Rails routing, kontrollerek
Gyakorlat

Kovács Gábor

2022. május 3.

1. Útvonalak módosítása

A gyakorlat során az előző gyakorlatokon megkezdett feladat megoldását folytatjuk. A felhasználókra vonatkozó útvonalaink azonban még nem az egyes felhasználókhoz kötődnek, hanem minden felhasznároła közösek. Először ezt módosítsuk. A felhasználói profil megjelenítése és a felhasználói profil módosítása útvonalakbe felveszünk egy-egy `id` paramétert, amely meg fog jelenni a `params` hashben.

```
Rails.application.routes.draw do
  get 'users/edit/:id', to: 'users#edit', as: 'profile'
  put 'users/update/:id', to: 'users#update', as: 'update_profile
      '
end
```

A belejentkezett felhasználó menüjének nézetében (`app/views/layouts/_user_menu.html.erb`) még paraméter nélkül hívjuk meg az `as` opció utáni útvonal helpert, ezért azt is módosítjuk. A `@user` példányváltozót az `ApplicationController` `find_user` függvényében inicializáltuk, így felhasználhatjuk, az `id` attribútum értékét hozzárendeljük az útvonal `:id` paraméteréhez.

```
<p><%= link_to "Profile", profile_path(@user.id) %></p>
```

A profil szerkesztése képernyőn (`app/views/users/edit.html.erb`) pedig a form eseménykezelője rossz helyre mutat, adjuk hozzá a hiányzó paramétert.

```
<%= form_for @user, url: update_profile_path(@user.id), method: :
    put do |form| %>
```

Az erőforrás alapú útvonalakat a `resources` függvény generálja, mégpedig hetet. Az erőforrás nevével megegyező útvonalhoz HTTP GET, illetve HTTP POST művelettel férhetünk hozzá. Az előbbi az erőforrás összes adatbázisbeli rekordját adja vissza egy listában, tömbben, az utóbbi pedig az erőforrás típusából egy új példányt hoz létre. Ha ehhez az útvonalhoz egy azonosítót adunk hozzá, akkor az erőforrás típusának az azonosító által azonosított példányán végezhetünk el műveleteket. A GET megmutatja az erőforrást, a DELETE törli az erőforrást, a PUT módosítja az erőforrást egy form adataival. Azt a képernyőt, ahol a módosítás elérhető szintén a GET művelettel érhetjük el, viszont ez ütközik az erőforrás megmutatásával, ezért hozzáteszünk az útvonalhoz egy `edit` szuffixet. Ehhez hasonlóan ütközés van, amikor azt a képernyőt akarjuk betölteni, ahol az erőforrás típusából új példányt létrehozó form van, az az összes erőforrás listájával üzközik, ezért ott egy `new` szuffixet adunk hozzá.

```
resources :parties # only: [] vagy #except: []
# get 'parties', to: 'parties#index', as: 'parties'
# get 'parties/:id', to: 'parties#show', as: 'party'
# delete 'parties/:id', to: 'parties#destroy'
# get 'parties/new', to: 'parties#new', as: 'new_party'
# post 'parties', to: 'parties#create'
# put 'parties/:id', to: 'parties#update', as: 'update_party'
# get 'parties/:id/edit, to: 'parties#edit', as: 'edit_party'
```

2. Lapozás

A következő dolgunk a témák nézet végleges változatának kialakítása. A pártok listája képernyőn (`parties/index.html.erb`) megjelenő pártok számára nincs felső korlát, ez azonban hamar betöltheti a rendelkezésre álló függőleges teret, ezért bevezetjük ezek több lapra való tördelését. Egy oldalon legyen legfeljebb két párt. 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 `parties/index.html.erb` nézeten felvesszük a lapozó linkeket a `@parties` példányváltozóra.

```
<%= will_paginate @parties %>
```

A `@parties` példányváltozót eddig az `index` metódusban állítottuk be, melynek törzsében a `page` paraméterhez tartozó töredéket vesszük elő az aktuális pártok közül. Az effektív tördelést a párt modell osztályában végezzük el egy osztálymetódusban.

```
class PartiesController < ApplicationController
  # GET /parties or /parties.json
  def index
    #@parties = Party.all
    @parties = Party.get_party_page(params[:page])
  end
```

A párt modell osztályt kiegészítjük a töredéket lekérdező osztálymetódussal, melynek egy paramétere van, a töredék sorszáma. A pártok töredékét úgy kérdezzük le, hogy egy töredék pontosan 2 elemet tartalmazzon.

```
class Party < ApplicationRecord
  def Party.get_party_page(page)
    Party.all.order(created_at: :asc).paginate(page: page,
        per_page: 2)
  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 `:page` paramétert. Mivel ez nem egy konkrét párt példányra vonatkozik, az `on` opció értéke `collection` lesz, így az `:id` paraméter nem lesz része az útvonalnak (egyébként `member`-t használnánk).

```
Rails.application.routes.draw do
  resources :parties do # only: [] vagy #except: []
    get 'page/:page', on: :collection, to: 'parties#index', as: '
        page'
  end
end
```

3. Fájlok fel- és letöltése

Minden pártnak van egy emblémája, amely egy kép, és amelyet fel, illetve le kell tudnunk tölteni. Léteznek erre kész API-k, mint a PaperClip, a CarrierWave vagy az ActiveStorage, most azonban fapados megoldást adunk rá.

A modellünket nevezzük `Logo`-nak. Egy logóról megjegyezzük a MIME-típusát, azt a pártot, amelyhez kapcsolódik, a feltöltött fájl elérési útvonalát

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

```
kovacsg@debian:~/gyakorlat/config> rails g model logo name:string
    size:integer mime:string path:string party:references
      invoke  active_record
      create    db/migrate/20220503104847_create_logos.rb
      create    app/models/logo.rb
      invoke    test_unit
      create      test/models/logo_test.rb
      create      test/fixtures/logos.yml
kovacsg@debian:~/gyakorlat/app/models> rails db:migrate
== 20220503104847 CreateLogos: migrating ======================
-- create_table(:logos)
   -> 0.0176s
== 20220503104847 CreateLogos: migrated (0.0177s) =============
```

A logó egy-egy kapcsolatban áll a párttal, a `Logo` modellben a kapcsolat automatikusan létrejött, a párt modelljéhez azonban hozzá kell adnunk.

```
class Party < ApplicationRecord
  has_one :logo
end
```

A logó fel- és letöltésének eseménykezelőjét a párt kontrollerrel valósítjuk meg, amelyben három akciót értint a feltöltés (`create` és `update`) és a letöltés (`logo`).

Módosítjuk a most pártok erőforrásának útvonalait fájlfeltöltő és -letöltő útvonalakat! Mivel a feltöltéshez a létrehozás, illetve a módosítás műveleteket használjuk, ahhoz nincs szükségünk új útvonalra, viszont a letöltéshez igen. A letöltésnél az `id` paraméter annak a pártnak az azonosítóját jelenti, amelyhez új logót rendelünk a csatolmány által, ezért a `on` opció értéke `member` lesz ebben az esetben.

```
Rails.application.routes.draw do
  resources :parties do # only: [] vagy #except: []
    get 'page/:page', on: :collection, to: 'parties#index', as: '
        page'
    get 'logo', on: :member, to: 'parties#logo', as: 'logo'
  end
```

A feltöltés helye a pártot létrehozó, illetve módosító képernyők formja, amely szerencsére egyetlen, azonos fájlban található, az `app/views/parties/_form.html.erb`-ben, amelyet kiegészítünk egy új mezővel A formot biná-

ris adatok átküldésére is alkalmassá kell tennünk. Ehhez fel kell vennünk a `multipart` opciót a paraméterek közé.

```
<%= form_with(model: party, multipart: true) do |form| %>
  ...
  <div>
    <%= form.label :logo %>
    <%= form.file_field :logo %>
  </div>
  ...
<% end %>
```

Hozzuk létre a feltöltés eseménykezelőjét a pártok kontrollerében. A logót akkor tudjuk hozzárendelni a párthoz, ha a párt már létezik az adatbázisban, ezért a `party_params` függvényben a `params` hasht fehérlistázó sort nem kell módosítanunk. Ha a form adataival sikeresen létrehoztuk vagy módosítottuk a pártot, akkor a feltöltött fájl objektumot elmentjük a `Logo` modell osztály egy osztálymetódusával.

```
class PartiesController < ApplicationController
  # POST /parties or /parties.json
  def create
    @party = Party.new(party_params)

    respond_to do |format|
      if @party.save
        Logo.save_file params[:party][:logo], @party
        format.html { redirect_to party_url(@party), notice: "
            Party was successfully created." }
        format.json { render :show, status: :created, location:
            @party }
      else
        format.html { render :new, status: :unprocessable_entity
            }
        format.json { render json: @party.errors, status: :
            unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /parties/1 or /parties/1.json
  def update
    respond_to do |format|
      if @party.update(party_params)
        if params[:party][:logo]
          Logo.save_file params[:party][:logo], @party
        end
        format.html { redirect_to party_url(@party), notice: "
            Party was successfully updated." }
```

```
        format.json { render :show, status: :ok, location: @party
            }
      else
        format.html { render :edit, status: :unprocessable_entity
            }
        format.json { render json: @party.errors, status: :
            unprocessable_entity }
      end
    end
  end
end
```

A logó elmentésének logikus helye a logó modell, mert több helyen használjuk, és így csak egy helyen kell karbantartanunk. Itt definiáljuk a fent hivatkozott osztálymetódust. Annak törzsében először definiáljuk a feltöltött logókat tartalmazó könyvtárt, és létrehozzuk, ha nem létezne. Ezután létrehozuk egy új logó objektumot, amelynek beállíthatjuk a fájlnevét, a MIME típusát, és a pártját amelyhez tartozik. 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 Logo < ApplicationRecord
  belongs_to :party

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

A csatolmányokat az `image` HTML tagek `src` attribútumában használjuk például a `show` nézetben, amely a `_party.html.erb` töredéket rendeleli.

```
<div id="<%= dom_id party %>">
  ...
  <div class="logo">
```

```
    <img src="<%= logo_party_path(@party) %>" alt="Logo not found
        ">
  </div>
</div>
```

A letöltés akciót a pártok kontrollerbe helyezzük el. A logó egy párt példányhoz tartozik, ezért a `before_action` listáját bővítenünk kell az új függvényünkkel, hogy ez előtt is meghívódjék a `set_party` függvény, amely beállítja az `:id` paraméter alapján a `@party` példányváltozót. A letöltéshez előkeressük a `@party` példányváltozó által reprezentált párthoz asszociált logó objektumot, ha van ilyen, és a `send_file` művelettel visszaküldjük a felhasználónak. Ha nincs ilyen azonosítóval csatolmány, akkor az erőforrás nem létezik hibát adunk vissza.

```
class AttachmentsController < ApplicationController
  before_action :set_party, only: %i[ show edit update destroy
     logo ]
  def logo
    l = @party.logo
    unless l.nil?
      send_file l.path, type: l.mime, disposition: :inline,
          filename: l.name

      return
    end
    head 404
  end
end
```

4. További nézetek és eseménykezelők hozzáadása

A szavazóportálunkon már csak szavazni nem lehet. Hogy lehessen, fel kell egy szavazat modellt. A szavazatot egy felhasználó adja le egy szavazókörzetben egy listára és egy jelöltre. Tehát a szavazat modellünk négy idegen kulcsból fog állni. A felhasználó és a körzet kötelező, a másik kettő viszont felvehet `nil` értéket, ha valaki még nem szavazott, vagy nem akart szavazni egyik opcióra sem.

```
kovacsg@debian:~/gyakorlat/app/models> rails g model vote user:
    references district:references list:references candidate:
    references
      invoke  active_record
      create    db/migrate/20220503112041_create_votes.rb
```

```
      create    app/models/vote.rb
      invoke    test_unit
      create      test/models/vote_test.rb
      create      test/fixtures/votes.yml
kovacsg@debian:~/gyakorlat/app/views/votes> rails db:migrate
== 20220503112041 CreateVotes: migrating ======================================
-- create_table(:votes)
   -> 0.0389s
== 20220503112041 CreateVotes: migrated (0.0391s) =============================
```

A felhasználók és a szavazókörök összerendelését az irányítószám alapján végezzük el, ezért egy felhasználó és egy szavazókör címét beállítjuk konzolon. A 25. sorban létrehozunk egy új cím objektumot, amelyet a 27. sorban hozzárendelünk lakcímként egy felhasználóhoz. A felhasználó objektum 28. sorban való mentése elmenti a cím objektumot is. A 29. sorban létrehozunk egy másik cím objektumot, amelyet a 30. sorban létrehozott körzet objektumhoz rendelünk a 31. sorban. Itt is a körzet objektum mentése maga után vonja a cím objektum mentését. Mivel a cím objektum polimorfikus, az `a` objektum `addr_type` paraméterének értéke `User` lesz, és az `addr_id` attribútumának értéke a birtokló felhasználó azonosítója (39. sor). Az `a2` objektum esetén pedig az `addr_type` paraméterének értéke `District` lesz, és az `addr_id` attribútumának értéke a birtokló körzet azonosítója (40. sor).

```
irb(main):025:0> a = Address.new city: 'Budapest', zip: 1000,
    street: 'Magyar tudosok utja 2', lat: 47, lng: 19
irb(main):026:0> u = User.first
irb(main):027:0> u.address = a
irb(main):028:0> u.save
irb(main):029:0> a2 = Address.new city: 'Budapest', zip: 1000,
    street: 'Magyar tudosok utja 2', lat: 47, lng: 19
irb(main):030:0> d = District.new name: 'Muegyetem'
irb(main):031:0> d.address = a2
irb(main):032:0> d.save
irb(main):039:0> a
=>
#<Address:0x00007f049cb05e80
 id: 1,
 city: "Budapest",
 street: "Magyar tudosok utja 2",
 zip: 1000,
 lat: 47.0,
 lng: 19.0,
 addr_type: "User",
 addr_id: 3,
 created_at: Tue, 03 May 2022 11:26:54.272710000 UTC +00:00,
```

```
 updated_at: Tue, 03 May 2022 11:26:54.272710000 UTC +00:00>
irb(main):040:0> a2
=>
#<Address:0x00007f049cc64d58
 id: 2,
 city: nil,
 street: "Magyar tudosok utja 2",
 zip: 1000,
 lat: 47.0,
 lng: 19.0,
 addr_type: "District",
 addr_id: 1,
 created_at: Tue, 03 May 2022 11:28:57.091486000 UTC +00:00,
 updated_at: Tue, 03 May 2022 11:28:57.091486000 UTC +00:00>
```

Ha arra vagyunk kíváncsiak, hogy egy felhasználó a lakcímének irányítószáma alapján melyik körzetbe tartozik, akkor azt a 35. sorban lévő kóddal tudhatjuk lekérdezni. Mivel a cím, amely alapján a keresést végezzük nem a körzet modellben van, hanem egy kapcsolt modellben, a `joins` függvényt használjuk a keresőfeltétel megadása előtt. A keresőfeltételben pedig tudatnunk kell a Rails-szel, hogy a feltétel a kapcsolt objektum egy attribútumára hivatkozik, és az értéke a felhasználó objektumhoz kapcsolt cím objektum irányítószám attribútuma. Ez egy egyelemű tömböt ad vissza, amelyből nekünk egy objektum kell, ezért meghívjuk a `first` függvényt az eredményre.

```
irb(main):035:0> District.joins(:address).where('address.zip': u.
    address.zip).first
  District Load (0.4ms)  SELECT `districts`.* FROM `districts`
      INNER JOIN `addresses` `address` ON `address`.`addr_type` =
        'District' AND `address`.`addr_id` = `districts`.`id`
      WHERE `address`.`zip` = 1000 ORDER BY `districts`.`id` ASC
      LIMIT 1
=>
#<District:0x000055ec5af66fe8
 id: 1,
 name: "Muegyetem",
 created_at: Tue, 03 May 2022 11:28:57.082747000 UTC +00:00,
 updated_at: Tue, 03 May 2022 11:28:57.082747000 UTC +00:00>
```

Most már minden a rendelkezésünkre áll a szavazáshoz, ezért hozzuk létre a szavazás kontrollerét és nézetét. Egy képernyőnk lesz a szavazás űrlapja, és két műveletünk az űrlapot betöltő és az azt feldolgozó akció.

```
kovacsg@debian:~/gyakorlat/app/models> rails g controller votes
    vote
      create  app/controllers/votes_controller.rb
       route  get 'votes/vote'
      invoke  erb
```

```
      create    app/views/votes
      create    app/views/votes/vote.html.erb
      invoke  test_unit
      create    test/controllers/votes_controller_test.rb
      invoke  helper
      create    app/helpers/votes_helper.rb
      invoke    test_unit
```

Módosítsuk az automatikusan létrehozott útvonalat, és hozzuk létre az eseménykezelő útvonalát.

```
Rails.application.routes.draw do
  get 'votes/vote', to: 'votes#vote', as: 'vote'
  post 'votes/do', to: 'votes#submit', as: 'submit'
```

A szavazáshoz szükségünk van a szavazat négy idegen kulcsára. A felhasználó megvan a sessionből. A körzetet a fenti minta alapján keressük elő a felhasználóhoz. A jelölteket nem soroltuk még körzetekben, ezért nem szűrünk rájuk, a felhasználó az összes közül választhat, akárcsak az összes pártlista közül választhat.

```
class VotesController < ApplicationController
  def vote
    @district = District.joins(:address).where('address.zip':
        @user.address.zip).first
    @candidates = Candidate.all
    @lists = List.all
  end

  def submit
  end
end
```

A szavazólapunk első változatát a `vote.html.erb` fájlban hozzuk létre, ahol rádiógombokkal választhat a felhasználó a pártlisták, illetve a jelöltek közül a formban.

```
<h1>Votes</h1>
<p>Dear <%= @user.name %>, please vote in the <%= @district.name
    %> district</p>

<%= form_tag submit_path, method: :post do %>
  <div>
  <% for list in @lists do %>
    <%= label_tag list.name %>
    <%= radio_button_tag :list, list.id %>
  <% end %>
  </div>
```

```
  <div>
  <% for candidate in @candidates do %>
    <%= label_tag candidate.name %>
    <%= radio_button_tag :candidate, candidate.id %>
  <% end %>
  </div>

  <%= submit_tag 'Vote' %>
<% end %>
```