Rails MVC, modell, session
Gyakorlat

Kovács Gábor

2018. október 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 alkalommak négy modellt hoztunk létre, a felhasználók `User` nevű modelljét, a feladatsorok `Quiz` nevű modelljét, a feladatot `Task` nevű modelljét és az értékelések `Review` nevű modelljét. A felhasználók modellje a következőképp néz ki az adatbázisban.

```
MariaDB [gyakorlat_development]> show tables;
+-----------------------------------+
| Tables_in_gyakorlat_development |
+-----------------------------------+
| ar_internal_metadata              |
| quizzes                           |
| reviews                           |
| schema_migrations                 |
| tasks                             |
| users                             |
+-----------------------------------+
6 rows in set (0.00 sec)

MariaDB [gyakorlat_development]> desc users;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| name       | varchar(255) | YES  |     | NULL    |                |
| password   | varchar(255) | YES  |     | NULL    |                |
| email      | varchar(255) | YES  |     | NULL    |                |
| neptun     | varchar(255) | YES  |     | NULL    |                |
| usertype   | int(11)      | YES  |     | NULL    |                |
| created_at | datetime     | NO   |     | NULL    |                |
| updated_at | datetime     | NO   |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
8 rows in set (0.00 sec)
```

```
MariaDB [gyakorlat_development]> select * from users;
+----+---------+----------+-------------------+---------+----------+---------------------+---------------------+
| id | name    | password | email             | neptun  | usertype | created_at          | updated_at          |
+----+---------+----------+-------------------+---------+----------+---------------------+---------------------+
|  1 | Valaki  | titok    | valaki@mail.bme.hu | aaaaaa  |        0 | 2018-09-25 11:18:40 | 2018-09-25 11:18:40 |
+----+---------+----------+-------------------+---------+----------+---------------------+---------------------+
1 row in set (0.00 sec)

MariaDB [gyakorlat_development]> Bye
```

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# rails g migration AddSaltToUsers salt:string
      invoke  active_record
      create    db/migrate/20181030111950_add_salt_to_users.rb
```

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

```
kovacsg@debian:~/gyakorlat# rails c
Loading development environment (Rails 5.2.1)
irb(main):001:0> SecureRandom
=> SecureRandom
irb(main):002:0> SecureRandom.hex 8
=> "1836f3ffa6479721"
irb(main):003:0> SecureRandom.hex 16
=> "fc89a02752686c040d93c3759964a5ae"
irb(main):004:0> SecureRandom.base64 16
=> "N0jIW99BHYKykuWEG+XzuQ=="
irb(main):006:0> Digest::SHA1.hexdigest 'titok'
=> "46ff53e764c4acf97b54db2020573049d2e3dab3"
irb(main):007:0>
```

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

```
class AddSaltToUsers < ActiveRecord::Migration[5.2]
  # def change
    #reversible.do |dir|
    #  dir.up
    #  dir.down
    # end
  # end

  def up
    add_column :users, :salt, :string
    rename_column :users, :password, :encrypted_password
  end

  def down
    remove_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
== 20181030111950 AddSaltToUsers: migrating
   ===================================
-- add_column(:users, :salt, :string)
   -> 0.0252s
-- rename_column(:users, :password, :encrypted_password)
   -> 0.0058s
== 20181030111950 AddSaltToUsers: migrated (0.0312s)
   ==========================
```

Nézzük meg a migráció eredményét. Az adatbázisban már jelen lévő rekorddal nem tudunk 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. Azt a rekordot meg kell semmisítenünk.

```
MariaDB [gyakorlat_development]> desc users;
+-----------------------+--------------+------+-----+---------+----------------+
| Field                 | Type         | Null | Key | Default | Extra          |
+-----------------------+--------------+------+-----+---------+----------------+
| id                    | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| name                  | varchar(255) | YES  |     | NULL    |                |
| encrypted_password    | varchar(255) | YES  |     | NULL    |                |
| email                 | varchar(255) | YES  |     | NULL    |                |
| neptun                | varchar(255) | YES  |     | NULL    |                |
| usertype              | int(11)      | YES  |     | NULL    |                |
| created_at            | datetime     | NO   |     | NULL    |                |
```

```
| updated_at          | datetime     | NO   |     | NULL    |
    |
| salt                | varchar(255) | YES  |     | NULL    |
    |
+---------------------+--------------+------+-----+---------+-----------------+

9 rows in set (0.00 sec)

MariaDB [gyakorlat_development]> select * from users;
+----+---------+--------------------+-----------------+---------+----------+
| id | name    | encrypted_password | email           | neptun  | usertype
    | created_at          | updated_at          | salt |
+----+---------+--------------------+-----------------+---------+----------+
|  1 | Valaki  | titok              | valaki@mail.bme.hu | aaaaaa |        0
    | 2018-09-25 11:18:40 | 2018-09-25 11:18:40 | NULL |
+----+---------+--------------------+-----------------+---------+----------+

1 row in set (0.00 sec)

MariaDB [gyakorlat_development]> Bye
```

A `new_record?` függvény azt állapítja meg egy ActiveRecord objektumról, hogy azt elmentettük-e már az adatbázisba, ezt az `id` attribútum ellenőrzésével végzi el, ami alapértelmezés szerint `nil`. A memóriában létrehozott `ActiveRercord` objektumok az első mentés műveletig új rekordnak számítanak, az `id` attribútumuk csak annak hatására inicializálódik.

A migrációban minden egyes felhasználói rekordhoz hozzáadtunk egy egyéni a jelszó titkosításához használt random kulcs tárololására használt attribútumot és egy a típust tároló attribútumot, továbbá a jelszó attribútumot pedig átnevezzük, így az titkosítatlanul nem kerülhet bele az adatbázisba.

A jelszó attribútumot átneveztük, viszont a felületen továbbra is használjuk, ezért csak a modell osztályra korlátozva elérhetővé újra tesszük.

```
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` példányváltozótót `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 meg-

hívása előtt hívódjék meg a `encrypt_password` metódus.

```
class User < ApplicationRecord
  before_save :encrypt_password

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

  def encrypt_password
    return if password.blank?
    if new_record?
      self.salt = SecureRandom.base64 8
    end
    self.encrypted_password = User.encrypt(password, salt)
  end
end
```

Nézzük meg, hogy milyen elemi műveletekkel férhetünk egy vagy több adatbázisbeli rekordhoz hozzá Rails-ből. A `first` (6. sor), `last` (5. sor) és `take` (7. 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 (8. sor), a visszatérési érték `ActiveRecord::Base` objektumról `ActiveRecord::Relation` objektumra változik, ami tömbként működik, és kaszkádosan végrehajthatók rá a keresési műveletek. Az `all` függvény az modell összes rekordját visszaadja egy tömbben.

```
kovacsg@debian:~/gyakorlat# rails c
Loading development environment (Rails 5.2.1)
irb(main):001:0> User.find_by email: 'senki@mail.bme.hu'
   (0.2ms)  SET NAMES utf8,  @@SESSION.sql_mode = CONCAT(CONCAT(@@sql_mode,
       ',STRICT_ALL_TABLES'), ',NO_AUTO_VALUE_ON_ZERO'),  @@SESSION.
       sql_auto_is_null = 0, @@SESSION.wait_timeout = 2147483
  User Load (0.3ms)  SELECT  `users`.* FROM `users` WHERE `users`.`email` =
      'senki@mail.bme.hu' LIMIT 1
=> #<User id: 2, name: "Senki", encrypted_password: "
    efb935d3efbc456d8d27a218a4078408bf2888a5", email: "senki@mail.bme.hu",
    neptun: nil, usertype: 0, created_at: "2018-10-30 11:30:40", updated_at:
     "2018-10-30 11:30:40", salt: "Zf3gg8BdfWg=">
irb(main):002:0> User.where email: 'senki@mail.bme.hu'
  User Load (0.4ms)  SELECT  `users`.* FROM `users` WHERE `users`.`email` =
      'senki@mail.bme.hu' LIMIT 11
=> #<ActiveRecord::Relation [#<User id: 2, name: "Senki", encrypted_password
    : "efb935d3efbc456d8d27a218a4078408bf2888a5", email: "senki@mail.bme.hu
    ", neptun: nil, usertype: 0, created_at: "2018-10-30 11:30:40",
    updated_at: "2018-10-30 11:30:40", salt: "Zf3gg8BdfWg=">]>
irb(main):003:0> User.where(email: 'senki@mail.bme.hu').where(usertype: 0)
  User Load (0.4ms)  SELECT  `users`.* FROM `users` WHERE `users`.`email` =
      'senki@mail.bme.hu' AND `users`.`usertype` = 0 LIMIT 11
=> #<ActiveRecord::Relation [#<User id: 2, name: "Senki", encrypted_password
    : "efb935d3efbc456d8d27a218a4078408bf2888a5", email: "senki@mail.bme.hu
    ", neptun: nil, usertype: 0, created_at: "2018-10-30 11:30:40",
    updated_at: "2018-10-30 11:30:40", salt: "Zf3gg8BdfWg=">]>
irb(main):004:0> User.where(email: 'senki@mail.bme.hu').where(usertype: 1)
  User Load (0.4ms)  SELECT  `users`.* FROM `users` WHERE `users`.`email` =
      'senki@mail.bme.hu' AND `users`.`usertype` = 1 LIMIT 11
=> #<ActiveRecord::Relation []>
```

```
irb(main):005:0> User.last
  User Load (0.3ms)  SELECT  `users`.* FROM `users` ORDER BY `users`.`id`
      DESC LIMIT 1
=> #<User id: 2, name: "Senki", encrypted_password: "
    efb935d3efbc456d8d27a218a4078408bf2888a5", email: "senki@mail.bme.hu",
    neptun: nil, usertype: 0, created_at: "2018-10-30 11:30:40", updated_at:
     "2018-10-30 11:30:40", salt: "Zf3gg8BdfWg=">
irb(main):006:0> User.first
  User Load (0.3ms)  SELECT  `users`.* FROM `users` ORDER BY `users`.`id`
      ASC LIMIT 1
=> #<User id: 1, name: "Valaki", encrypted_password: "titok", email: "
    valaki@mail.bme.hu", neptun: "aaaaaa", usertype: 0, created_at:
    "2018-09-25 11:18:40", updated_at: "2018-09-25 11:18:40", salt: nil>
irb(main):007:0> User.take
  User Load (0.3ms)  SELECT  `users`.* FROM `users` LIMIT 1
=> #<User id: 1, name: "Valaki", encrypted_password: "titok", email: "
    valaki@mail.bme.hu", neptun: "aaaaaa", usertype: 0, created_at:
    "2018-09-25 11:18:40", updated_at: "2018-09-25 11:18:40", salt: nil>
irb(main):008:0> User.take(2)
  User Load (0.4ms)  SELECT  `users`.* FROM `users` LIMIT 2
=> [#<User id: 1, name: "Valaki", encrypted_password: "titok", email: "
    valaki@mail.bme.hu", neptun: "aaaaaa", usertype: 0, created_at:
    "2018-09-25 11:18:40", updated_at: "2018-09-25 11:18:40", salt: nil>, #<
    User id: 2, name: "Senki", encrypted_password: "
    efb935d3efbc456d8d27a218a4078408bf2888a5", email: "senki@mail.bme.hu",
    neptun: nil, usertype: 0, created_at: "2018-10-30 11:30:40", updated_at:
     "2018-10-30 11:30:40", salt: "Zf3gg8BdfWg=">]
irb(main):009:0> User.where(email: 'senki@mail.bme.hu').first
  User Load (0.4ms)  SELECT  `users`.* FROM `users` WHERE `users`.`email` =
      'senki@mail.bme.hu' ORDER BY `users`.`id` ASC LIMIT 1
=> #<User id: 2, name: "Senki", encrypted_password: "
    efb935d3efbc456d8d27a218a4078408bf2888a5", email: "senki@mail.bme.hu",
    neptun: nil, usertype: 0, created_at: "2018-10-30 11:30:40", updated_at:
     "2018-10-30 11:30:40", salt: "Zf3gg8BdfWg=">
irb(main):010:0>
```

Elsődleges kulcs alapján, amely jellemzően a weboldalon megjelenő azonosító önmaga, a `find` metódussal kereshetünk egy objektumot. Egy objektum törlése is történhet az azonosító alapján. Valamely attribútum értéke alapján való keresést többféleképpen végezhetünk. Ezek a metódusok minden esetben egy Ruby tömböt adnak vissza, amely azonban lehet akár üres is, vagy tartalmazhat egyetlen objektumot is. A Rails 4-es verziójától a `where` keresőmetódus (előző kódblokk 4. sor) a javasolt. Ezzel nem csak paraméter egyezésére kereshetünk, hanem egy string paraméterrel általánosabb feltételt is megadhatunk, például egy attribútum értéke és egy szám összehasonlítását. A keresőmetódusok tetszőleges számú alkalommal egymás után láncolhatók, a `where` által visszaadott tömbből jellemzően a `take` (vagy `first`) metódussal vesszük ki a keresés eredményét, ha tudjuk, hogy pontosan egy lesz.

```
irb(main):014:0> User.find 1
  User Load (0.4ms)  SELECT  `users`.* FROM `users` WHERE `users`.`id` = 1
      LIMIT 1
=> #<User id: 1, name: "Valaki", encrypted_password: "titok", email: "
    valaki@mail.bme.hu", neptun: "aaaaaa", usertype: 0, created_at:
```

```
    "2018-09-25 11:18:40", updated_at: "2018-09-25 11:18:40", salt: nil>
irb(main):015:0> User.delete 1
  User Destroy (10.8ms)  DELETE FROM `users` WHERE `users`.`id` = 1
=> 1
```

Hozzunk létre egy felhasználót, és nézzük meg, hogy működik-e a jelszó titkosítássa.

```
kovacsg@debian:~/gyakorlat# rails c
Loading development environment (Rails 5.2.1)
irb(main):001:0> u = User.new
   (0.4ms)  SET NAMES utf8,  @@SESSION.sql_mode = CONCAT(CONCAT(@@sql_mode,
       ',STRICT_ALL_TABLES'), ',NO_AUTO_VALUE_ON_ZERO'),  @@SESSION.
       sql_auto_is_null = 0, @@SESSION.wait_timeout = 2147483
=> #<User id: nil, name: nil, encrypted_password: nil, email: nil, neptun:
    nil, usertype: nil, created_at: nil, updated_at: nil, salt: nil>
irb(main):002:0> u.name = 'Senki'
=> "Senki"
irb(main):003:0> u.password = 'titok'
=> "titok"
irb(main):004:0> u.email = 'senki@mail.bme.hu'
=> "senki@mail.bme.hu"
irb(main):005:0> u.usertype = 0
=> 0
irb(main):006:0> u.save
   (0.2ms)  BEGIN
  User Create (0.2ms)  INSERT INTO `users` (`name`, `encrypted_password`, `
      email`, `usertype`, `created_at`, `updated_at`, `salt`) VALUES ('Senki
      ', 'efb935d3efbc456d8d27a218a4078408bf2888a5', 'senki@mail.bme.hu', 0,
       '2018-10-30 11:30:40', '2018-10-30 11:30:40', 'Zf3gg8BdfWg=')
   (7.4ms)  COMMIT
=> true
```

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/config# rails g controller sessions create
    destroy
      create  app/controllers/sessions_controller.rb
       route  get 'sessions/create'
get 'session/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    coffee
      create      app/assets/javascripts/sessions.coffee
      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/_loginform.html.erb`-ben a form akciója a `/sessions/create`-re mutat-e, illetve a belépett hallgató vagy oktató típusú felhasználó menüjében a `Logout` link a `/sessions/ destroy`-ra mutat-e. Belépéskor, illetve kilépéskor, megpróbálunk az aktuális oldalon maradni. A `routes.rb` konfigurációs fájlhoz adjuk hozzá a `post 'sessions/ create'` és a `get 'sessions/ destroy'`, ugyanis bejelentkezéskor HTTP POST üzenetben adatokat is küldünk a szerver felé, kijelentkezéskor pedig HTTP GET vagy DELETE üzenet elég. Valósítsuk ezeket meg, és rendeljük hozzájuk a `login`, illetve a `logout` címkéket. Ez utóbbiak `login_path` és `login_url` azonosítóval segédfüggvényeket hoznak létre, amelyekkel az útvonalra, illetve a teljes URL-re hivatkozjatunk a `login` alias esetén, a `logout` alias hasonlóan működik. Végül nevezzük el a hello világ nézetünket `hello`-nak.

```
Rails.application.routes.draw do
  post 'sessions/create', to: 'sessions#create', as: 'login'
  match 'sessions/destroy', to: 'sessions#destroy', as: 'logout', via:  [:
      get, :delete]
  get 'say/hello', to: 'say#hello', as: 'hello'
end
```

A következő lépés a felhasználó hitelesítésének megvalósítása, amit a `User` modellben teszünk meg egy osztálymetódussal. A hitelesítés két argumentummal rendelkezik egy felhasználói email címmel és egy jelszóval, és a sikeresen hitelesített felhasználó objektumával vagy `nil`-lel tér vissza. Először megkeresi a rekordok között a felhasználó azonosítójának megfelelő rekordot, majd elvégzi a hitelesítést. Bármelyik sikertelensége esetén a visszatérési érték `nil`. A hitelesítés (`authenticated?` metódus) azt ellenőrzi, hogy a titkosított jelszó attribútum megegyezik-e a jelszó titkosítása által visszaadott értékkel.

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

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

```
end
```

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

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

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

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

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

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

```
  end
end
```

1.3. Regisztráció, profil szerkesztése

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

```
class UsersController < ApplicationController
  def create
    @user = User.new(user_params)
    if @user.save
      session[:user] = @user.id
      flash[:notice] = 'Successful login'
      redirect_to hello_path
    else
      flash[:notice] = 'Email address in use or non-matching passwords'
      redirect_back fallback_location: register_path
    end
  end

  def edit
  end

  def update
    if @user.update(user_params)
      flash[:notice] = 'Update successful'
      redirect_to hello_path
    else
      flash[:notice] = 'Could not update personal data'
      redirect_back fallback_location: register_path
    end
  end

  private
    def find_user
      @user = User.find params[:id]
    end

    def user_params
      params.require(:user).permit(:name, :email, :password, :
          password_confirmation, :usertype, :neptun)
    end
end
```

A felhasználó `id` attribútumának értéke, akárcsak az HTTP kérés összes egyéb paramétere bekerül a `params` hash-be, ahonnan kikereshetjük, ha szükségünk van rá. Ha épp nem új felhasználót hozunk létre, akkor ezt meg kell tennünk. Bejelentkezett felhasználó esetén ugyanerre a célra felhasználhatjuk

a sessionbel tárolt értéket. A felhasználó azonosító alapján való előrekeresésére több akció esetén is szükségünk van, de nem akarjuk többször ugyanazt a kódrészletet leírni, ezért felhasználjuk a `before_action` függvényt, melynek argumentuma egy függvényazonosító. 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 UsersController < ApplicationController
  before_action :find_user, only: [ :edit, :update, :destroy, :show ] #
      except: [:new, :create, :forgotten, :send_forgotten]
  private
    def find_user
      @user = User.find params[:id]
    end
end
```

Több felhasználó számára is elérhetővé szeretnénk tenni a portálunkat, ez lehetséges, mert a felhasználók adatait most már az adatbázisból vesszük elő. Ha egy konkrét felhasználó adait szeretnénk megnézni, szerkeszteni vagy módosítani, akkor a HTTP kérés paramétereként át kell adnunk a felhasználó adatbázisbeli azonosítóját is, hogy a megfelelő felhasználó adatai jelenlenek meg, módosuljanak. Az első módosítás a HTTP kérés paraméterezhetővé tétele, amit a `routes.rb` konfiguráció állományban teszünk meg – magyarázat a következő előadáson.

```
  get 'users/new', to: 'users#new', as: 'register'
  post 'users/create'

  get 'users/edit/:id', to: 'users#edit', as: 'edit_profile'
  put 'users/update/:id', to: 'users#update', as: 'update_profile'

  get 'users/forgotten'
  post 'users/send_forgotten'

  delete 'users/destroy'

  get 'users/show/:id', to: 'users#show', as: 'profile'
```

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. A Neptun-kód pedig hat alfanumerikus karakterből áll. A felhasználó típusa `lecturer` vagy `student` lehet, erre bevezetünk egy enumot.

```
class User < ActiveRecord::Base
  enum usertype: [:lecturer, :student]

  validates :email, { presence: true, uniqueness: true }
  validates :name, presence: true
  validates :password, confirmation: true, if: :password_required?
  validates :neptun, format: { with: /[a-zA-Z0-9]{6}/ }

  def password_required?
    self.new_record? || !self.password.blank?
  end
end
```

Konzolon és a webfelületen ellenőrizhetjük, hogy elmenthető-e üres névvel névvel és létező email címmel egy új rekord! Az `ActiveRecord` példányokról a `valid?` metódussal kérdezhetjük meg, hogy átmennek-e az osztályában definiált validációkon. Ha nem, akkor a hibaüzeneteket az `errors` példányváltozóban érhetjük el, amiket kivezethetünk a nézetekre.

```
kovacsg@debian:~/gyakorlat# rails c
Loading development environment (Rails 5.2.1)
irb(main):001:0> User.new
   (0.3ms)  SET NAMES utf8,  @@SESSION.sql_mode = CONCAT(CONCAT(@@sql_mode,
       ',STRICT_ALL_TABLES'), ',NO_AUTO_VALUE_ON_ZERO'),  @@SESSION.
       sql_auto_is_null = 0, @@SESSION.wait_timeout = 2147483
=> #<User id: nil, name: nil, encrypted_password: nil, email: nil, neptun:
    nil, usertype: nil, created_at: nil, updated_at: nil, salt: nil>
irb(main):002:0> u.save
NameError: undefined local variable or method `u' for main:Object
        from (irb):2
irb(main):003:0> u = User.new
=> #<User id: nil, name: nil, encrypted_password: nil, email: nil, neptun:
    nil, usertype: nil, created_at: nil, updated_at: nil, salt: nil>
irb(main):004:0> u.save
   (0.2ms)  BEGIN
  User Exists (0.3ms)  SELECT  1 AS one FROM `users` WHERE `users`.`email`
    IS NULL LIMIT 1
   (0.1ms)  ROLLBACK
=> false
irb(main):005:0> u.errors
=> #<ActiveModel::Errors:0x007f2ff0017ee0 @base=#<User id: nil, name: nil,
    encrypted_password: nil, email: nil, neptun: nil, usertype: nil,
    created_at: nil, updated_at: nil, salt: nil>, @messages={:email=>["can't
     be blank"], :name=>["can't be blank"]}, @details={:email=>[{:error=>:
    blank}], :name=>[{:error=>:blank}]}>
irb(main):006:0> u.errors.messages
=> {:email=>["can't be blank"], :name=>["can't be blank"]}
irb(main):007:0> u.name = 'Valaki Mas'
=> "Valaki Mas"
irb(main):008:0> u.usertype = 0
```

```
=> 0
irb(main):009:0> u.email = 'senki@mail.bme.hu'
=> "senki@mail.bme.hu"
irb(main):010:0> u.save
   (0.1ms)  BEGIN
  User Exists (6.9ms)  SELECT  1 AS one FROM `users` WHERE `users`.`email` =
     BINARY 'senki@mail.bme.hu' LIMIT 1
   (0.3ms)  ROLLBACK
=> false
irb(main):011:0> u.errors.messages
=> {:email=>["has already been taken"]}
```

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

```
<h1>Registration</h1>
<%= flash[:notice] %>

  <% if @user.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@user.errors.count, "error") %> prohibited this user
           from being saved:</h2>

      <ul>
      <% @user.errors.full_messages.each do |message| %>
        <li><%= message %></li>
      <% end %>
      </ul>
    </div>
  <% end %>
```

Az enum a felsorolt típusok kezelését könnyíti meg.

```
kovacsg@debian:~/gyakorlat# rails c
Loading development environment (Rails 5.2.1)
irb(main):001:0> User.lecturer
   (0.2ms)  SET NAMES utf8,  @@SESSION.sql_mode = CONCAT(CONCAT(@@sql_mode,
       ',STRICT_ALL_TABLES'), ',NO_AUTO_VALUE_ON_ZERO'),  @@SESSION.
       sql_auto_is_null = 0, @@SESSION.wait_timeout = 2147483
  User Load (0.2ms)  SELECT  `users`.* FROM `users` WHERE `users`.`usertype`
       = 0 LIMIT 11
=> #<ActiveRecord::Relation [#<User id: 2, name: "Senki", encrypted_password
    : "efb935d3efbc456d8d27a218a4078408bf2888a5", email: "senki@mail.bme.hu
    ", neptun: nil, usertype: "lecturer", created_at: "2018-10-30 11:30:40",
     updated_at: "2018-10-30 11:30:40", salt: "Zf3gg8BdfWg=">, #<User id: 3,
     name: "Valaki Mas", encrypted_password: nil, email: "valakimas@mail.bme
    .hu", neptun: nil, usertype: "lecturer", created_at: "2018-10-30
    12:01:18", updated_at: "2018-10-30 12:01:18", salt: nil>]>
irb(main):002:0> User.student
  User Load (0.4ms)  SELECT  `users`.* FROM `users` WHERE `users`.`usertype`
       = 1 LIMIT 11
=> #<ActiveRecord::Relation []>
irb(main):003:0> User.lecturer.last
  User Load (0.4ms)  SELECT  `users`.* FROM `users` WHERE `users`.`usertype`
       = 0 ORDER BY `users`.`id` DESC LIMIT 1
=> #<User id: 3, name: "Valaki Mas", encrypted_password: nil, email: "
    valakimas@mail.bme.hu", neptun: nil, usertype: "lecturer", created_at:
    "2018-10-30 12:01:18", updated_at: "2018-10-30 12:01:18", salt: nil>
irb(main):003:0> User.first.lecturer!
  User Load (0.4ms)  SELECT  `users`.* FROM `users` ORDER BY `users`.`id`
      ASC LIMIT 1
```

```
   (0.3ms)  BEGIN
  User Exists (0.4ms)  SELECT  1 AS one FROM `users` WHERE `users`.`email` =
       BINARY 'senki@mail.bme.hu' AND `users`.`id` != 2 LIMIT 1
   (0.1ms)  COMMIT
=> true
irb(main):004:0> User.first
  User Load (0.2ms)  SELECT  `users`.* FROM `users` ORDER BY `users`.`id`
      ASC LIMIT 1
=> #<User id: 2, name: "Senki", encrypted_password: "
    efb935d3efbc456d8d27a218a4078408bf2888a5", email: "senki@mail.bme.hu",
    neptun: "bbbbbb", usertype: "lecturer", created_at: "2018-10-30
    11:30:40", updated_at: "2018-10-30 12:19:25", salt: "Zf3gg8BdfWg=">
irb(main):005:0> User.first.usertype
  User Load (0.2ms)  SELECT  `users`.* FROM `users` ORDER BY `users`.`id`
      ASC LIMIT 1
=> "lecturer"
```

Próbáljunk meg ezután felhasználót felvenni a webfelületen hibás adatokkal, és nézzük meg a hibaüzeneteket! Láthatjuk, hogy ezek a műveletek nem hajtódnak végre, és a Rails megjelöli a hibás beviteli mezőket. Ezután a validációs üzenet is megváltozik, amit konzolon ellenőrizhetünk.

A hibaüzenetet testreszabhatjuk a lokalizációs beállításokban:

```
  activerecord:
    errors:
      models:
        user:
          attributes:
            mail:
              blank: 'Empty email
              taken: 'Email already used'
```

Az előző gyakorlaton kezdeti adatokkal módosítottuk az események kontrollert. Most távolítsuk el azokat, és írjuk vissza az automatikusan generált kódrészleteket.

2. Az adatmodell kialakítása

2.1. Modellosztályok kapcsolatai

A feladatjavító portálunk adatmodellje a következő elemekből áll:

- Felhasználó (`User`), aki lehet hallgató vagy oktató
- Feladatsor (`Quiz`)
- Feladat (`Task`)
- Megoldás (`Solution`)
- Javítás (`Review`)

Ezek közül a megoldások kivételével a többit a korábbi gyakorlatokon már létrehoztuk. Pótoljuk ezt most be! A megoldás az valami, amit egy felhasználó egy feladatra ad. Egyelőre ennyi információt tárolunk róla, ez később még bővülhet.

```
kovacsg@debian:~/gyakorlat/db# rails g model solution user:references task:
    references
      invoke  active_record
      create    db/migrate/20181030122515_create_solutions.rb
      create    app/models/solution.rb
      invoke    test_unit
      create      test/models/solution_test.rb
      create      test/fixtures/solutions.yml
kovacsg@debian:~/gyakorlat/db$ rails db:migrate
== 20181030122515 CreateSolutions: migrating
   ==========================================
-- create_table(:solutions)
   -> 0.0230s
== 20181030122515 CreateSolutions: migrated (0.0232s)
   =================================
```

A modell két idegen kulcsot tartalmaz, ami két `belongs_to`-t jelent a modell osztályban.

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

Egy feladatsor (`Quiz`) több feladatból áll. A `references` típus a migrációban automatikusan létrehozza a `belongs_to` deklarációkat, adjuk hozzá a másik irányt is. Mindemellett egy feladatsort több hallgató is megold, ezt pedig onnan tudjuk, hogy a feladatsor egyes feladataira megoldást adtak be.

```
class Quiz < ApplicationRecord
  has_many :tasks
  has_many :user, through: :tasks
end
```

Egy feladat (`Task`) egy feladatsorhoz tartozik, ez a kapcsolat már létezik. Ezen kívül egy feladatra sok megoldás születhet a sok hallgató által. Az előbbi egy közvetlen kapcsolat a megoldások modellel, az utóbbi egy közvetett a megoldások modellen keresztül.

```
class Task < ApplicationRecord
  belongs_to :quiz
  has_many :solutions
  has_many :users, through: :solutions
end
```

A javítás (`Review`) tartozhat mind a megoldáshoz, ezt a hallgató típusú felhasználó végzi, mind a javításokhoz, ezt pedig az oktató típusú felhasználó végzi. Ezt egy polimorfikus kapcsolattal valósítjuk meg. Ehhez módosítanunk kell a javítás modellt, fel kell vennünk a polimorfikus kapcsolatot

lehetővé tevő mezőket. A `reviewable_id` attribútum tartalmazza vagy a megoldás, vagy a javítás azonosítóját, és azt, hogy a kettő közül melyiket, azt a `reviewable_type` attribútum mondja meg.

```
kovacsg@debian:~/gyakorlat/db$ rails g migration AddReviewableToReviews
    reviewable_id:integer reviewable_type:string
      invoke  active_record
      create    db/migrate/20181030122724_add_reviewable_to_reviews.rb
kovacsg@debian:~/gyakorlat/db/migrate$ rails db:migrate
== 20181030122724 AddReviewableToReviews: migrating
   ===========================
-- add_column(:reviews, :reviewable_id, :integer)
   -> 0.0271s
-- add_column(:reviews, :reviewable_type, :string)
   -> 0.0178s
== 20181030122724 AddReviewableToReviews: migrated (0.0453s)
   ==================
```

A javítás modellük ezután a következő lesz. Egy javítás egy javítandó objektumhoz kapcsolódik polimorfikus módon, és egyben több javítással rendelkezik kihasználva a polimorfikus kapcsolatot.

```
class Review < ApplicationRecord
  has_many :reviews, as: :reviewable
  belongs_to :reviewable, polymorphic: true
end
```

A megoldás osztályunkat is bővítenünk kell, ez is több javítandóval áll kapcsolatban.

```
class Solution < ApplicationRecord
  has_many :reviews, as: :reviewable
end
```

Végül a felhasználóknak több feladatsort, illetve feladatot kell megoldania.

```
  has_many :tasks
  has_many :quizes, through: :tasks
```

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 feladatokat, és feladatsorokat, majd konzolon vegyünk fel egy megoldás, és ellenőrizzük a kapcsolatokat.

```
for i in 1..10 do
    q = Quiz.create deadline: Time.now+i.weeks, number: i, title: "#{i}"
    for j in 1..5 do
       t = Task.create quiz: q, number: j, text: "#{j}", points: 1, solution
           : ''
```

```
    end
end
```

Töltsük be ezeket az adatokat:

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

Konzolon végezzünk el néhány műveletet: keressünk egy feladatot (2. sor), kérdezzük le az összes feladatsort (3. sor), kérdezzük le a legutolsó feladatsor adatait (4. sor), kérdezzük le a legutolsó feladatsor feladatait (5. sor). Hozzunk létre egy megoldást, amely erre a feladatra született az utoljára felvett felhasználó által (6. sor). Nézzünk mindjár meg ennek a feladatnak a megoldásait (7. sor), és azt, hogy kik adtak be rá megoldást (8. sor). Javítsuk ki ezt a megoldást (9-14. sor).

```
kovacsg@debian:~/gyakorlat# rails c
Loading development environment (Rails 5.2.1)
irb(main):001:0> Time.now + 1.weeks
=> 2018-11-06 13:34:55 +0100
irb(main):002:0> Task.take
   (0.2ms)  SET NAMES utf8,  @@SESSION.sql_mode = CONCAT(CONCAT(@@sql_mode,
       ',STRICT_ALL_TABLES'), ',NO_AUTO_VALUE_ON_ZERO'),  @@SESSION.
       sql_auto_is_null = 0, @@SESSION.wait_timeout = 2147483
  Task Load (0.2ms)  SELECT  `tasks`.* FROM `tasks` LIMIT 1
=> #<Task id: 1, quiz_id: 3, number: 1, text: "Hany eves vagy", points: 1,
    solution: "", created_at: "2018-10-09 10:56:21", updated_at: "2018-10-09
     10:56:21">
irb(main):003:0> Quiz.all
  Quiz Load (0.3ms)  SELECT  `quizzes`.* FROM `quizzes` LIMIT 11
=> #<ActiveRecord::Relation [#<Quiz id: 2, deadline: "2018-09-25", number:
    1, title: "", created_at: "2018-09-25 11:30:58", updated_at: "2018-09-25
     11:30:58">, #<Quiz id: 3, deadline: "2018-10-09", number: 1, title: "
    Elso zh", created_at: "2018-10-09 10:51:43", updated_at: "2018-10-09
    10:51:43">, #<Quiz id: 4, deadline: "2018-11-06", number: 1, title: "1",
     created_at: "2018-10-30 12:37:23", updated_at: "2018-10-30 12:37:23">,
    #<Quiz id: 5, deadline: "2018-11-06", number: 1, title: "1", created_at:
     "2018-10-30 12:38:04", updated_at: "2018-10-30 12:38:04">, #<Quiz id:
    6, deadline: "2018-11-13", number: 2, title: "2", created_at:
    "2018-10-30 12:38:04", updated_at: "2018-10-30 12:38:04">, #<Quiz id: 7,
     deadline: "2018-11-20", number: 3, title: "3", created_at: "2018-10-30
    12:38:04", updated_at: "2018-10-30 12:38:04">, #<Quiz id: 8, deadline:
    "2018-11-27", number: 4, title: "4", created_at: "2018-10-30 12:38:04",
    updated_at: "2018-10-30 12:38:04">, #<Quiz id: 9, deadline:
    "2018-12-04", number: 5, title: "5", created_at: "2018-10-30 12:38:04",
    updated_at: "2018-10-30 12:38:04">, #<Quiz id: 10, deadline:
    "2018-12-11", number: 6, title: "6", created_at: "2018-10-30 12:38:04",
    updated_at: "2018-10-30 12:38:04">, #<Quiz id: 11, deadline:
    "2018-12-18", number: 7, title: "7", created_at: "2018-10-30 12:38:04",
    updated_at: "2018-10-30 12:38:04">, ...]>
irb(main):004:0> Quiz.last
  Quiz Load (0.3ms)  SELECT  `quizzes`.* FROM `quizzes` ORDER BY `quizzes`.`
      id` DESC LIMIT 1
=> #<Quiz id: 14, deadline: "2019-01-08", number: 10, title: "10",
    created_at: "2018-10-30 12:38:04", updated_at: "2018-10-30 12:38:04">
irb(main):005:0> Quiz.last.tasks
  Quiz Load (0.2ms)  SELECT  `quizzes`.* FROM `quizzes` ORDER BY `quizzes`.`
      id` DESC LIMIT 1
  Task Load (0.4ms)  SELECT  `tasks`.* FROM `tasks` WHERE `tasks`.`quiz_id`
      = 14 LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy [#<Task id: 47, quiz_id:
    14, number: 1, text: "1", points: 1, solution: "", created_at:
    "2018-10-30 12:38:04", updated_at: "2018-10-30 12:38:04">, #<Task id:
    48, quiz_id: 14, number: 2, text: "2", points: 1, solution: "",
    created_at: "2018-10-30 12:38:04", updated_at: "2018-10-30 12:38:04">,
```

```
    #<Task id: 49, quiz_id: 14, number: 3, text: "3", points: 1, solution:
     "", created_at: "2018-10-30 12:38:04", updated_at: "2018-10-30
     12:38:04">, #<Task id: 50, quiz_id: 14, number: 4, text: "4", points: 1,
      solution: "", created_at: "2018-10-30 12:38:04", updated_at:
     "2018-10-30 12:38:04">, #<Task id: 51, quiz_id: 14, number: 5, text:
     "5", points: 1, solution: "", created_at: "2018-10-30 12:38:04",
     updated_at: "2018-10-30 12:38:04">]>
irb(main):006:0> Solution.create task_id: 51, user: User.last
  User Load (0.1ms)  SELECT  `users`.* FROM `users` ORDER BY `users`.`id`
      DESC LIMIT 1
   (0.1ms)  BEGIN
  Task Load (0.3ms)  SELECT  `tasks`.* FROM `tasks` WHERE `tasks`.`id` = 51
      LIMIT 1
  Solution Create (0.2ms)  INSERT INTO `solutions` (`user_id`, `task_id`, `
      created_at`, `updated_at`) VALUES (4, 51, '2018-10-30 12:39:10', '
      2018-10-30 12:39:10')
   (8.7ms)  COMMIT
=> #<Solution id: 1, user_id: 4, task_id: 51, created_at: "2018-10-30
    12:39:10", updated_at: "2018-10-30 12:39:10">
irb(main):007:0> Task.find(51).solutions
  Task Load (0.2ms)  SELECT  `tasks`.* FROM `tasks` WHERE `tasks`.`id` = 51
      LIMIT 1
  Solution Load (0.2ms)  SELECT  `solutions`.* FROM `solutions` WHERE `
      solutions`.`task_id` = 51 LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy [#<Solution id: 1, user_id:
     4, task_id: 51, created_at: "2018-10-30 12:39:10", updated_at:
    "2018-10-30 12:39:10">]>
irb(main):008:0> Task.find(51).users
   (0.2ms)  SET NAMES utf8,  @@SESSION.sql_mode = CONCAT(CONCAT(@@sql_mode,
       ',STRICT_ALL_TABLES'), ',NO_AUTO_VALUE_ON_ZERO'),  @@SESSION.
      sql_auto_is_null = 0, @@SESSION.wait_timeout = 2147483
  Task Load (0.1ms)  SELECT  `tasks`.* FROM `tasks` WHERE `tasks`.`id` = 51
      LIMIT 1
  User Load (0.4ms)  SELECT  `users`.* FROM `users` INNER JOIN `solutions`
      ON `users`.`id` = `solutions`.`user_id` WHERE `solutions`.`task_id` =
      51 LIMIT 11
=> #<ActiveRecord::Associations::CollectionProxy [#<User id: 4, name: "
    valaki", encrypted_password: "1255c85cb0ef5f33971548595436a739831d2479",
     email: "valaki@mail.bme.hu", neptun: "aaaaaa", usertype: "lecturer",
    created_at: "2018-10-30 12:21:40", updated_at: "2018-10-30 12:21:40",
    salt: "58SIIdtYl08=">]>
irb(main):009:0> r = Review.new
=> #<Review id: nil, mark: nil, comment: nil, created_at: nil, updated_at:
    nil, reviewable_id: nil, reviewable_type: nil>
irb(main):010:0> r.reviewable
=> nil
irb(main):011:0> r.reviewable = Solution.last
  Solution Load (0.2ms)  SELECT  `solutions`.* FROM `solutions` ORDER BY `
      solutions`.`id` DESC LIMIT 1
=> #<Solution id: 1, user_id: 4, task_id: 51, created_at: "2018-10-30
    12:39:10", updated_at: "2018-10-30 12:39:10">
irb(main):012:0> r
=> #<Review id: nil, mark: nil, comment: nil, created_at: nil, updated_at:
    nil, reviewable_id: 1, reviewable_type: "Solution">
irb(main):006:0> r.mark = 5
=> 5
irb(main):013:0> r.comment = "Remek"
=> "Remek"
irb(main):014:0> r.save
   (0.2ms)  BEGIN
  Review Create (0.2ms)  INSERT INTO `reviews` (`mark`, `comment`, `
      created_at`, `updated_at`, `reviewable_id`, `reviewable_type`) VALUES
```

```
        (5, 'Remek', '2018-10-30 12:43:21', '2018-10-30 12:43:21', 1, '
        Solution')
    (8.2ms)  COMMIT
=> true
```