Rails routing, kontrollerek
Gyakorlat

Kovács Gábor

2020. április 21.

1. Útvonalak módosítása

A gyakorlat során az előző gyakorlatokon megkezdett feladat megoldását folytatjuk. Az előző gyakorlaton már átírtuk az útvonalakat a a `routes.rb`-ben, és menüben a felhasználóra vonatkozó útvonalak helyett már a létrehozott helpereket használjuk. Az útvonalakban így már felasználóazonosítók is szerepeljenek.

Az `ApplicationController` kontrollerben a felhasználót csak akkor kell előkeresnünk, ha be van jelentkezve, e feltételt utólag pótoljuk.

```
  def find_user
    @user = User.find session[:user] if session[:user]
  end
```

2. Tördelés

A következő dolgunk a témák nézet végleges változatának kialakítása. A feladatok listája (`tasks/index.html.erb`) nézeten megjelenő feladatok számára nincs felső korlát, ez azonban hamar betöltheti a rendelkezésre álló függőleges teret, ezért bevezetjük ezek több lapra való tördelését. Egy oldalon legyen öt feladat. A lapozás megvalósításához felveszünk egy új függőséget a `Gemfile`-ba:

```
gem 'will_paginate'
```

majd telepítjük a hiányzó függőségeinket:

```
bundle install
```

A tördelés megvalósítása ezután egyszerűvé válik. A `tasks/index.html.erb` nézeten felvesszük a lapozó linkeket az `@tasks` példányváltozóra.

```
<%= will_paginate @tasks %>
```

Ez `page` nevű paramétert generál minden kéréshez, amelyet átadhatunk a `paginate` függvénynek. Az összes feladat helyett keressük elő a paraméternek megfelelő tördedéket. A keresőművelet logikus helye a feladatsor modell osztályban lesz.

```
class TasksController < ApplicationController
  def index
    @tasks = Task.get_task_page(params[:page])
  end
end
```

A feladat modell osztályt kiegészítjük a töredéket lekérdező függvénnyel, melynek egy paramétere van, a töredék sorszáma. A feladatokat létrehozási idejük szerint csökkenő sorrendbe rendezzük, majd lekérdezzük a töredéket úgy, hogy egy töredék pontosan 5 feladatot tartalmazzon.

```
class Topic < ApplicationRecord
  def self.get_task_page(page)
    Task.all.order(created_at: :asc).paginate(page: page,
        per_page: 5)
  end
end
```

A `page` paraméter például `?page=2` formában jelenik meg a böngésző címsorában. Ezt Railsben nem szeretjük, ezért egy új útvonalat definiálunk a paraméter számára, ami ugyanúgy az `index` akcióra mutat, de tartalmazza a paramétert. Mivel ez nem egy konkrét téma példányra vonatkozik, az `on` opció értéke `collection` lesz (egyébként `member`-t használnánk).

```
  resources :tasks do
    get 'page/:page', on: :collection, to: 'tasks#index', as: '
        page'
  end
```

3. Fájlok fel- és letöltése

A témákhoz csatolhatunk fájlokat, amit csatolmány feltöltésével valósítunk meg. Léteznek erre kész API-k, mint a PaperClip vagy a CarrierWave, most azonban fapados megoldást adunk rá.

Először létrehozunk egy csatolmány modellt. Egy csatolmányról megjegyezzük a MIME-típusát, azt a megoldást, amelyhez kapcsolódik, a feltöltött

fájl elérési útvonalát a fájlrendszeren, az eredeti nevét és méretét. A megoldásokat a fájlrendszeren a `public` könyvtárban létrehozott `data` alkönyvtárban tároljuk, mert bináris adatot az adatbázis sem tud a fájlrendszernél hatékonyabban tárolni.

```
class CreateAttachments < ActiveRecord::Migration[6.0]
  def change
    create_table :attachments do |t|
      t.references :solution, null: false, foreign_key: true
      t.string :name
      t.string :mime
      t.string :path
      t.integer :size

      t.timestamps
    end
  end
end
```

Majd végrehajtjuk a migrációt.

```
kovacsg@debian:~/gyakorlat# rails g model attachment solution:
    references name:string mime:string path:string size:integer
      invoke  active_record
      create    db/migrate/20200420070231_create_attachments.rb
      create    app/models/attachment.rb
      invoke    test_unit
      create      test/models/attachment_test.rb
      create      test/fixtures/attachments.yml
```

A csatolmány egy megoldáshoz tartozik, de egy megoldáshoz több csatolmány is tartozhat, több változatot is elfogadunk, de csak az utolsót vesszük figyelembe. Ezért módosítjuk a megoldások modellünket:

```
class Solution < ApplicationRecord
  has_many :attachments

  def attachment
    self.attachments.order(created_at: :desc).first
  end
end
```

A csatolmány fel- és letöltésének eseménykezelőjét egy új kontrollerrel valósítjuk meg, amelyben két akció lesz a feltöltés (`create`) és a letöltés (`download`). A nézeteket töröljük, ezekhez nem kell, hogy önálló nézet tartozzék.

```
kovacsg@debian:~/gyakorlat/app# rails g controller solutions
    upload download
```

```
      create  app/controllers/solutions_controller.rb
       route  get 'solutions/upload'
get 'solutions/download'
      invoke  erb
      create    app/views/solutions
      create    app/views/solutions/upload.html.erb
      create    app/views/solutions/download.html.erb
      invoke  test_unit
      create    test/controllers/solutions_controller_test.rb
      invoke  helper
      create    app/helpers/solutions_helper.rb
      invoke    test_unit
      invoke  assets
      invoke    scss
      create      app/assets/stylesheets/solutions.scss
```

Módosítjuk a fájlfeltöltő és -letöltő útvonalakat! A fájlfeltöltésnek két paramétere lesz: a feltöltő felhasználó azonosítója, a feladat azonosítója; a letöltésnek pedig egy, a csatolmány azonosítója.

```
Rails.application.routes.draw do
  post 'solutions/upload/:user/:task', to: 'solutions#upload', as
      : 'upload'
  get 'solutions/download/:attachment', to: 'solutions#download',
       as: 'download'
end
```

Módosítsuk a feladatot beadó űrlapunkat (`tasks/show.html.erb`), hogy az képes legyen a fájlfeltöltésre. Ehhez fel kell vennünk a `multipart` opciót a paraméterek közé, és az útvonal paraméternek át kell adunk az aktuális felhasználó és feladat azonosítóit, amelyeket a kontrollerben már előkerestünk.

```
<% if is_student? %>
  <%= form_tag upload_path(@user, @task), multipart: true, method
      : :post do %>
    <%= label_tag 'file', 'Solution' %>
    <%= file_field_tag 'file' %>
    <%= submit_tag "Send␣solution" %>
  <% end %>
<% end %>
```

Hozzuk létre a feltöltés eseménykezelőjét a csatolmányok kontrollerben. A megoldás objektum létrehozásához szükségünk van a témára és a felhasználóra, melyek idegen kulcsok a modellben, azokat a paraméterül átadott azonosító alapján előkereshetjük az adatbázisból. Ha létezik ezzel a párossal megoldás, akkor előkeressük, ellenkező esetben létrehozzuk. A feltöltött fájlra a a nézetben megadott `upload` paraméternévvel hivatkozhatunk. Ha e paraméter rendelkezik értékkel, akkor mentsük el. Önálló nézetünk nincs,

ezért a következő oldalunk az előző oldal újratöltése lesz, ami egy feladat adatlapja.

```
class SolutionsController < ApplicationController
  def upload
    unless params[:file].nil?
      solution = Solution.find_or_create_by user: @user, task_id:
          params[:task]
      Attachment.save_file params[:file], solution
    end
    redirect_back(fallback_location: task_path(params[:task]))
  end
end
```

A csatolmány elmentésének logikus helye a csatolmány modell, ahol definiáljuk a fent hivatkozott osztálymetódust. Annak törzsében először definiáljuk a feltöltött állományokat tartalmazó könyvtárt, és létrehozzuk, ha nem létezne. Ezután létrehozunk egy új csatolmány objektumot, aminek beállíthatjuk a fájlnevét, a MIME típusát, és a megoldást, amelyhez tartozik, ez utóbbit most hozzuk létre, hiszen most szüketett. A fájlt még nem mentettük el, ezért sem a méretét, sem az elérési útját nem tudjuk még, ennek ellenére létrehozzuk a csatolmány objektumot. A következő lépés a fájl elmentése a kijelölt könyvtárba. Az elmentett fájl elérési útjával és méretével frissítjük a csatolmány objektumunkat.

```
class Attachment < ApplicationRecord
  belongs_to :solution

  def self.save_file(upload, solution)
    return if upload.nil?
    dir = Rails.root.join('public', 'data')
    unless File.exists? dir
      Dir.mkdir dir
    end
    fname = upload.original_filename
    a = Attachment.create name: fname, mime: upload.content_type,
        solution: solution
    path = File.join(dir, a.id.to_s)
    File.open(path, 'wb') do |f| f.write(upload.read) end
    a.update path: path, size: File.size(path)
  end
end
```

A téma adatainak megjelenítésekor a téma előkeresésén túl szükségünk van a feladathoz kapcsolódó összes csatolmányra is. Ha oktatóról van szó, akkor a feladat megoldásainak utolsó verzióira minden egyes hallgatótól vagyunk kíváncsiak, ha hallgatók, akkor pedig a feladatra beadott megoldások

listájára.

```
class TasksController < ApplicationController
  def show
    if @user.student?
      @attachments = @user.solutions.where(task: @task).first.
          attachments
    else
      @attachments = @task.solutions.collect &:attachment
    end
  end
end
```

A csatolmányokat a `tasks/show.html.erb` fájban kell megjelenítenünk. Ehhez szükség van a letöltés műveletre. A beágyazást HTML objektum által valósítjuk meg, amelyhez szükségünk van egy URL-re. Az URL a letöltés útvonalunk adja, amelynek egy paramétere van, a csatolmány azonosítója. Az oktatónak megjelenítjük a feltöltő nevét, valamint a fájl nevét, a hallgató csak a fájl nevét látja.

```
<p>
  <strong>Solutions:</strong>
  <ul>
    <% @attachments.each do |attachment| %>
      <% unless attachment.nil? %>
        <li><%= link_to "#{attachment.solution.user.name unless
            is_student?} #{attachment.name}", download_path(
            attachment.id) %></li>
      <% end %>
    <% end %>
  </ul>
</p>
```

A letöltés akciót a megoldások kontrollerbe helyezzük el. Az akció a paraméterük kapott azonosító alapján előkeresi a csatolmányt az adatbázisból, és annak útvonal attribútuma által kijelölt fájlt visszaküldi közben jelzi a böngészőnek, hogy próbálja beágyazva megjeleníteni a MIME típusa alapján. Ha nincs ilyen azonosító, akkor 404-es hibakóddal térünk vissza.

```
class SolutionsController < ApplicationController
  def download
    a = Attachment.find(params[:attachment])
    unless a.nil?
      send_file a.path, type: a.mime, disposition: :attachment,
          filename: a.name
      return
    end
    head 404
  end
```

```
end
```

4. Felhasználótípusok és a menüik

Az oktató és a hallgató felhasználótípusok megkülönböztetése végett felveszünk egy új mezőt a felhasználók modellbe, a szerepet.

```
kovacsg@debian:~/gyakorlat# rails g migration AddRoleToUsers role
    :integer{1}
      invoke  active_record
      create    db/migrate/20200420061133_add_role_to_users.rb
```

A szerep alapértelmezett értéke legyen a hallgató, amelyet az 1 érték reprezentál.

```
class AddRoleToUsers < ActiveRecord::Migration[6.0]
  def change
    add_column :users, :role, :integer, limit: 1, null: false,
        default: 1
  end
end
```

Végrehajtjuk a migrációt.

```
kovacsg@debian:~/gyakorlat/db/migrate# rails db:migrate
== 20200420061133 AddRoleToUsers: migrating
   ===========================================
-- add_column(:users, :role, :integer, {:limit=>1, :null=>false,
    :default=>1})
   -> 0.0323s
== 20200420061133 AddRoleToUsers: migrated (0.0327s)
   ==========================
```

A modellben definiáljuk a szerep lehetséges értékeit egy enummal.

```
class User < ApplicationRecord
  enum role: { lecturer: 0, student: 1 }
end
```

Az egyik felhasználónkat konzolon kinevezzük oktatónak (22. sor).

```
irb(main):021:0> User.last
  User Load (1.8ms)  SELECT `users`.* FROM `users` ORDER BY `
      users`.`id` DESC LIMIT 1
=> #<User id: 3, name: "admin", neptun: "ADMIN0", email: "
    admin@mail.bme.hu", encrypted_password: [FILTERED],
    created_at: "2020-04-03 11:12:36", updated_at: "2020-04-03
    11:12:36", salt: "0s5I396XxC6cMWhMEu1tCg==", role: "student">
irb(main):022:0> User.last.lecturer!
```

```
  User Load (0.5ms)  SELECT `users`.* FROM `users` ORDER BY `
      users`.`id` DESC LIMIT 1
DEPRECATION WARNING: Uniqueness validator will no longer enforce
    case sensitive comparison in Rails 6.1. To continue case
    sensitive comparison on the :email attribute in User model,
    pass `case_sensitive: true` option explicitly to the
    uniqueness validator. (called from irb_binding at (irb):22)
   (0.1ms)  BEGIN
  User Exists? (0.3ms)  SELECT 1 AS one FROM `users` WHERE `users
      `.`email` = BINARY 'admin@mail.bme.hu' AND `users`.`id` !=
      3 LIMIT 1
  User Update (0.5ms)  UPDATE `users` SET `users`.`role` = 0, `
      users`.`updated_at` = '2020-04-20 07:56:24.654085' WHERE `
      users`.`id` = 3
   (1.6ms)  COMMIT
=> true
```

A két gyakorlattal ezelőtt létrehozott helper metódusunkat most már ki tudjuk javítani.

```
module ApplicationHelper
  def is_student?
    logged_in? && @user.student?
  end
end
```

A feladatok `index` és `show` nézetén ezután felhasználótípushoz tudjuk rendelni az egyes funkciókat. A feladat szerkesztése és törlése csakis oktatók számára lesz elérhető az `index` oldalon, az új feladat létrehozása ugyancsak a `show` oldalon. A feladat beadása viszont csak a hallgatók számára érhető el a `show` oldalon.

Módosítjuk a hallgató típusú felhasználó menüjét. Először a linkekhez tartozó útvonalakat módosítjuk. A második, harmadik és negyedik linkek ugyanoda mutattak, tegyünk most ezek között különbséget. A második link a megoldandó feladatok listája mara. A harmadik is egy lista, de az azon feladatokat jeleníti meg, amelyekre a hallgató megoldást adott be. A negyedik is egy lista, de az azon feladatokat jeleníti meg, amelyeket a hallgatónak véleményeznie kell. Így szükségünk lesz két új, a feladatokhoz kapcsolódó útvonalra.

```
<p>Hello, <%= @user.name %>!</p>
<p><%= flash[:notice] %></p>
<%= link_to "Profile", edit_profile_path(@user.id) %><br/>
<%= link_to "Tasks", tasks_path %><br/>
<%= link_to "My solutions", my_tasks_path %><br/>
<%= link_to "Solutions to correct", review_tasks_path %><br/>
<%= link_to "Logout", logout_path %>
```

Mindkét új útvonal feladatok egy halmazára vonatkozik, ezért ezek is `collection` értéket kapnak az `on` paraméterhez, és egy-egy új kontroller akcióra képeződnek le. Mindkét új útvonalnak tördelhetőnek kell lennie, mert az `index` nézeten tördelést végzünk.

```
Rails.application.routes.draw do
 resources :tasks do
   get 'page/:page', on: :collection, to: 'tasks#index', as: '
       page'
   get 'my(/:page)', on: :collection, to: 'tasks#my', as: 'my'
   get 'reviews(/:page)', on: :collection, to: 'tasks#reviews',
       as: 'review'
 end
```

Az új kontroller akcióink, előkeresik a saját feladathalmazaikat. A saját megoldások (`my` akció) a felhasználó feladatait az előző órán létrehozott kapcsolaton keresztül, a javítandó megoldások (`reviews` akció) pedig egy most létrehozandó kapcsolaton keresztül. Mivel egyik akció sem rendelkezik önálló nézetsablonnal, mindkét esetben az index nézetet használjuk megjelenítésre a render függvény felhasználásával.

```
class TasksController < ApplicationController
 def my
   @tasks = @user.tasks.paginate(page: params[:page], per_page:
       5)
   render :index
 end

 def reviews
   @tasks = @user.reviewables.paginate(page: params[:page],
       per_page: 5)
   render :index
 end
end
```

A javító egy új kapcsolat a megoldások és a felhasználók között, ezért módosítjuk a megoldások modellt, majd végrehajtjuk a migrációt.

```
kovacsg@debian:~/gyakorlat# rails g migration
    AddReviewerToSolutions reviewer:references
      invoke  active_record
      create    db/migrate/20200420084525
          _add_reviewer_to_solutions.rb
kovacsg@debian:~/gyakorlat/db/migrate# rails db:migrate
== 20200420084525 AddReviewerToSolutions: migrating
   ============================
-- add_reference(:solutions, :reviewer, {:foreign_key=>true})
```

A megoldás modellben jeleznünk kell, hogy új relációnk van, és az a felhasználók modellre mutat. A `belongs_to` paramétereként megnevezzük azt az osztály amely példányaira e ráláció hivatkozik (`class_name: 'User'`), és azt, hogy melyik az az idegen kulcs attribútum az ehhez az osztályhoz tartozó táblában (`foreign_key: 'reviewer_id'`).

```
class Solution < ApplicationRecord
  belongs_to :user
  belongs_to :reviewer, class_name: 'User', foreign_key: '
      reviewer_id'
end
```

A felhasználók modellben jelenzünk kell, hogy a javítandók különböznek a megoldásoktól (`reviewables`), fent, a kontrollerben ezt a relációt használtuk. Mivel a reláció neve eltér a hivatkozott modell nevétől, itt is használnunk kell a `class_name: 'Solution'` és a `foreign_key: 'reviewer_id'` opciókat a kapcsolat egyértelműsítése végett.

```
class User < ApplicationRecord
  has_many :solutions
  has_many :reviewables, class_name: 'Solution', foreign_key: '
      reviewer_id'
end
```

Az utolsó kérdés az, hogy a javítandók halmaza mikor áll elő egy feladat vagy egy hallgató típusú felhasználó esetén. Kezdetben, amikor egy megoldás létrejön, javítóként rendeljünk hozzá egy oktató típusú felhasználót, ha még nem lett volna beállítva.

```
class SolutionsController < ApplicationController
  def upload
    unless params[:file].nil?
      solution = Solution.find_or_create_by user: @user, task_id:
          params[:task]
      if solution.reviewer.nil?
        solution.update reviewer: User.lecturer.take
      end
      Attachment.save_file params[:file], solution
    end
    redirect_back(fallback_location: task_path(params[:task]))
  end
```

Az oktatóknak egy-egy feladat esetére pedig tegyük azt lehetővé, hogy ahhoz egy random hallgatót rendeljenek hozzá javítóként. A funkciót a `show` nézetben helyezzük el, és tesszük elérhetővé egy új linkkel.

```
<%= link_to 'Shuffle', shuffle_task_path(@task) unless is_student
    ? %>
```

A művelethez egy új útvonalra van szükségünk, amely azonban egy konkrét, azonosítóval rendelkező feladatra értelmezett, így az `on` opció értéke most `member` lesz.

```
Rails.application.routes.draw do
  resources :tasks do
    get 'page/:page', on: :collection, to: 'tasks#index', as: '
        page'
    get 'my(/:page)', on: :collection, to: 'tasks#my', as: 'my'
    get 'reviews(/:page)', on: :collection, to: 'tasks#reviews',
        as: 'review'
    get 'shuffle', on: :member, to: 'tasks#shuffle', as: 'shuffle
        '
  end
end
```

Az új útvonal a `shuffle` akcióra hivatkozik a feladatok kontrollerében, azt létre kell hoznunk. Az akció kigyűjti az adatbázisból az összes hallgató típusú felhasználó azonosítóját, majd létrehoz egy új tömböt, amely a hozzárendeleléseket tartalmazza, és a kezdeti értéke a tömbünk elemeinek random permutációja. Eztán a tömbünk minden eleme tekintetében megnézzük, hogy az elem indexe megegyezik-e a régi és az új tömbben, ha igen, akkor további permutációt végzünk az új tömbben. Végül az aktuális feladat minden egyes megoldása esetére beállítjuk az felhasználó azonosítójának eredeti tömbbeli pozíciójában lévő elemet az új tömbből, és hozzárendeli a megoldáshoz azt javítóként.

```
class TasksController < ApplicationController
  before_action :set_task, only: [:show, :edit, :update, :destroy
      , :shuffle]

  def shuffle
    ar = User.student.collect &:id
    new = ar.shuffle
    ar.each_with_index do |item, index| if new[index] == item
        then new[index], new[index-1] = new[index-1], new[index]
        end end
    @task.solutions.each do |solution|
      solution.update(reviewer_id: new[ar.index(solution.user_id)
          ])
    end
  end
end
```