Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2023. május 2.

1. Session, felhasználókezelelés

1.1. Titkosított jelszó

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

```
MariaDB [randi_development]> select * from users;
+----+----------+-------------------+----------+----------------------------+----------------------------+
| id | username | email             | password | created_at                 | updated_at                 |
+----+----------+-------------------+----------+----------------------------+----------------------------+
|  1 | valaki   | valaki@mail.bme.hu | titok    | 2023-03-28 11:29:17.763533 | 2023-04-25 10:43:06.231163 |
+----+----------+-------------------+----------+----------------------------+----------------------------+
1 row in set (0.000 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:~/randi/db/migrate$ rails g migration AddSaltToUsers salt:
    string
      invoke  active_record
      create    db/migrate/20230502102140_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:~/randi/db/migrate$ rails c
Loading development environment (Rails 7.0.4.3)
irb(main):001:0> SecureRandom
=> SecureRandom
irb(main):002:0> SecureRandom.hex 8
=> "cd983f91c6db3462"
irb(main):003:0> SecureRandom.base64 8
=> "TpfZrO4sIYA="
irb(main):005:0> 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. Most azonban ezt a problémát a rekordok törlésével oldjuk meg.

```
irb(main):007:0> User.delete 1
  User Delete All (3.5ms)  DELETE FROM `users` WHERE `users`.`id` = 1
=> 1
```

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

Hajtsuk végre a migrációt érvényre juttatandó az adatmodellünk változásait!

```
kovacsg@debian:~/randi/db/migrate$ rails db:migrate
== 20230502102140 AddSaltToUsers: migrating
   ===================================
-- add_column(:users, :salt, :string)
   -> 0.0263s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0068s
== 20230502102140 AddSaltToUsers: migrated (0.0338s)
   ==========================
```

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 [randi_development]> desc users;
+--------------------+--------------+------+-----+---------+----------------+
| Field              | Type         | Null | Key | Default | Extra          |
+--------------------+--------------+------+-----+---------+----------------+
| id                 | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| username           | varchar(255) | YES  |     | NULL    |                |
| email              | varchar(255) | YES  |     | NULL    |                |
| encrypted_password | varchar(255) | YES  |     | NULL    |                |
| created_at         | datetime(6)  | NO   |     | NULL    |                |
| updated_at         | datetime(6)  | NO   |     | NULL    |                |
| salt               | varchar(255) | YES  |     | NULL    |                |
+--------------------+--------------+------+-----+---------+----------------+
7 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, 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
  attr_accessor :password
  before_save :encrypt_password

  def self.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(password, salt)
  end
end
```

Hozzunk létre egy felhasználót (1. sor), mentsük el (2. sor), és nézzük meg, hogy működik-e a jelszó titkosítása (3. sor).

```
kovacsg@debian:~/randi/app/models$ rails c
Loading development environment (Rails 7.0.4.3)
irb(main):001:0> u = User.new username: 'Senki', email: 'senki@mail.bme.hu',
     passw
ord: 'titok'
=>
#<User:0x000055fbfd3c19d8
...
irb(main):002:0> u.save
  TRANSACTION (1.1ms)  BEGIN
  User Create (1.9ms)  INSERT INTO `users` (`username`, `email`, `
      encrypted_password`, `created_at`, `updated_at`, `salt`) VALUES ('
      Senki', 'senki@mail.bme.hu', '2bdec43e317380b1f4bfcc5b057ca0c0291bbbff
      ', '2023-05-02 10:30:44.723526', '2023-05-02 10:30:44.723526', '
      JRlZNX1fe19BIvP9lAnHXw==')
  TRANSACTION (0.3ms)  COMMIT
```

```
=> true
irb(main):003:0> u.encrypted_password
=> "2bdec43e317380b1f4bfcc5b057ca0c0291bbbff"
```

Nézzük meg, hogy milyen elemi műveletekkel férhetünk egy vagy több adatbázisbeli rekordhoz hozzá Rails-ből. Az `all` (5. sor) függvény az modell összes rekordját visszaadja egy tömbben. A `find` az `id` attribútum értéke alapján keres (21. sor), és egy rekordhoz tartozó Ruby objektumot ad vissza. A `first` (6. sor), `last` (7. sor) és `take` (8. 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 (9. sor). 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` (17. sor) függvényt használhatjuk. Kisebb, nagyobb relációkra csak sztrigben megadott feltétellel (19-20. sor) kereshetünk.

```
irb(main):005:0> User.all
  User Load (0.3ms)  SELECT `users`.* FROM `users`
=>
[#<User:0x000055fbfdf27e58
  id: 1,
  username: "valaki",
  email: "valaki@mail.bme.hu",
  encrypted_password: "[FILTERED]",
  created_at: Tue, 28 Mar 2023 11:29:17.763533000 UTC +00:00,
  updated_at: Tue, 25 Apr 2023 10:43:06.231163000 UTC +00:00,
  salt: nil>,
 #<User:0x000055fbfdf27d40
  id: 2,
  username: "Senki",
  email: "senki@mail.bme.hu",
  encrypted_password: "[FILTERED]",
  created_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
  updated_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
  salt: "[FILTERED]">]
irb(main):006:0> User.first
  User Load (0.3ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC
       LIMIT 1
=>
#<User:0x000055fbfdccc730
 id: 1,
 username: "valaki",
 email: "valaki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 28 Mar 2023 11:29:17.763533000 UTC +00:00,
 updated_at: Tue, 25 Apr 2023 10:43:06.231163000 UTC +00:00,
 salt: nil>
irb(main):007:0> User.last
  User Load (0.3ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id`
```

```
      DESC LIMIT 1
=>
#<User:0x00007fe2a0775170
 id: 2,
 username: "Senki",
 email: "senki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 updated_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 salt: "[FILTERED]">
irb(main):008:0> User.take
  User Load (0.3ms)  SELECT `users`.* FROM `users` LIMIT 1
=>
#<User:0x000055fbfd988808
 id: 1,
 username: "valaki",
 email: "valaki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 28 Mar 2023 11:29:17.763533000 UTC +00:00,
 updated_at: Tue, 25 Apr 2023 10:43:06.231163000 UTC +00:00,
 salt: nil>
irb(main):009:0> User.find_by username: 'senki'
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`username`
      = 'senki' LIMIT 1
=>
#<User:0x000055fbfe0084d0
 id: 2,
 username: "Senki",
 email: "senki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 updated_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 salt: "[FILTERED]">
irb(main):010:0> User.find_by_username 'senki'
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`username`
      = 'senki' LIMIT 1
=>
#<User:0x000055fbfdd5ea18
 id: 2,
 username: "Senki",
 email: "senki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 updated_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 salt: "[FILTERED]">
irb(main):011:0> User.find_by_username_and_email 'senki', 'senki@mail.bme.hu
    '
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`username`
      = 'senki' AND `users`.`email` = 'senki@mail.bme.hu' LIMIT 1
=>
#<User:0x00007fe2a0445ef0
 id: 2,
 username: "Senki",
 email: "senki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 updated_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 salt: "[FILTERED]">
irb(main):012:0> User.where(username: 'senki')
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`username`
      = 'senki'
=>
```

```
[#<User:0x000055fbfdea0b60
  id: 2,
  username: "Senki",
  email: "senki@mail.bme.hu",
  encrypted_password: "[FILTERED]",
  created_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
  updated_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
  salt: "[FILTERED]">]
irb(main):013:0> User.find_by username: 'senki'
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`username`
      = 'senki' LIMIT 1
=>
#<User:0x000055fbfdcf3c68
 id: 2,
 username: "Senki",
 email: "senki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 updated_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 salt: "[FILTERED]">
irb(main):014:0> User.where(username: 'senki')
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`username`
      = 'senki'
=>
[#<User:0x000055fbfdc496c8
  id: 2,
  username: "Senki",
  email: "senki@mail.bme.hu",
  encrypted_password: "[FILTERED]",
  created_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
  updated_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
  salt: "[FILTERED]">]
irb(main):015:0> User.where(username: 'senki').first
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`username`
      = 'senki' ORDER BY `users`.`id` ASC LIMIT 1
=>
#<User:0x000055fbfd9b8b70
 id: 2,
 username: "Senki",
 email: "senki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 updated_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 salt: "[FILTERED]">
irb(main):016:0> User.where(username: 'senki').where(email: 'senki@mail.bme.
    hu')
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`username`
      = 'senki' AND `users`.`email` = 'senki@mail.bme.hu'
=>
[#<User:0x000055fbfdd6ec88
  id: 2,
  username: "Senki",
  email: "senki@mail.bme.hu",
  encrypted_password: "[FILTERED]",
  created_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
  updated_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
  salt: "[FILTERED]">]
irb(main):017:0> User.where.not(username: 'senki')
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`username`
      != 'senki'
=>
[#<User:0x000055fbfc26f978
```

```
  id: 1,
  username: "valaki",
  email: "valaki@mail.bme.hu",
  encrypted_password: "[FILTERED]",
  created_at: Tue, 28 Mar 2023 11:29:17.763533000 UTC +00:00,
  updated_at: Tue, 25 Apr 2023 10:43:06.231163000 UTC +00:00,
  salt: nil>]
irb(main):019:0> User.where("created_at <= ?", Date.today)
  User Load (1.6ms)  SELECT `users`.* FROM `users` WHERE (created_at <= '
      2023-05-02')
=>
[#<User:0x000055fbfd97e678
  id: 1,
  username: "valaki",
  email: "valaki@mail.bme.hu",
  encrypted_password: "[FILTERED]",
  created_at: Tue, 28 Mar 2023 11:29:17.763533000 UTC +00:00,
  updated_at: Tue, 25 Apr 2023 10:43:06.231163000 UTC +00:00,
  salt: nil>]
irb(main):020:0> User.where("created_at <= ?", Time.now)
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE (created_at <= '
      2023-05-02 10:36:37.801206')
=>
[#<User:0x00007fe2a015f830
  id: 1,
  username: "valaki",
  email: "valaki@mail.bme.hu",
  encrypted_password: "[FILTERED]",
  created_at: Tue, 28 Mar 2023 11:29:17.763533000 UTC +00:00,
  updated_at: Tue, 25 Apr 2023 10:43:06.231163000 UTC +00:00,
  salt: nil>,
 #<User:0x00007fe2a015f538
  id: 2,
  username: "Senki",
  email: "senki@mail.bme.hu",
  encrypted_password: "[FILTERED]",
  created_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
  updated_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
  salt: "[FILTERED]">]
irb(main):021:0> User.find 2
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`id` = 2
      LIMIT 1
=>
#<User:0x000055fbfdf8bf98
 id: 2,
 username: "Senki",
 email: "senki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 updated_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 salt: "[FILTERED]">
```

1.2. Be- és kijelentkezés

Következő lépésként tegyük rendbe a felhasználói session kezelését, ami a `menu` akcióinak megvalósítását, a jelszó titkosítását, és a titkosított jelszó adatbázisban való eltárolását jelenti első körben. Másodszor pedig a felhasz-

náló 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:~/randi/app/controllers$ 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/_user.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', via: [:delete, :get], as
      : 'logout'
  get 'say/hello', to: 'say#hello', as: 'hello'
end
```

A következő lépés a felhasználó hitelesítésének megvalósítása, amit a `User` modellben teszünk meg egy osztálymetódussal. A hitelesítés két argumentummal rendelkezik egy felhasználói email címmel és egy jelszóval, és a sikeresen hitelesített felhasználó objektumával vagy `nil`-lel tér vissza. Először megkeresi a rekordok között a felhasználó azonosítójának megfelelő rekordot,

majd elvégzi a hitelesítést. Bármelyik sikertelensége esetén a visszatérési érték `nil`. A hitelesítés (`authenticated?` metódus) azt ellenőrzi, hogy a titkosított jelszó attribútum megegyezik-e a jelszó titkosítása által visszaadott értékkel.

```
class User < ApplicationRecord
  def self.authenticate(email, password)
    user = User.where(email: email).take
    user && user.authenticated?(password) ? user : nil
  end

  def authenticated?(pass)
    self.encrypted_password == User.encrypt(pass, self.salt)
  end
end
```

Konzolon ellenőrizhetjük, hogy működik-e így a felhasználó hitelesítése.

```
kovacsg@debian:~/randi/app/views/layouts$ rails c
Loading development environment (Rails 7.0.4.3)
irb(main):001:0> User.authenticate 'senki@mail.bme.hu', 'titok'
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`email` = '
      senki@mail.bme.hu' LIMIT 1
=>
#<User:0x00007fa9207a9380
 id: 2,
 username: "Senki",
 email: "senki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 updated_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 salt: "[FILTERED]">
```

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 user name 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>
```

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 new
    @user = User.new
  end

  def create
    @user = User.new(user_params)
    if @user.save
      session[:user] = @user.id
      flash[:notice] = 'Successful registration'
      redirect_back fallback_location: hello_path
    else
      flash[:notice] = @user.errors.messages
      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

  def forgotten
  end

  def send_forgotten
  end

  private
    def user_params
      params.require(:user).permit([:username, :email, :password, :
          encrypted_password])
    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
```

A felhasználó így mindig a saját adatlapját látja. Ha a felhasználó nincs bejelentkezve, akkor a profil szerkesztése oldalra hibát kapunk. Ez alapvetően helyes működés. Ha azonban ez mégis ki szeretnénk cselezni, akkor a felhasználók kontrollerében is felveszünk egy `before_action` függvényt. Így az `edit` és `update` függvények minden körülmények között működnek.

```
class UsersController < ApplicationController
  before_action :find_user_by_id, only: [:edit, :update]

  private
    def find_user_by_id
```

```
      if @user.nil?
        @user = User.find params[:id]
      end
    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_profile_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', to: 'users#new', as: 'new'
  post 'users/create', to: 'users#create', as: 'register'
  get 'users/edit/:id', to: 'users#edit', as: 'edit_profile', default: {
      hello: 'hello'}
  put 'users/update/:id', to: 'users#update', as: 'update_profile'
  get 'users/forgotten'
  post 'users/send_forgotten', to: 'users#send_forgotten'
  resources :people
  get 'say/hello', to: 'say#hello', as: 'hello'
end
```

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

```
Hello, <%= @user.username %>!

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

<p><%= link_to 'Profile', edit_profile_path(@user.id) %></p>
<p><%= link_to 'Datasheet', '/people/1' %></p>
<p><%= link_to "Edit datasheet", '/people/1/edit' %></p>
<p><%= link_to "Search partner", '/people' %></p>
<p><%= link_to "Logout", logout_path %></p>
```

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 :username, 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:~/randi/app/models$ rails c
Loading development environment (Rails 7.0.4.3)
irb(main):001:0> u = User.new
=>
#<User:0x00007f509c41c320
...
irb(main):002:0> u.save
  TRANSACTION (0.1ms)  BEGIN
  User Exists? (1.0ms)  SELECT 1 AS one FROM `users` WHERE `users`.`email`
      IS NULL LIMIT 1
  TRANSACTION (0.2ms)  ROLLBACK
=> false
irb(main):003:0> u.errors.messages
=> {:username=>["can't be blank"], :email=>["can't be blank"]}
irb(main):004:0> u = User.new username: 'Senki', email: 'senki@mail.bme.hu'
=>
#<User:0x0000560814553e68
...
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
  TRANSACTION (0.1ms)  ROLLBACK
=> false
irb(main):006:0> u.errors.messages
=> {:email=>["has already been taken"]}
irb(main):010:0> u = User.new username: 'Valaki', email: 'valaki@mail.bme.hu
    ', password: 'titok', password_confirmation: 'titok2'
=>
#<User:0x00005608143981f0
...
irb(main):011:0> u.save
  TRANSACTION (0.1ms)  BEGIN
  User Exists? (0.3ms)  SELECT 1 AS one FROM `users` WHERE `users`.`email` =
```

```
        'valaki@mail.bme.hu' LIMIT 1
  TRANSACTION (0.1ms)  ROLLBACK
=> false
irb(main):012:0> u.errors.messages
=> {:password_confirmation=>["doesn't match Password"]}
irb(main):013:0> u = User.new username: 'Valaki', email: 'valaki@mail.bme.
    husword: 'titok', password_confirmation: 'titok'
=>
#<User:0x000056081306f330
...
irb(main):014:0> u.save
  TRANSACTION (0.2ms)  BEGIN
  User Exists? (0.3ms)  SELECT 1 AS one FROM `users` WHERE `users`.`email` =
     'valaki@mail.bme.hu' LIMIT 1
  User Create (1.8ms)  INSERT INTO `users` (`username`, `email`, `
    encrypted_password`, `created_at`, `updated_at`, `salt`) VALUES ('Valaki
    ', 'valaki@mail.bme.hu', 'b100ec40e67e680af6d35d34cf81f23a98d08eb9', '
    2023-05-02 11:28:40.864964', '2023-05-02 11:28:40.864964', '
    ISqEaLXYzunEOj2hQ+98dQ==')
  TRANSACTION (0.4ms)  COMMIT
=> true
```

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 randiportálunk adatmodellje a következő elemekből áll:

- Felhasználó (`User`)
- Felhasználói adatlap (`Person`)
- Üzenet (`Comment`)

Ezeket a korábbi gyakorlatokon már létrehoztuk.

A felhasználók és az adatlapok között jelenleg nincs kapcsolat, ezért fel kell vennünk egy idegen kulcsot az adatlap típusban.

```
kovacsg@debian:~/randi/db> rails g migration AddUserToPeople user:references
      invoke  active_record
      create    db/migrate/20230502113542_add_user_to_people.rb
```

Az automatikusan generált migrációs fájl jó, azonban az idegen kulcs nemnull volta miatt hibára futhatunk az adatbázisban lévő rekordok tekintetében. Töröljük a felhasználókat, majd hajtsuk végre a migrációt.

```
kovacsg@debian:~/randi/db> rails db:migrate
== 20230502113542 AddUserToPeople: migrating
   ===================================
-- add_reference(:people, :user, {:null=>false, :foreign_key=>true})
   -> 0.0261s
== 20230502113542 AddUserToPeople: migrated (0.0265s)
   ==========================
```

Ezek után beállíthatjuk az egy-egy kapcsolatot a felhasználók és az adatlapjaik között.

```
class User < ApplicationRecord
  has_one :person
end
```

```
class Person < ApplicationRecord
  belongs_to :user
end
```

Hozzuk létre a törölt felhasználót, majd állítsuk be az adatlapját konzolon, és közben ellenőrizzük a két kapcsolat settet és getter függvényeinek működését (2., 3. és 6. sor).

```
kovacsg@debian:~/randi/db$ rails c
Loading development environment (Rails 7.0.4.3)
irb(main):001:0> u = User.first
  User Load (0.2ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC
       LIMIT 1
=>
#<User:0x0000561d11540608
...
irb(main):002:0> u.person
  Person Load (6.0ms)  SELECT `people`.* FROM `people` WHERE `people`.`
      user_id` = 2 LIMIT 1
=> nil
irb(main):003:0> u.person = Person.new
  TRANSACTION (0.3ms)  BEGIN
  Person Create (1.1ms)  INSERT INTO `people` (`name`, `birthdate`, `gender
      `, `into`, `created_at`, `updated_at`, `user_id`) VALUES (NULL, NULL,
      NULL, NULL, '2023-05-02 11:42:34.494717', '2023-05-02 11:42:34.494717'
      , 2)
  TRANSACTION (0.4ms)  COMMIT
=>
#<Person:0x0000561d11825350
...
irb(main):004:0> Person.all
  Person Load (0.3ms)  SELECT `people`.* FROM `people`
=>
[#<Person:0x0000561d11963050
  id: 2,
  name: nil,
  birthdate: nil,
  gender: nil,
  into: nil,
  created_at: Tue, 02 May 2023 11:42:34.494717000 UTC +00:00,
  updated_at: Tue, 02 May 2023 11:42:34.494717000 UTC +00:00,
  user_id: 2>]
irb(main):005:0> p = Person.first
  Person Load (0.3ms)  SELECT `people`.* FROM `people` ORDER BY `people`.`id
      ` ASC LIMIT 1
=>
#<Person:0x0000561d117b8ca0
...
irb(main):006:0> p.user
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users`.`id` = 2
      LIMIT 1
=>
#<User:0x0000561d1171e538
 id: 2,
 username: "Senki",
 email: "senki@mail.bme.hu",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 updated_at: Tue, 02 May 2023 10:30:44.723526000 UTC +00:00,
 salt: "[FILTERED]">
irb(main):007:0>
```

Az adatlap automatikus inicializálását a felhasználó létrehozásakor kell megtennünk, ha a felhasználót sikeresen létrehoztuk.

```
class UsersController < ApplicationController
  def create
    @user = User.new(user_params)
```

```
    if @user.save
      @user.person = Person.new
      session[:user] = @user.id
      flash[:notice] = 'Successful registration'
      redirect_back fallback_location: hello_path
    else
      flash[:notice] = @user.errors.messages
      redirect_back fallback_location: hello_path
    end
  end
end
```

Az üzenetek és az adatlapok között már létezik a kapcsolatunk. Az üzenet egy felhasználói adatlapon jön létre. Az még hiányzik az üzenet modellből, hogy az üzenet egy felhasználó által jön létre. Adjuk ezt hozzá, és hajtsuk végre a migrációt. Jelenleg nincs üzenetünk az adatbázisban, ezért a kötelezően megadandó idegen kulcs beszúrása nem okozhat hibát.

```
kovacsg@debian:~/randi/app/models> rails c
Loading development environment (Rails 7.0.4.3)
irb(main):001:0> Comment.last
  Comment Load (0.2ms)  SELECT `comments`.* FROM `comments` ORDER BY `
      comments`.`id` DESC LIMIT 1
=> nil
irb(main):002:0>
kovacsg@debian:~/randi/db> rails g migration AddPersonToComments person:
    references
      invoke  active_record
      create    db/migrate/20230502114444_add_person_to_comments.rb
kovacsg@debian:~/randi/db$
kovacsg@debian:~/randi/db/migrate> rails db:migrate
== 20230502114444 AddPersonToComments: migrating
   ================================
-- add_reference(:comments, :person, {:null=>false, :foreign_key=>true})
   -> 0.0281s
== 20230502114444 AddPersonToComments: migrated (0.0285s)
   =======================
```

A felhasználók és az üzenetek, valamint az adatlapok és az üzenetek között egyaránt egy-több kapcsolat van. Az egy oldalon `has_many`, a több oldalon `belongs_to` deklarációt teszünk. Azt is kereshetővé tehetjük, hogy mely felhasználó írt üzenetet az egyes adatlapokra, és fordítva egy felhasználó mely adatlapokra írt üzenetet. Ezt a `through` opcióval tesszük meg, amellyel azt jelezzük, hogy a kapcsolat az üzenet táblán keresztül valósul meg, és az üzenet tábla kapcsolótáblaként is működik.

```
class User < ApplicationRecord
  has_many :comments
  has_many :people, through: :comments
end

class Person < ApplicationRecord
  has_many :comments
  has_many :users, through: :comments
end
```

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

Próbáljuk ki, hogy létrejöttek-e a kapcsolatok setter és getter függvényei. Az egy oldalon mindig a másik típus egy objektuma áll, a több oldalon pedig a másik típus példányaiból álló tömb.

```
kovacsg@debian:~/randi/app/models> rails c
Loading development environment (Rails 7.0.4.3)
irb(main):001:0> u = User.first
  User Load (0.2ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC
       LIMIT 1
=>
#<User:0x000055fef32dee28
...
irb(main):002:0> u.comments
  Comment Load (0.2ms)  SELECT `comments`.* FROM `comments` WHERE `comments
      `.`user_id` = 2
=> []
irb(main):003:0> u.people
  Person Load (0.3ms)  SELECT `people`.* FROM `people` INNER JOIN `comments`
       ON `people`.`id` = `comments`.`person_id` WHERE `comments`.`user_id`
      = 2
=> []
irb(main):004:0> p = Person.first
  Person Load (0.3ms)  SELECT `people`.* FROM `people` ORDER BY `people`.`id
      ` ASC LIMIT 1
=>
#<Person:0x000055fef3618148
...
irb(main):005:0> p.users
  User Load (1.5ms)  SELECT `users`.* FROM `users` INNER JOIN `comments` ON
      `users`.`id` = `comments`.`user_id` WHERE `comments`.`person_id` = 2
=> []
```

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:~/randi/db> rails db:seed
```