Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2021. március 30.

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 projektek `Item` nevű modelljét, valamint az erőforrások `Resources` nevű modelljét. A felhasználók modellje a következőképp néz ki az adatbázisban.

```
MariaDB [gyakorlat_development]> desc users;

+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| name       | varchar(255) | YES  |     | NULL    |                |
| email      | varchar(255) | YES  |     | NULL    |                |
| password   | varchar(255) | YES  |     | NULL    |                |
| created_at | datetime(6)  | NO   |     | NULL    |                |
| updated_at | datetime(6)  | NO   |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
6 rows in set (0.091 sec)

MariaDB [gyakorlat_development]> select * from users;
+----+---------+-----------------+----------+----------------------------+----------------------------+
| id | name    | email           | password | created_at                 | updated_at                 |
+----+---------+-----------------+----------+----------------------------+----------------------------+
|  1 | Valaki  | valaki@mail.bme.hu | titok    | 2021-03-02 14:51:01.183955 | 2021-03-02 14:51:01.183955 |
+----+---------+-----------------+----------+----------------------------+----------------------------+
1 row in set (0.013 sec)
```

Először növeljük portálunk biztonságát azzal, hogy a jelszavakat nem szövegesen, hanem titkosítva tároljuk. Ez a jelszó mező átnevezéséből és egy új, a titkosítás során a felhasználó számára egyedi attribútum felvételéből áll. A Rails az új attribútum felvételéhez képes automatikusan invertálható migrációt generálni jól definiált migrációnév esetén. A hozzáadott attribútumnak ilyen esetben a migráció neve után kell szerepelnie. Azonban az attribútum átnevezését magunknak kell majd hozzáadnunk a migrációhoz.

```
kovacsg@debian:~/gyakorlat/app/views/users> rails g migration AddSaltToUsers
    salt:string
Running via Spring preloader in process 15018
      invoke  active_record
      create    db/migrate/20210330102045_add_salt_to_users.rb
```

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

```
irb(main):019:0> SecureRandom
=> SecureRandom
irb(main):020:0> SecureRandom.hex 8
=> "bcfda401ad729ace"
irb(main):021:0> SecureRandom.hex 8
=> "3cbb324547a16a6a"
irb(main):022:0> Digest::SHA1.hexdigest 'titok'
=> "46ff53e764c4acf97b54db2020573049d2e3dab3"
irb(main):023:0> SecureRandom.base64 8
=> "VF7JQkmXIWg="
irb(main):024:0> SecureRandom.base64 16
=> "AkZ/6gW4BaJFuxIgRvTMMg=="
```

Mivel az attribútum átnevezése nem invertálható, vagy a `change` metódusban használjuk a `reversible` függvényt, vagy a `change` helyett két függvényt definiálunk `up`, illetve `down` néven. Most ez utóbbit választjuk. Az automatikusan generált invertálható műveletről el kell feledkeznünk, magunknak kell kettéválasztanunk a műveletet. Mivel az adatbázisban már van adat, fel irányú migráció esetén gondoskodunk kell azok új attribútumainak inicializálásáról.

```
class AddSaltToUsers < ActiveRecord::Migration[6.1]
  def up
    add_column :users, :salt, :string
    rename_column :users, :password, :encrypted_password
  end

  def down
    drop_column :users, :salt
    rename_column :users, :encrypted_password, :password
  end
end
```

Hajtsuk végre a migrációt érvényre juttatandó az adatmodellünk változásait!

```
kovacsg@debian:~/gyakorlat/db/migrate> rails db:migrate
== 20210330102045 AddSaltToUsers: migrating
   ================================
-- add_column(:users, :salt, :string)
   -> 0.0403s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0312s
== 20210330102045 AddSaltToUsers: migrated (0.0726s)
   =======================
```

Nézzük meg a migráció eredményét. Az adatbázisban már jelen lévő rekorddal nem tudunk most mit kezdeni. Ugyan titkosíthatjuk a jelszó attribútumát, de egy le-, majd egy felirányú migrációval az érvénytelen lesz, hiszen a hash egy egyirányú művelet, és így nem visszaállítható.

```
MariaDB [gyakorlat_development]> desc users;
+--------------------+--------------+------+-----+---------+----------------+
| Field              | Type         | Null | Key | Default | Extra          |
+--------------------+--------------+------+-----+---------+----------------+
| id                 | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| name               | varchar(255) | YES  |     | NULL    |                |
| email              | varchar(255) | YES  |     | NULL    |                |
| encrypted_password | varchar(255) | YES  |     | NULL    |                |
| 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 tárolására használt attribútumot és egy a típust tároló attribútumot, továbbá a jelszó attribútumot pedig átnevezzük, így az titkosítatlanul nem kerülhet bele az adatbázisba.

A jelszó attribútumot átneveztük, viszont a felületen továbbra is használjuk, ezért csak a modell osztályra korlátozva elérhetővé újra tesszük. Az attribútum csak a memóriában él, az adatbázisba nem kerül ki az értéke.

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

Bejelentkezéskor a megadott jelszót már az adatbázisban található titkosított jelszóval kell összevetnünk, ezért a `User` modell példányának mentésekor (regisztráció során vagy a felhasználói jelszó módosításakor) a `password` példányváltozót `encrypted_password` attribútummá kell transzformálnunk. Ezt a következőképp tesszük meg. Definiálunk egy `encrypt` azonosítójú osztálymetódust, amely a `password` és `salt` példányváltozók alapján egy hash függvénnyel egy kódolt karaktersorozatot hoz létre. Definiálunk továbbá egy `encrypt_password` azonosítójú metódust, amely az összes nem üres jelszó esetére elvégzi a titkosítást, illetve új, még el nem mentett rekord esetén inicializálja a `salt` attribútum értékét egy véletlen számmal. Végül a `before_save` metódussal jelezzük, hogy a `save` metódus minden egyes meghívása előtt hívódjék meg a `encrypt_password` metódus.

```
class User < ApplicationRecord
  before_save :encrypt_password

  def User.encrypt(password, salt)
    Digest::SHA1.hexdigest(password + 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 (2. sor), és nézzük meg, hogy működik-e a jelszó titkosítása.

```
kovacsg@debian:~/gyakorlat> rails c
Running via Spring preloader in process 17465
Loading development environment (Rails 6.1.3)
irb(main):001:0> u = User.new name: 'Valaki', email: 'valaki@mail.bme.hu',
    password: 'titok'
=> #<User id: nil, name: "Valaki", email: "valaki@mail.bme.hu",
    encrypted_password: nil, created_at: nil, updated_at: nil, salt: nil>
irb(main):002:0> u.save
  TRANSACTION (0.2ms)  BEGIN
  User Create (7.1ms)  INSERT INTO `users` (`name`, `email`, `
      encrypted_password`, `created_at`, `updated_at`, `salt`) VALUES ('
      Valaki', 'valaki@mail.bme.hu', '49
      eda2d4520c7fd9ceb0ff01d1e163f0850c6a68', '2021-03-30 10:37:00.029567',
       '2021-03-30 10:37:00.029567', '6OFxuJfyfoTTtiWmVwc5Lw==')
  TRANSACTION (2.7ms)  COMMIT
=> true
```

Nézzük meg, hogy milyen elemi műveletekkel férhetünk egy vagy több adatbázisbeli rekordhoz hozzá Rails-ből. A `find` az `id` attribútum érté-

ke alapján keres, és egy rekordhoz tartozó Ruby objektumot ad vissza. A `first`, `last` és `take` 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`, amely esetén a visszatérési érték `ActiveRecord::Base` objektumról `ActiveRecord::Relation` objektumra változik, ami tömbként működik, és kaszkádosan végrehajthatók rá a fenti keresési műveletek, például a `take`, amellyel egy találatot kivehetünk abból. Az `all` függvény az modell összes rekordját visszaadja egy tömbben.

```
kovacsg@debian:~/gyakorlat> rails c
Loading development environment (Rails 6.0.3.3)
irb(main):003:0> User.where(email: 'valaki@mail.bme.hu')
  User Load (0.7ms)  SELECT `users`.* FROM `users` WHERE `users`.`email` = '
      valaki@mail.bme.hu' /* loading for inspect */ LIMIT 11
=> #<ActiveRecord::Relation [#<User id: 2, name: "Valaki", email: "
    valaki@mail.bme.hu", encrypted_password: [FILTERED], created_at:
    "2021-03-30 10:37:00.029567000 +0000", updated_at: "2021-03-30
    10:37:00.029567000 +0000", salt: [FILTERED]>]>
irb(main):004:0> User.where(email: 'valaki@mail.bme.hu').take
  User Load (1.1ms)  SELECT `users`.* FROM `users` WHERE `users`.`email` = '
      valaki@mail.bme.hu' LIMIT 1
=> #<User id: 2, name: "Valaki", email: "valaki@mail.bme.hu",
    encrypted_password: [FILTERED], created_at: "2021-03-30
    10:37:00.029567000 +0000", updated_at: "2021-03-30 10:37:00.029567000
    +0000", salt: [FILTERED]>
irb(main):005:0> User.where(email: 'valaki2@mail.bme.hu').take
  User Load (1.1ms)  SELECT `users`.* FROM `users` WHERE `users`.`email` = '
      valaki2@mail.bme.hu' LIMIT 1
=> nil
irb(main):006:0> User.first
  User Load (0.6ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC
       LIMIT 1
=> #<User id: 2, name: "Valaki", email: "valaki@mail.bme.hu",
    encrypted_password: [FILTERED], created_at: "2021-03-30
    10:37:00.029567000 +0000", updated_at: "2021-03-30 10:37:00.029567000
    +0000", salt: [FILTERED]>
```

1.2. Be- és kijelentkezés

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

Ezután rátérhetünk a felhasználói session megvalósítására. Ehhez létrehozzuk a `sessions` kontrollert, amelynek `create` és `destroy` metódusai léptetik be, illetve ki a felhasználót.

```
kovacsg@debian:~/gyakorlat/app/controllers> rails g controller sessions
    create destroy
Running via Spring preloader in process 17886
      create  app/controllers/sessions_controller.rb
       route  get 'sessions/create'
get 'sessions/destroy'
      invoke  erb
      create    app/views/sessions
      create    app/views/sessions/create.html.erb
      create    app/views/sessions/destroy.html.erb
      invoke  test_unit
      create    test/controllers/sessions_controller_test.rb
      invoke  helper
      create    app/helpers/sessions_helper.rb
      invoke    test_unit
      invoke  assets
      invoke    scss
      create      app/assets/stylesheets/sessions.scss
```

A létrehozott `create` és `destroy` nézetekre nincs szükségünk, azokat töröljük. A `sessions` kontrollerhez ezek után nem tartozik nézet, a `login`, illetve a `logout` link eseményeit kezeli le. Ellenőrizzük, hogy a menüben, vagyis a `layouts/_guest_menu.html.erb`-ben a form akciója a `/sessions/create`-re mutat-e, illetve a belépett felhasználó menüjében a `Logout` link a `/sessions/destroy`-ra mutat-e. Belépéskor, illetve kilépéskor, megpróbálunk az aktuális oldalon maradni. A `routes.rb` konfigurációs fájlhoz adjuk hozzá a `post 'sessions/ create'` és a `get 'sessions/ destroy'`, ugyanis bejelentkezéskor HTTP POST üzenetben adatokat is küldünk a szerver felé, kijelentkezéskor pedig HTTP GET 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'
  get 'sessions/destroy', to: 'sessions#destroy', 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, pass)
    user = User.where(email: email).take
    user && user.authenticated?(pass) ? user : nil
  end

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

A kontrollerünk ezek után a következőképp néz ki. A hitelesítés imént megírt metódusának visszatérési értékét a `user` kontroller példányváltozóhoz rendeljük. Az email címet és a jelszót a `params` hash-ből vesszük ki a `menu`-ben megadott név alapján. A `params` hash alábbi használata veszélyes lehet, éles rendszerben ne használjuk közvetlenül! A SQL injection támadásokat elkerülendő az aposztrófokat escape-elnünk kell!

Ha a hitelesített felhasználó értéke nem `nil`, akkor a `session` hash `:user` szimbólummal hivatkozott értékének beállítjuk a felhasználó `id` attribútumának értékét, majd visszairányítjuk a felhasználót az előző oldalra. Ellenkező esetben egy hibaüzenetet küldünk a következő oldalnak a `flash` hash-en keresztül, és ugyancsak visszairányítjuk a felhasználót az előző oldalra. Kilépéskor töröljük a `session` hash tartalmát, és egy `flash` üzenettel visszairányítjuk a felhasználót az előző oldalra. Az előző oldal vagy a JavaScript history-ból, vagy a kérés fejrészének `HTTP_REFERER` opciójából határozható meg, ha egyik sem adott, akkor a helló világ oldalra kerükjünk át.

```
class SessionsController < ApplicationController
  def create
    @user = User.authenticate params[:email], params[:password]
    if @user
      session[:user] = @user.id
      flash[:notice] = 'Successful login'
    else
      flash[:notice] = 'Invalid email address or password'
    end
    redirect_back fallback_location: hello_path
  end

  def destroy
    reset_session
    flash[:notice] = 'Successful logout'
    redirect_back fallback_location: hello_path
  end
end
```

A flash hashen keresztül értéket adhatunk át a következő HTTP kérésre adott válasz számára. A megjelenítendő üzenet helye legyen az oldal szerkezetében a menü felett hozzáadjuk.

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

Ezek után már el tudjuk dönteni, hogy egy felhasználó mikor van bejelentkezve az előző alkalommal írt alkalmazás szintű helperben. Ha a `session :user` szimbólumhoz tartozó értéke nem üres, akkor a felhasználó be van jelentkezve.

```
module ApplicationHelper
  def logged_in?
    session[:user]
  end
end
```

1.3. Regisztráció, profil szerkesztése

Felhasználó létrehozásához és adatainak módosításához szükséges nézeteket már létrehoztuk, valósítsuk meg a formokat kezelő kontroller akciókat. A regisztrációhoz a `users` kontroller `create` akciója tartozik, a profil módosításához pedig az `update` akció tartozik. Takarítsuk ki az előző gyakorlaton bedrótozott értékeket a kontrollerből! A rekordok biztonsága végett a HTTP kérés paramétereinek lehetséges kulcsait korlátozzunk, és az alapján hozunk létre új felhasználót. Az ellenőrzést a `user_params` privát metódus végzi el.

```
class UsersController < ApplicationController
  def create
    @user = User.new(user_params)
    if @user.save
      flash[:user] = 'Successful registration'
      session[:user] = @user.id
      redirect_back fallback_location: hello_path
    else
      flash[:user] = @user.errors.messages
      redirect_to register_path
    end
  end

  def edit
  end

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

  private
    def user_params
      params.require(:user).permit([:name, :email, :password, :
          password_confirmation])
    end
end
```

A felhasználó `id` attribútumának értéke, akárcsak az HTTP kérés összes egyéb paramétere bekerül a `params` hash-be, ahonnan kikereshetjük, ha szükségünk van rá. Ha épp nem új felhasználót hozunk létre, akkor ezt meg kell tennünk. Bejelentkezett felhasználó esetén ugyanerre a célra felhasználhatjuk a sessionbel tárolt értéket. A felhasználó azonosító alapján való előrekeresésére több akció esetén is szükségünk van, de nem akarjuk többször ugyanazt a kódrészletet leírni, ezért felhasználjuk a `before_action` függvényt, melynek argumentuma egy függvényazonosító. Mivel több kontroller esetén is szükségünk van a felhasználóra, a keresést a kontrollerek közös ősosztályában tesszük meg. Az a függvény a kontroller összes publikus akciója előtt felut, ha benne van az `only` utáni felsorolásban, vagy nincs benne a `except` utáni felsorolásban az akció azonosítója. Ha egyik felsorolás sincs megadva, akkor mindenképp lefut a függvény. Ez a kódunk karbantarthatóságát javítja.

```
class ApplicationController < ActionController::Base
  before_action :find_user

  private
    def find_user
      @user = User.find session[:user] 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 'sessions/create', to: 'sessions#create', as: 'login'
  get 'sessions/destroy', to: 'sessions#destroy', as: 'logout'
  resources :resources
  get 'users/new', to: 'users#new', as: 'register'
  post 'users/create'
  get 'users/edit/:id', to: 'users#edit', as: 'edit_profile'
  put 'users/update/:id', to: 'users#update', as: 'update_profile'
  get 'users/show/:id', to: 'users#show', as: 'profile'
  get 'users/forgotten'
  post 'users/send_forgotten'
  resources :items
  get 'say/hello', to: 'say#hello', as: 'hello'

  root to: 'say#hello'
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> rails c
Running via Spring preloader in process 23292
Loading development environment (Rails 6.1.3)
irb(main):001:0> u = User.new
=> #<User id: nil, name: nil, email: nil, encrypted_password: nil,
    created_at: nil, updated_at: nil, salt: nil>
irb(main):002:0> u.save
  TRANSACTION (0.2ms)  BEGIN
  User Exists? (0.4ms)  SELECT 1 AS one FROM `users` WHERE `users`.`email`
      IS NULL LIMIT 1
  TRANSACTION (0.3ms)  ROLLBACK
=> false
irb(main):003:0> u.errors.messages
=> {:name=>["can't be blank"], :email=>["can't be blank"]}
irb(main):004:0> u.name = 'Valaki2'
=> "Valaki2"
irb(main):005:0> u.email = 'valaki@mail.bme.hu'
=> "valaki@mail.bme.hu"
irb(main):006:0> u.save
  TRANSACTION (1.8ms)  BEGIN
  User Exists? (0.8ms)  SELECT 1 AS one FROM `users` WHERE `users`.`email` =
      'valaki@mail.bme.hu' LIMIT 1
  TRANSACTION (0.3ms)  ROLLBACK
=> false
```

```
irb(main):007:0> u.errors.messages
=> {:email=>["has already been taken"]}
irb(main):008:0>
```

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

```
<h1>Registration</h1>
<p><%= flash[:user] %></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 |m| %>
        <li><%= m %></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:
            email:
              blank: 'Empty mail'
              take: '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 videómegosztó portálunk adatmodellje a következő elemekből áll:

- Felhasználó (`User`)
- Projekt (`Item`)
- Erőforrás (`Resource`)

Ezeket a korábbi gyakorlatokon már létrehoztuk.

Egy projekt (`Item`) a létrehozója egy felhasználó tulajdonában áll, de jelenleg nincs meg ez a kapcsolatunk, ezért egy új migrációt hozunk létre, amellyel egy, a felhasználók táblára hivatkozó idegen kulcsot adunk a táblánkhoz.

```
kovacsg@debian:~/gyakorlat/config> rails g migration AddUserToItems user:
    references
Running via Spring preloader in process 26849
      invoke  active_record
      create    db/migrate/20210330113551_add_user_to_items.rb
```

Mivel van már egy rekord a táblában, a migráció során be kell állítani annak az idegen kulcsát, amelyet például a default opcióval tehetjük meg.

```
class AddUserToItems < ActiveRecord::Migration[6.1]
  def change
    add_reference :items, :user, null: false, foreign_key: true #, default:
        2
    #Item.all.each do |item| item.update user: User.first end
  end
end
```

A reláció navigálhatóságának lehetővé tétele végett módosítjuk a felhasználók és a projektek modell osztályait.

```
class User < ApplicationRecord
  has_many :items
end
```

A projektek több erőforrást (fájlt) tartalmaznak. A `references` típus az erőforrás modellben automatikusan létrehozta a `belongs_to` deklarációt, a fordított irányban a kapcsolat hiányzik, ezért a projekt létrehozó felhasználójára hivatkozó deklaráció mellett azt is hozzáadjuk.

```
class Item < ApplicationRecord
  has_many :resources
  belongs_to :user
end
```

Nézzük meg, hogyan is működnek ezek a kapcsolatok a gyakorlatban. Keressünk egy felhasználót (1. sor), és kérdezzük le az összes projektjét (2. sor), majd vegyünk abból a tömbből egyet, és keressünk elő annak összes erőforrását (5. sor).

```
kovacsg@debian:~/gyakorlat> rails c
Running via Spring preloader in process 27301
Loading development environment (Rails 6.1.3)
irb(main):001:0> User.first
  User Load (0.6ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC
      LIMIT 1
=> #<User id: 2, name: "Valaki", email: "valaki@mail.bme.hu",
    encrypted_password: [FILTERED], created_at: "2021-03-30
    10:37:00.029567000 +0000", updated_at: "2021-03-30 10:37:00.029567000
    +0000", salt: [FILTERED]>
```

```
irb(main):002:0> User.first.items
  User Load (0.5ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC
       LIMIT 1
  Item Load (0.9ms)  SELECT `items`.* FROM `items` WHERE `items`.`user_id` =
       2 /* loading for inspect */ LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy [#<Item id: 1, name: "My
    project", description: "My project", created_at: "2021-03-16
    12:14:18.818988000 +0000", updated_at: "2021-03-16 12:14:18.818988000
    +0000", user_id: 2>]>
irb(main):004:0> User.first.items.first
  User Load (1.1ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC
       LIMIT 1
  Item Load (1.2ms)  SELECT `items`.* FROM `items` WHERE `items`.`user_id` =
       2 ORDER BY `items`.`id` ASC LIMIT 1
=> #<Item id: 1, name: "My project", description: "My project", created_at:
    "2021-03-16 12:14:18.818988000 +0000", updated_at: "2021-03-16
    12:14:18.818988000 +0000", user_id: 2>
irb(main):005:0> User.first.items.first.resources
  User Load (1.0ms)  SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC
       LIMIT 1
  Item Load (0.4ms)  SELECT `items`.* FROM `items` WHERE `items`.`user_id` =
       2 ORDER BY `items`.`id` ASC LIMIT 1
  Resource Load (7.0ms)  SELECT `resources`.* FROM `resources` WHERE `
      resources`.`item_id` = 1 /* loading for inspect */ LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy []>
irb(main):006:0> Resource.all
  Resource Load (1.2ms)  SELECT `resources`.* FROM `resources` /* loading
      for inspect */ LIMIT 11
=> #<ActiveRecord::Relation []>
```

Az erőforrásokhoz a későbbiekben tartozni fog egy-egy csatolmány, a projektekhez pedig kommentek.

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.

```
User.create name: "Valaki", email: "valaki@mail.bme.hu", password: 'titok',
    password_confirmation: 'titok'
```

Töltsük be ezeket az adatokat:

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