
 salt: nil>
irb(main):011:0> User.last
  User Load (0.3ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id`
      DESC LIMIT 1
=>
#<User:0x000055c06bae5d18
 id: 1,
 name: "Senki",
 email: "senki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 pluto: "aaaaaa",
 created_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
 updated_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
 salt: nil>
irb(main):012:0> User.where(name:'Senki')
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`name` = '
      Senki'
=>
[#<User:0x000055c06c43b8b8
  id: 1,
  name: "Senki",
  email: "senki@mail.bme.hu",
  encrypted_password: "[FILTERED]",
  pluto: "aaaaaa",
  created_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
  updated_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
  salt: nil>]
irb(main):013:0> User.where(name:'Senki', pluto: 'aaaaaa')
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`name` = '
      Senki' AND `users`.`pluto` = 'aaaaaa'
=>
[#<User:0x000055c06c13b578
  id: 1,
  name: "Senki",
  email: "senki@mail.bme.hu",
  encrypted_password: "[FILTERED]",
  pluto: "aaaaaa",
```

```
  created_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
  updated_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
  salt: nil>]
irb(main):014:0> User.where(name:'Senki', pluto: 'aaaaaa').take
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`name` = '
      Senki' AND `users`.`pluto` = 'aaaaaa' LIMIT 1
=>
#<User:0x000055c06bbbd650
 id: 1,
 name: "Senki",
 email: "senki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 pluto: "aaaaaa",
 created_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
 updated_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
 salt: nil>
irb(main):015:0> User.where("created_at > ?", Date.new(2022,9,1)).take
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE (created_at > '
      2022-09-01') LIMIT 1
=>
#<User:0x000055c06b9520f0
 id: 1,
 name: "Senki",
 email: "senki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 pluto: "aaaaaa",
 created_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
 updated_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
 salt: nil>
irb(main):016:0> User.where.not(name:'Valaki')
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`name` != '
      Valaki'
=>
[#<User:0x000055c06c4090e8
  id: 1,
  name: "Senki",
  email: "senki@mail.bme.hu",
  encrypted_password: "[FILTERED]",
  pluto: "aaaaaa",
  created_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
  updated_at: Tue, 27 Sep 2022 11:44:24.300391000 UTC +00:00,
  salt: nil>]
```

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:~/pluto> 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 bejelentkező képernyőn a form akciója a `/sessions/create`-re mutat-e, illetve a belépett felhasználó menüjében, vagyis a `layouts/_menu.html.erb`-ben 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, a bejelentkező képernyőnek pedig adjuk egy random nevet.

```
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 'home/login', to: 'home#login', as: 'hogyhivjuk'
  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(pluto_code, pass)
        user = User.where(pluto: pluto_code).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[:pluto], params[:password])
    if @user
        session[:user] = @user.id
        redirect_to hello_path
    else
        flash[:notice] = 'Invalid Pluto code or password'
        redirect_to login_path
    end
  end

  def destroy
    reset_session
    flash[:notice] = 'Logged out successfully'
    redirect_to login_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>
```

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_to hello_path
    else
      flash[:notice] = 'Unsuccessful registration'
      redirect_back fallback_location: registration_path
    end
  end

  def edit
  end

  def update
  end

  private
    def user_params
      params.require(:user).permit([:name, :email, :pluto, :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] 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. A felhasználó menüjében, illetve a profil szerkesztése képernyőn lecseréljük a profil linket és a form eseménykezelőjét, hogy azok az `edit_user_path`, illetve a `update_profile_path` függvényeket használják, és a paraméterünk az aktuálisan bejelentkezett felhasználó azonosítója, vagyis `@user.id`.

```
Rails.application.routes.draw do
  get 'users/new'
  post 'users/create', to: 'users#create', as: 'registration'
  get 'users/edit/:id', to: 'users#edit', as: 'edit_user'
  put 'users/update/:id', to: 'users#update', as: 'update_profile'
  get 'users/forgotten'
  match 'users/delete/:id', to: 'users#destroy', as: :delete_user, via: [:
      get, :delete]
  post 'users/send_forgotten'
end
```

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

```
<p>Hello, <%= @user.name %>!<p>
<%= flash[:notice] %>

<%= link_to "Profile", edit_user_path(@user.id) %><br/>
<%= link_to "Subjects", '/subjects' %><br/>
<%= link_to "Application", '/subjects' %><br/>
<%= link_to "Caledar", '/calendar' %><br />
<%= link_to "Logout", logout_path%>
```

1.4. Felhasználó által megadott adatok ellenőrzése

A regisztrációkor az elmentendő felhasználót még az elmentés előtt validáljuk, a név, az emailcím és a Pluto-kód attribútumoknak nemüresnek kell lennie (`:presence`), az emailcím és a Plutó-kód attribútumoknak emellett egyedinek is (`:uniqueness`) kell lenniük, és ha ez nem teljesül, akkor visszajelzünk, hogy nem megfelelő attribútumról van szó. 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 < ApplicationRecord
    validates :name, presence: true
    validates :email, { presence: true, uniqueness: true}
    validates :pluto, { 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:~/pluto> rails c
Loading development environment (Rails 7.0.4)
irb(main):001:0> u = User.new
=>
#<User:0x000055c93f0dd2a8
...
irb(main):002:0> u.save
  TRANSACTION (0.1ms)  BEGIN
  User Exists? (0.2ms)  SELECT 1 AS one FROM `users` WHERE `users`.`email`
       IS NULL LIMIT 1
  User Exists? (0.2ms)  SELECT 1 AS one FROM `users` WHERE `users`.`pluto`
       IS NULL LIMIT 1
  TRANSACTION (0.1ms)  ROLLBACK
=> false
irb(main):003:0> u.errors.messages
=> {:name=>["can't be blank"], :email=>["can't be blank"], :pluto=>["can't
    be blank"]}
irb(main):004:0> u = User.new name: 'Senki', email: 'senki@mail.bme.hu',
    pluto: 'aaaaaa'
=>
#<User:0x000055c93f4f2dc0
...
irb(main):005:0> u.save
  TRANSACTION (0.1ms)  BEGIN
  User Exists? (0.3ms)  SELECT 1 AS one FROM `users` WHERE `users`.`email` =
       'senki@mail.bme.hu' LIMIT 1
  User Exists? (0.2ms)  SELECT 1 AS one FROM `users` WHERE `users`.`pluto` =
       'aaaaaa' LIMIT 1
  TRANSACTION (0.1ms)  ROLLBACK
=> false
irb(main):006:0> u.errors.messages
=> {:email=>["has already been taken"], :pluto=>["has already been taken"]}
irb(main):007:0> u.errors.count
```

```
=> 2
irb(main):008:0> pluralize(u.errors.count, 'error')
Traceback (most recent call last):
(irb):8:in '<main>': undefined method 'pluralize' for main:Object (
    NoMethodError)
irb(main):009:0> u.errors.full_messages
=> ["Email has already been taken", "Pluto has already been taken"]
irb(main):010:0>
k
```

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

```
<h1>Registration</h1>

<p><%= flash[:notice] %></p>
<% if @user.errors.any? %>
    <div id="error_explanations">
        <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 (`config/locales/en.yml`):

```
en:
  activerecord:
    errors:
      models:
        user:
          attributes:
            email:
              blank: 'Email cannot be blank'
              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 hallgatói portálunk adatmodellje a következő elemekből áll:

- Felhasználó, hallgató (`User`)
- Tanév (`Semester`)
- Tantárgy (`Subject`)
- Kurzus (`Course`)

Ezeket a korábbi gyakorlatokon már létrehoztuk.

Egy kurzus (`Course`) egy tárgy egy szemeszterben indított példánya. A `references` típus a migrációban automatikusan létrehozta a `belongs_to` deklarációt a kurzus modellben. A reláció fordított irányú navigációjának engedélyezése végett módosítjuk a tanév és a tantárgy modellek osztályát. A kurzus kapcsolótáblaként működik a tanév és a tantárgy között, így közvetve kapcsoalatot teremthetünk közöttük a kurzusokon keresztül: a tanév tantárgyai, tantárgy tanévei. Ezt a `has_many` függvény `through` opciójával tesszük meg, amely értéke annak az egy-több kapcsolatnak a neve, amelyen keresztül elérjük a közvetett több-több kapcsolatot.

```
class Subject < ApplicationRecord
    has_many :courses
    has_many :semesters, through: :courses
end
class Semester < ApplicationRecord
    has_many :courses
    has_many :subjects, through: :courses
end
```

Egy hallgató sok kurzusra jelenthethet, és egy kurzusnak sok hallgatója lehet. Ez egy több-több kapcsolat. A gyakorlaton úgy döntöttünk, hogy magáról a jelentkezésről nem tartunk nyilván további adatot, ezért csak egy kapcsolótáblát hozunk létre. Először létrehozzuk a migrációt.

```
kovacsg@debian:~/pluto> rails g migration CreateJoinTableUserCourse
      invoke  active_record
      create    db/migrate/20221025113215_create_join_table_user_course.rb
```

A migráció fájljában egy kapcsolótábla létrehozását definiáljuk a két tábla nevének ábécé sorrendben való megadásával.

```
class CreateJoinTableUserCourse < ActiveRecord::Migration[7.0]
  def change
    create_join_table :courses, :users
  end
end
```

Végül érvényre juttatjuk a változást.

```
kovacsg@debian:~/pluto> rails db:migrate
== 20221025113215 CreateJoinTableUserCourse: migrating
   ========================
-- create_join_table(:course, :user)
   -> 0.0069s
```

```
== 20221025113215 CreateJoinTableUserCourse: migrated (0.0071s)
   ==============
```

A több-több kapcsolatot a modell osztályokban a `has_and_belongs_to_many` függvénnyel deklaráljuk.

```
class Course < ApplicationRecord
  belongs_to :subject
  belongs_to :semester
  has_and_belongs_to_many :users
end
class User < ApplicationRecord
    has_and_belongs_to_many :courses
end
```

Próbáljuk ki konzolon ezeket az új függvényeket. Vegyünk egy kurzust (1-2. sor), kérdezzük le azt a tárgyat, amelyhez tartozik (3. sor) és azt a tanévet, amelyben fut (4. sor). Vegyük egy tanévet (5. sor), kérdezzük le a kurzusait (6. sor) és a tantárgyait (7. sor). Vegyünk egy tantárgyat (10-11. sor), és kérdezzük le a kurzusait (13. sor) és a szemesztereit (14. sor).

```
kovacsg@debian:~/pluto> rails c
Loading development environment (Rails 7.0.4)
irb(main):001:0> c = Course.take
  Course Load (0.2ms)  SELECT `courses`.* FROM `courses` LIMIT 1
=>
#<Course:0x000055b3005cb058
...
irb(main):002:0> c
=>
#<Course:0x000055b3005cb058
 id: 1,
 subject_id: 1,
 semester_id: 1,
 t: "practice",
 date1: Tue, 11 Oct 2022 10:00:00.000000000 UTC +00:00,
 date2: Wed, 12 Oct 2022 12:00:00.000000000 UTC +00:00,
 location1: "Uszoda",
 location2: "Uszoda",
 limit: 100,
 created_at: Tue, 11 Oct 2022 14:25:29.995266000 UTC +00:00,
 updated_at: Tue, 11 Oct 2022 14:25:29.995266000 UTC +00:00>
irb(main):003:0> c.subject
  Subject Load (0.2ms)  SELECT `subjects`.* FROM `subjects` WHERE `subjects
      `.`id` = 1 LIMIT 1
=>
#<Subject:0x000055b2ffa45008
 id: 1,
 name: "Testnevelés",
 lecturer: "Futó Béla",
 pluto: "BMETT001",
 credit: 0,
 created_at: Tue, 11 Oct 2022 11:20:18.121945000 UTC +00:00,
 updated_at: Tue, 11 Oct 2022 11:20:18.121945000 UTC +00:00>
irb(main):004:0> c.semester
  Semester Load (0.2ms)  SELECT `semesters`.* FROM `semesters` WHERE `
      semesters`.`id` = 1 LIMIT 1
=>
#<Semester:0x000055b300947bf0
```

```
 id: 1,
 name: "2022/2023_spring",
 year: 2023,
 season: "spring",
 created_at: Tue, 11 Oct 2022 11:25:45.696363000 UTC +00:00,
 updated_at: Tue, 11 Oct 2022 11:26:14.456019000 UTC +00:00>
irb(main):005:0> s = c.semester
=>
#<Semester:0x000055b300947bf0
...
irb(main):006:0> s.courses
  Course Load (0.4ms)  SELECT `courses`.* FROM `courses` WHERE `courses`.`
      semester_id` = 1
=>
[#<Course:0x000055b300b6d790
  id: 1,
  subject_id: 1,
  semester_id: 1,
  t: "practice",
  date1: Tue, 11 Oct 2022 10:00:00.000000000 UTC +00:00,
  date2: Wed, 12 Oct 2022 12:00:00.000000000 UTC +00:00,
  location1: "Uszoda",
  location2: "Uszoda",
  limit: 100,
  created_at: Tue, 11 Oct 2022 14:25:29.995266000 UTC +00:00,
  updated_at: Tue, 11 Oct 2022 14:25:29.995266000 UTC +00:00>]
irb(main):007:0> s.subjects
  Subject Load (0.4ms)  SELECT `subjects`.* FROM `subjects` INNER JOIN `
      courses` ON `subjects`.`id` = `courses`.`subject_id` WHERE `courses`.`
      semester_id` = 1
=>
[#<Subject:0x000055b300beaa88
  id: 1,
  name: "Testnevelés",
  lecturer: "Futó_Béla",
  pluto: "BMETT001",
  credit: 0,
  created_at: Tue, 11 Oct 2022 11:20:18.121945000 UTC +00:00,
  updated_at: Tue, 11 Oct 2022 11:20:18.121945000 UTC +00:00>]
irb(main):010:0> t = s.subjects.where(name: 'Testnevelés').take
  Subject Load (0.4ms)  SELECT `subjects`.* FROM `subjects` INNER JOIN `
      courses` ON `subjects`.`id` = `courses`.`subject_id` WHERE `courses`.`
      semester_id` = 1 AND `subjects`.`name` = 'Testnevelés' LIMIT 1
=>
#<Subject:0x00007fc18087b6c8
...
irb(main):011:0> t
=>
#<Subject:0x00007fc18087b6c8
 id: 1,
 name: "Testnevelés",
 lecturer: "Futó_Béla",
 pluto: "BMETT001",
 credit: 0,
 created_at: Tue, 11 Oct 2022 11:20:18.121945000 UTC +00:00,
 updated_at: Tue, 11 Oct 2022 11:20:18.121945000 UTC +00:00>
irb(main):013:0> t.courses
  Course Load (0.3ms)  SELECT `courses`.* FROM `courses` WHERE `courses`.`
      subject_id` = 1
=>
[#<Course:0x000055b2fd8817a0
  id: 1,
```

```
  subject_id: 1,
  semester_id: 1,
  t: "practice",
  date1: Tue, 11 Oct 2022 10:00:00.000000000 UTC +00:00,
  date2: Wed, 12 Oct 2022 12:00:00.000000000 UTC +00:00,
  location1: "Uszoda",
  location2: "Uszoda",
  limit: 100,
  created_at: Tue, 11 Oct 2022 14:25:29.995266000 UTC +00:00,
  updated_at: Tue, 11 Oct 2022 14:25:29.995266000 UTC +00:00>]
irb(main):014:0> t.semesters
  Semester Load (0.3ms)  SELECT `semesters`.* FROM `semesters` INNER JOIN `
      courses` ON `semesters`.`id` = `courses`.`semester_id` WHERE `courses
      `.`subject_id` = 1
=>
[#<Semester:0x000055b300a7eac8
  id: 1,
  name: "2022/2023_spring",
  year: 2023,
  season: "spring",
  created_at: Tue, 11 Oct 2022 11:25:45.696363000 UTC +00:00,
  updated_at: Tue, 11 Oct 2022 11:26:14.456019000 UTC +00:00>]
```

A hallgatók és a kurzusok közötti több-több kapcsolat tömbként működik. Először meggyőződünk arról, hogy egy kurzusnak (1. sor) nincsenek hallgatói (2. sor), és fordítva, egy hallgató (3. sor) nincs feljelentkezve egy kurzusra (4. sor). A jelentkezés a tömbbe való beszúrás operátorral hajtjuk végre (5. sor), amely hatására bekerül az adatbázisba a kapcsolat.

```
kovacsg@debian:~/pluto> rails c
Loading development environment (Rails 7.0.4)
irb(main):001:0> c = Course.first
  Course Load (0.2ms)  SELECT `courses`.* FROM `courses` ORDER BY `courses
      `.`id` ASC LIMIT 1
=>
#<Course:0x000055c9c3f0dee8
...
irb(main):002:0> c.users
  User Load (0.2ms)  SELECT `users`.* FROM `users` INNER JOIN `courses_users
      ` ON `users`.`id` = `courses_users`.`user_id` WHERE `courses_users`.`
      course_id` = 1
=> []
irb(main):003:0> u = User.last
  User Load (0.3ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id`
      DESC LIMIT 1
=>
#<User:0x000055c9c4234bd0
...
irb(main):004:0> u.courses
  Course Load (0.3ms)  SELECT `courses`.* FROM `courses` INNER JOIN `
      courses_users` ON `courses`.`id` = `courses_users`.`course_id` WHERE `
      courses_users`.`user_id` = 4
=> []
irb(main):005:0> c.users << u
  TRANSACTION (0.2ms)  BEGIN
  Course::HABTM_Users Create (0.2ms)  INSERT INTO `courses_users` (`
      course_id`, `user_id`) VALUES (1, 4)
  TRANSACTION (0.4ms)  COMMIT
```

```
MariaDB [pluto_development]> select * from courses_users;
```

```
+-----------+---------+
| course_id | user_id |
+-----------+---------+
|         1 |       4 |
+-----------+---------+
1 row in set (0.000 sec)
```

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 tageket.

```
u = User.create name: "Senki", email: 'senki@mail.bme.hu', pluto: 'aaaaaa',
    password: 'titok'
```

Töltsük be ezeket az adatokat:

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