Rails routing, kontrollerek
Gyakorlat

Kovács Gábor

2016. november 8.

A gyakorlatot az előző gyakorlat végén félbemaradt adatmodell véglegesítésével folytattuk. A kérdőív állapotának, vagyis hogy az szerkesztés alatt áll, véglegesített vagy már kitöltött és ezért lezárt, tárolása végett vegyünk fel egy egy digit hosszú státusz attribútumot a táblába. A kérdőív egy adott kérdése szövegének tárolására pedig elmulaszottunk felvenni egy szöveg típusú attribútumot, ezt most pótoljuk.

```
kovacsg@debian:~/gyakorlat/app/models> rails g migration
    AddStatusToQuestionnaire status:integer
       invoke  active_record
       create    db/migrate/20161108111740
           _add_status_to_questionnaire.rb
kovacsg@debian:~/gyakorlat/db/migrate> rake db:migrate
(in /home/kovacsg/gyakorlat)
== 20161108111740 AddStatusToQuestionnaire: migrating
   ============================
-- add_column(:questionnaires, :status, :integer, {:limit=>1})
   -> 0.2827s
== 20161108111740 AddStatusToQuestionnaire: migrated (0.2830s)
   ===================
kovacsg@debian:~/gyakorlat/db> rails g migration
    AddQuestionToQuestions question:text
       invoke  active_record
       create    db/migrate/20161108123313
           _add_question_to_questions.rb
kovacsg@debian:~/gyakorlat/db> rake db:migrate
(in /home/kovacsg/gyakorlat)
== 20161108123313 AddQuestionToQuestions: migrating
   ============================
-- add_column(:questions, :question, :text)
   -> 0.1327s
== 20161108123313 AddQuestionToQuestions: migrated (0.1332s)
   ===================
```

Az előző gyakorlaton láttuk, hogy az egy-több relációk több oldalán a `belongs_to` automatikusan generálódik, ha az attribútumot `references` típusúként deklaráltuk. A másik irányú navigáció megvalósítására az egy oldal modelljében fel kell vennünk a `has_many` hívásokat.

A kérdőív státusz attribútuma és a kérdés típus attribútuma egy digiten ábrázolható, véges értékkészletű listából vesz fel értéket. Az értékeket elnevezhetjük egy-egy Rails enum segítségével, amelynek egy hash-t adunk át, aminek a kulcsa a véges értékkészletű attribútum azonosítója, és értéke az értékek listája szimbólumokként. A kérdés modellben óvatlanul jártunk el, és ezt az attribútumot `type`-nak neveztük el, amely viszont foglalt attribútumnév Railsben, de ezt a foglaltságot a `inheritance_column` változó felüldefiniálásával feloldhatjuk, amit meg is teszünk.

```
class Questionnaire < ApplicationRecord
  belongs_to :user
  has_many :questions
  enum status: [ :edit, :final, :closed ]

end

class Question < ApplicationRecord
  belongs_to :questionnaire
  has_many :answers

  self.inheritance_column = :inheritance_type
  enum type: [ :text, :evaluate, :choice, :multichoice]
end
```

Nézzük meg az `enum`-ok működését! Az enumként definiált mezőkhöz szimbólum vagy string típusú értéket rendelhetünk. Minden egyes enum értékhez létrejött egy metódus, ami megvalósítja az adott státusszal rendelkező rekordok keresését.

```
kovacsg@debian:~/gyakorlat/app/models> rails c
Loading development environment (Rails 5.0.0.1)
irb(main):001:0> q = Questionnaire.first
  Questionnaire Load (0.4ms)  SELECT  `questionnaires`.* FROM `
      questionnaires` ORDER BY `questionnaires`.`id` ASC LIMIT 1
=> #<Questionnaire id: 1, title: "Ime, az elso", description: "Az
     elso kerdoivem hosszu-hosszu leirasa", user_id: 3,
    created_at: "2016-10-25 11:44:28", updated_at: "2016-10-25
    11:44:28", status: nil>
irb(main):002:0> q.status
=> nil
irb(main):003:0> q.status = :edit
=> :edit
irb(main):004:0> q.status
```

```
=> "edit"
irb(main):005:0> q.save
   (0.9ms)  BEGIN
  User Load (0.8ms)  SELECT  `users`.* FROM `users` WHERE `users
      `.`id` = 3 LIMIT 1
  SQL (0.5ms)  UPDATE `questionnaires` SET `updated_at` = '
      2016-11-08 11:23:23', `status` = 0 WHERE `questionnaires`.`
      id` = 1
   (46.2ms)  COMMIT
=> true
irb(main):006:0> q = Questionnaire.where(status: :edit)
  Questionnaire Load (1.1ms)  SELECT `questionnaires`.* FROM `
      questionnaires` WHERE `questionnaires`.`status` = 0
=> #<ActiveRecord::Relation [#<Questionnaire id: 1, title: "Ime,
    az elso", description: "Az elso kerdoivem hosszu-hosszu
    leirasa", user_id: 3, created_at: "2016-10-25 11:44:28",
    updated_at: "2016-11-08 11:23:23", status: "edit">]>
irb(main):007:0> Questionnaire.edit
  Questionnaire Load (0.8ms)  SELECT `questionnaires`.* FROM `
      questionnaires` WHERE `questionnaires`.`status` = 0
=> #<ActiveRecord::Relation [#<Questionnaire id: 1, title: "Ime,
    az elso", description: "Az elso kerdoivem hosszu-hosszu
    leirasa", user_id: 3, created_at: "2016-10-25 11:44:28",
    updated_at: "2016-11-08 11:23:23", status: "edit">]>
irb(main):008:0> Questionnaire.final
  Questionnaire Load (0.8ms)  SELECT `questionnaires`.* FROM `
      questionnaires` WHERE `questionnaires`.`status` = 1
=> #<ActiveRecord::Relation []>
irb(main):009:0> Questionnaire.closed
  Questionnaire Load (0.6ms)  SELECT `questionnaires`.* FROM `
      questionnaires` WHERE `questionnaires`.`status` = 2
=> #<ActiveRecord::Relation []>
```

A gyakorlat a felhasználói történetek megvalósításával folytatjuk. A felhasználó belépés után létrehezoz egy új kérdőívet, és kérdéseket ad hozzá ahhoz. A belépés után a saját menüjében látnia kell a saját kérdőíveit, amelyet lehetővé tevő link még nincs jelen a felhasználói menüben. Ha már a felhasználói menüben vagyunk, váltsuk ki az ott stringként bedrótozott útvonalakat a `routes.rb`-ben megadott helperekkel. Nevezzük el ezeket az útvonalakat a jobb karbantarthatóság végett!

```
  post 'sessions/create', to: 'sessions#create', as: 'login'

  get 'sessions/destroy', to: 'sessions#destroy', as: 'logout'
```

Módosítsuk ennek megfelelően a bejelentkezett felhasználó menüjét. A menüben már az elnevezett útvonalhoz létrejött helper metódussal hivatkozunk az útvonalra az aktuális felhasználóval paraméterezve.

```
<p>Hello, user!</p>
<%= link_to "My questionnaires", questionnaires_path %><br/>
<%= link_to "Profile", edit_user_path(session[:user]) %><br/>
<%= link_to "Logout", logout_path %>
```

A bejelentkezett felhasználó így átnavigálhat a kérdőívek listája nézetre, ahol azonban alapértelmezés szerint az összes kérdőív megjelenik, nem csak a sajátok. A saját kérdőívek lekérdezéséhez szükségünk van egy felhasználó objektumra, amit a `session`-ben eltárolt azonosító alapján kereshetünk elő. Ezt a keresést megvalósítottuk a felhasználók kontrollerében, de mivel úgy néz ki, hogy erre nem csak ott lesz szükség, átmozgatjuk egy szinttel feljebb az `application_controller`-be, és egyúttal a felhasználók kontrollerében definiált azonos viselkedést megszűntetjük.

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception
  before_action :find_user

  protected
    def find_user
      @user = User.find_by id: session[:user]
    end
end
```

A kérdőívek kontrollerében így már lekérdezhetjük a bejelentkezett felhasználó kérdőíveit, ha van olyan, ha nincs, akkor pedig az összeset.

```
class QuestionnairesController < ApplicationController
  def index
    if @user
      @questionnaires = @user.questionnaires
    else
      @questionnaires = Questionnaire.all
    end
  end
end
```

A kérdőívek sémát módosítottuk, ezért az automatikusan generált nézeteket is módosítanunk kell. Először is a kérdőív tulajdonosát email címmel jelenítjük meg az `index`

```
        <td><%= questionnaire.user.email %></td>
```

és a `show` nézeten,

```
<p>
  <strong>User:</strong>
  <%= @questionnaire.user.email %>
</p>
```

valamint a `new` és `edit` nézetek formjából eltávolítjuk annak szerkeszthetőségét. Ugyanakkor felvettünk egy státusz attribútumot, melyet beállíthatóvá kell tennünk a formban egy `select` helperrel, aminek első paramétere az attribútum neve, a második pedig címke és `value` HTML attribútum értékének szánt érték párokból alkotott tömbök listája (57. sor), melyekből HTML `option` elem lesz. A kérdőív státusz attribútum enum címkéiből állítjuk össze a listát (56. sor), a felületen megjelenő címke az érték nagybetűsített változata, a `value` pedig maga az érték lesz, amelyet stringként hozzá tudunk majd rendelni a `status` attribútumhoz minden módosítás nélkül (60. sor).

```
irb(main):054:0> Questionnaire.statuses
=> {"edit"=>0, "final"=>1, "closed"=>2}
irb(main):056:0> Questionnaire.statuses.keys
=> ["edit", "final", "closed"]
irb(main):057:0> Questionnaire.statuses.keys.map { |x| [x.
    titleize, x] }
=> [["Edit", "edit"], ["Final", "final"], ["Closed", "closed"]]
irb(main):058:0> q = Questionnaire.new
=> #<Questionnaire id: nil, title: nil, description: nil, user_id
    : nil, created_at: nil, updated_at: nil, status: nil>
irb(main):059:0> q.status = :edit
=> :edit
irb(main):060:0> q.status = 'edit'
=> "edit"
```

```
  <div class="field">
    <%= f.label :status %>
    <%= f.select :status, Questionnaire.statuses.keys.map { |x| [
        x.titleize, x] } %>
  </div>
```

Mivel egy felhasználónak nagyon sok kérdőíve lehet, és azok nem fognak feltétlenül elférni a maximum 400 pixel magasságban, amely a táblázatban a rendelkezésünkre áll, azokat több oldalra kell tördelnünk. Először vegyünk fel sok, véletlen kérdőívet több felhasználó számára. Az összes felhasznát listába gyűjtjük, és azok közül kiválaszunk egy véletlen értéket (26. sor), és annak azonosítóját hozzárendelhetünk az újonnal létrehozanó kérdőív tulajdonos attribútumához. A kérdőív címeként és szövegeként pedig a kérdőív sorszámát adjuk meg `ActiveSupport` helper felhasználásával angol szöveges sorszámmá alakítva (41.sor).

```
irb(main):026:0> User.all[rand(User.count)].id
   (0.8ms)  SELECT COUNT(*) FROM `users`
  User Load (0.1ms)  SELECT `users`.* FROM `users`
=> 2
irb(main):041:0> for i in 0..25 do Questionnaire.create title: "
    #{i.ordinalize} questionnaire", description: "The topic of #{
```

```
    i.ordinalize}␣questionnaire", user_id: User.all[rand(User.
    count)].id, status: :edit end
```

Megvan a sok kérdőívünk, amelyek nem férnek el egy oldalon. A tördeléshez segítségül hívjuk a Rails egyik API-ját, felveszünk egy új függőséget a `Gemfile`-ba:

```
gem 'will_paginate'
```

majd frissítjük a függőségeinket:

```
bundle update
```

Mindenekelőtt nézzük meg konzolon, hogyan működik az `ActiveRecord`-ok tördelése. A 48. sor visszaadja a `3`-as azonosítóval rendelkező felhasználó kérdőívet utolsó módosítás szerint időben csökkenő sorrendenben. Ezek közül van nekünk szükségünk egy töredékre, melyet a `paginate` függvény erre az eredményre való meghívásával érünk el. Két paramétere van, a `page`, amely a töredék sorszámát, és a `per_page`, amely a töredék méretét adja meg, ami legyen `5`.

```
irb(main):048:0> User.find(3).questionnaires.order(updated_at: :
    desc)
  User Load (0.7ms)  SELECT `users`.* FROM `users` WHERE `users
      `.`id` = 3 LIMIT 1
  Questionnaire Load (1.7ms)  SELECT `questionnaires`.* FROM `
      questionnaires` WHERE `questionnaires`.`user_id` = 3 ORDER
      BY `questionnaires`.`updated_at` DESC
...
irb(main):049:0> User.find(3).questionnaires.order(updated_at: :
    desc).paginate(page: 1, per_page: 5)
  User Load (0.3ms)  SELECT `users`.* FROM `users` WHERE `users
      `.`id` = 3 LIMIT 1
  Questionnaire Load (0.7ms)  SELECT `questionnaires`.* FROM `
      questionnaires` WHERE `questionnaires`.`user_id` = 3 ORDER
      BY `questionnaires`.`updated_at` DESC LIMIT 5 OFFSET 0
...
```

A tördelés megvalósítása négy lépésből áll. Először az kell kitalálnunk, hogy mi is az, amit tördelünk. Egy felhasználó kérdőíveit tördeljük, ami felhasználóra való keresést implikál, így a tördelést vagy a felhasználó modellben objektum szinten, vagy a kérdőív modellben osztály szinten felhasználó paraméterrel valósítjuk meg. Az előbbi minden tervezési elv alapján célravezetőbbnel tűnik. A felhasználó modell osztályban definiálunk egy függvény, ami az oldalszám paraméterhez előkeresi az adott oldalra eső töredéket a felhasználó összes, módosítási idő tekintetében csökkenő sorrendbe rendezett kérdőíve közül. Egy oldalon 5 kérdőívet engedélyezzünk.

```
  def get_questionnaires_page(page)
    self.questionnaires.order(updated_at: :desc).paginate(page:
        page, per_page: 5)
  end
```

A kontrollerben ezután nem a felhasználó összes kérdőívét kérjük el az adatbázisból, hanem csak az aktuális töredéket.

```
class UsersController < ApplicationController
  def index
    if @user
      @questionnaires = @user.get_questionnaires_page(params[:
          page])
    else
      @questionnaires = Questionnaire.all
    end
  endend
```

A nézethez pedig hozzáadjuk a lapozó linkeket. A linkek alapéertelmezés szerint a `page` paraméterhez rendelt értékben adják át a megjelenítendő töredés sorszámát.

```
    <% @questionnaires.each do |questionnaire| %>
        ...
    <% end %>
    <%= will_paginate @questionnaires %>
```

Negyedik lépésként a linkünk szebb kinézete végett módosítanunk kellene az útvonalat, hogy az opcionálisan tarlamazhassa a töredéket is, de a resourceful útvonalak kezeléséhez egyelőre még nincs meg a lexikális tudásunk, így az későbbre mara.

A felhasználó immáron létre tud hozni kérdőívet, megtekintheti a saját kérdőíve listáját. A következő lépés egy kérdés hozzáadása egy kérdőívhez. Kérdést a kérdőív szerkesztésekor adhatunk hozzá, így kézenfekvő, hogy a `edit` nézetet kell módosítanunk. Adjuk hozzá ahhoz a kérdőívhez tartozó összes kérdést táblázatosan a kérdések `index` nézetének mintája alapján, és tegyünk elérhetővé egy új kérdés létrehozását lehetővé tevő linket. E kettő teendőt egy csapással meg tudjuk valósítani, ha beágyazzuk a hivatkozott nézetet. Ehhez el kell készítenünk annak beágyazható változatát, `_` prefixszel. Az `edit` nézetünk csak ennyiben változik:

```
<%= render '/questions/index' %>

|
```

Ez a nézetek könyvárt `questions` alkönyvtárában `_index` nevű fájlt keres, és próbál beágyazni, amelyet az `index` másolásával hozunk létre.

Új kérdés létrehozásakor, illetve egy létező szerkesztésekor tudnunk kell, hogy az mely kérdőívhez tartozik. Mivel a felhasználó egyszerre csak egy kérdőívet tud szerkeszteni, a kontroller `edit` akciójában elmenjük annak azonosítóját a `session` hash-be.

```
def edit
  @questions = @questionnaire.questions
  session[:questionnaire] = @questionnaire.id
end
```

Új kérdés létrehozásakor pedig ezt az értéket elővesszük onnan.

```
def create
  @question = Question.new(question_params)
  @question.questionnaire_id = session[:questionnaire]
  ...
end
```

Az új kérdést létrehozó formban módosítanunk kell a típus attribútum beállításának módját. Ezt pont ugyanúgy tesszük meg, mint a kérdőív státuszának beállítását. Továbbá a formunkból hiányzik az utólag felvett, a kérdés szövegét tároló attribútum, annak értéke bekérése végett kiegészítjük a formot egy szövegmezővel.

```
<div class="field">
  <%= f.label :question %>
  <%= f.text_area :question, rows: 5, cols: 66 %>
</div>

<div class="field">
  <%= f.label :type %>
  <%= f.select :type, Question.types.keys.map { |x| [x.titleize
      , x] } %>
</div>
```

Végül az útvonalakat tartalmazó `config/routes.rb` fájlban állítsuk át a webszerverünk gyökerét a hello nézetre.

```
root :to => "say#hello"
```

A kérdések tördelése, megjelenítése, a kérdésekre adott válaszlehetőségek kérdésekbe ágyazása, tördelése és megjelenítése ugyanezen minták alapján mechanikusan elvégezhető, a gyakorlaton nem maradt rá időnk.