Rails routing, kontrollerek
Gyakorlat

Kovács Gábor

2018. április 17.

A gyakorlat során az előző gyakorlatokon megkezdett feladat megoldását folytatjuk. A harmadik gyakorlaton kialakított menüben a felhasználóra vonatkozó útvonalak közösek az összes felhasználóra, azokat testre kell szabnunk az egyes felhasználók tekintetében, vagyis a felasználóazonosítóknak szerepelniük kell a linkekben, ezért módosítjuk a `routes.rb`-ben a megfelelő bejegyzéseket. Az átalakítást úgy tesszük meg, hogy az hasonlítson a `resource` függvénnyel generált útvonalakra. Az értintett linkek a felhasználói profil megtekintése, a profil szerkesztése nézetek, illetve az utóbbihoz tartozó eseménykezelők. Nevezzük el ezeket az útvonalakat a jobb karbantarthatóság végett.

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

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

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

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

  get 'users/forgotten'
  post 'users/send_forgotten'
  get 'download/:id/:aid', to: 'users#download', as: 'download'

  resources :events
  get 'say/hello', as: :welcome
end
```

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

```
<%= link_to "View profile", profile_path(@current_user.id) %><br
    />
<%= link_to "Edit profile", edit_profile_path(@current_user.id)
    %><br/>

<%= link_to "Logout", logout_path %>
</fieldset>
```

Egyúttal módosítjuk az üdvözlő üzenetet is, hogy az a felhasználót a nevén szólítsa. Az aktuálisan bejelentkezett felhasználót a kontrollerek ősosztályában állítjuk be, hiszen arra a megjelenítendő oldaltól, vagyis az elvégzendő műveletektől függetlenül szükségünk van.

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

  protected
    def find_session
      @current_user = User.find session[:user]
    end
end
```

Ezt a példányváltozót hivatkozhatjuk meg a nézetben.

```
Welcome, <%= @current_user.name %>!
```

Előzetesen csak egy linket definiáltuk, ahol az események megtekinthetők, azonban azokat több szempontból is elkülöníthetjük: a felhasználó által rendezett események, a korábban látogatott események, jövőbeli események, és az összes esemény. Ezekre hozzunk létre külön linkeket a menüben. A `resource` helper ezeket az útvonalakat nem generálja, ezért azt előbb ki kell egészítenünk. Mivel mindegyik több esemény listájára vonatkozik, ezért ezek `collection` típusú útvonalak lesznek, és nem fog bennük szerepelni azonosító.

```
Rails.application.routes.draw do
  resources :events do
    get 'past', on: :collection
    get 'forthcoming', on: :collection
    get 'my', on: :collection
  end
end
```

A menü nézetben felhasználandó útvonal helpereket a konzolon kiadott `rake routes` paranccsal nézhetjük meg, így az a következőképp alakul:

```
<%= link_to "View my events", my_events_path %><br/>
<%= link_to "View past events", past_events_path %><br/>
<%= link_to "View forthcoming events", forthcoming_events_path
    %><br/>
<%= link_to "View all events", events_path %><br/>
```

A három útvonal három új metódus definícióját teszi szükségessé az események kontrollerében, azonban mindhárom ugyanazt a sablont, az `index` nézet sablonját, használja az adatok megjelenítésére, viszont másként szűrt adatokat használ. A múltbeli események határidejének értéke kisebb a mai napnál, a jövőbelieknél ez legalább a mai nap, a saját események pedig az aktuálisan bejelentkezett felhasználó tulajdonont események asszociációján (lásd `User` modell `has many` deklarációját) keresztül érhetők el.

```
class EventsController < ApplicationController
  def past
    @events = @current_user.events.where('deadline < ?', Date.
        today)
    render action: :index
  end

  def forthcoming
    @events = @current_user.events.where('deadline >= ?', Date.
        today)
    render action: :index
  end

  def my
    @events = @current_user.owned_events
    render action: :index
  end
end
```

Tegyük lehetővé az eseményre való meghívást. Ehhez is egy új útvonalat kell definiálnunk a `events` erőforrásban. Mivel meghívni valakit egy konkrét eseményre lehet, ezért abban szerepelnie kell az esemény azonosítójának, ezért ez egy `member` útvonal lesz.

```
Rails.application.routes.draw do
  resources :events do
    get 'past', on: :collection
    get 'forthcoming', on: :collection
    get 'my', on: :collection
    post 'invite', on: :member
  end
```

```
end
```

Az összes felhasználó közül hívhatunk meg valamit, ezért módosítsuk ennek megfelelően a nézetünkben a kódot, továbbá használjuk az új útvonal helperünket az aktuális eseménnyel paraméterezve.

```
<p>
  <strong>Invite user:</strong>
  <%= form_tag invite_event_path(@event), method: :post do %>
    <%= select_tag 'user', options_for_select(User.all.map do |x|
        [x.name, x.id] end) %>
    <%= submit_tag 'Invite' %>
  <% end %>
</p>
```

Definiáljuk a meghívás eseménykezelőjét. Az eseményt az azonosítója alapján elő kell keresnünk, ezért módosítjuk a `before_filter` hívást hozzáadva az új függvény azonosítóját. A kérés paramétereként értezik a meghívandó felhasználó azonosítója, amelyhez tartozó felhasználó előkeressük az adatbázisból, hozzárendeljük az esemény résztvevőinek tömbjéhez, ha még nem lett volna annak eleme, és frissítjük a `show` nézetünket.

```
class EventsController < ApplicationController
  before_action :set_event, only: [:show, :edit, :update, :
      destroy, :invite]
  def invite
    u = User.find params[:user]
    @event.users << u unless @event.users.include? u
    render action: :show
  end
end
```

Mivel eseményből akár végtelenül sok lehet, az oldalunk vertikális dimenziója viszont véges, tördeljük az üzeneteket lapozhatóan több oldalra. Ehhez felveszünk egy új függőséget a `Gemfile`-ba:

```
gem 'will_paginate'
```

majd telepítjük a hiányzó függőségeinket:

```
bundle install
```

A tördelés megvalósítása ezután egyszerűvé válik. Az `index` nézeten felvesszük a lapozó linkeket az `@events` példányváltozóra, amely négy kontroller akcióból is származhat.

```
<%= will_paginate @events %>
```

Ez `page` nevű paramétert generál minden kéréshez, amelyet átadhatunk a `paginate` függvénynek. Az egy oldalon megjelenítendő események száma legyen 10. Mivel mind a négy kontroller akcióban ugyanazt a módosítást kellene elvégeznünk, a jobb karbantarthatóság végett definiálunk egy privát függvényt.

```
class EventsController < ApplicationController
  def index
    @events = paginate(Event.all, params[:page])
  end

  def past
    @events = paginate(@current_user.events.where('deadline < ?',
        Date.today), params[:page])
    render action: :index
  end

  def forthcoming
    @events = paginate(@current_user.events.where('deadline >= ?'
        , Date.today), params[:page])
    render action: :index
  end

  def my
    @events = paginate(@current_user.owned_events, params[:page])
    render action: :index
  end

  private
    def paginate(events, page)
      events.paginate(page: page, per_page: 10)
    end
end
```

Tegyük lehetővé, hogy a felhasználó feltölthessen pontosan egy profilképet a saját adatlapjára. Ehhez fájlfeltöltést valósítunk meg, amelyhez szükségünk lesz egy új modellre, ami a feltöltött képekről tárol adatot. Egy csatolmányról megjegyezzük a MIME-típusát, azt az felhasználót, akihez kapcsolódik, a feltöltött fájl elérési útvonalát a fájlrendszeren, az eredeti nevét és méretét. A képeket a fájlrendszeren a `public` könyvtárban létrehozott `data` alkönyvtárban tároljuk, mert bináris adatot az adatbázis sem tud a fájlrendszernél hatékonyabban tárolni.

```
kovacsg@debian:~/gyakorlat/test/models# rails g model attachment
    filename:string mime:string user:references path:string size:
    integer
      invoke  active_record
      create    db/migrate/20180417111331_create_attachments.rb
```

```
      create    app/models/attachment.rb
      invoke    test_unit
      create      test/models/attachment_test.rb
      create      test/fixtures/attachments.yml
```

Hajtsuk végre a migrációt, majd indítsuk újra a webszerverünket.

```
kovacsg@debian:~/gyakorlat/test/models# rails db:migrate
== 20180417111331 CreateAttachments: migrating ================================
-- create_table(:attachments)
   -> 0.0183s
== 20180417111331 CreateAttachments: migrated (0.0185s) =======================
```

Egy felhasználóhoz egy kép tartozhat, ezért felvesszük ezt az egy-egy relációt a két modellbe. A tulajdonos a felhasználó modell.

```
class User < ApplicationRecord
  has_one :attachment
end
```

A birtokolt objektum a csatolmány. Itt a kapcsolat a `references` típus a migrációban való használata miatt már eleve létezik.

```
class Attachment < ApplicationRecord
  belongs_to :user
end
```

A fájlfelöltéshez módosítanunk kell a felhasználót létrehozó (`new.html.erb`), illetve az adatlapot szerkesztő nézet formjait (`edit.html.erb`), hogy az a szöveges mellett bináris adatok átküldésére is alkalmas legyen. Ehhez HTML opcióként megadjuk, hogy az elküldendő üzenet törzse `multipart` MIME típussal legyen tördelve a böngésző által. Továbbá felveszünk a formba egy fájlkiválasztó beviteli mezőt.

```
<%= form_for :user, html: {multipart: true}, url: { controller: '
    users', action: 'update' }, method: :put do |form| %>
  ...
    <div>
      <%= form.label :attachment %>
      <%= form.file_field :attachment %>
    </div>
  ...
<% end %>
```

Ezek az eseménykezelője a `create`, illetve az `update` metódus a `users` kontrollerben. Először a kérés paraméterei közül fehérlistáznunk kell az `attachment` kulcsot is, de mivel az nem a felhasználó közvetlen tuljadonsága,

a `params` hash-t nem adhatjuk át a `create`, illetve az `update` metódusnak, ezért a mentést két lépésben valósítjuk meg. Először a közvetlen attribútumokat frissítjük, majd, ha az sikeres volt, elmentjük a csatolt képet, ha az lézetik.

```
class UsersController < ApplicationController
  def create
    @user = User.new(name: user_params[:name], password:
        user_params[:password], email: user_params[:email],
        password_confirmation: user_params[:password_confirmatio
n])
    if @user.save
      unless params[:user][:attachment].nil?
        Attachment.save_file params[:user][:attachment], @user
      end
      session[:user] = @user.id
      flash[:notice] = "Successful registration"
      redirect_to welcome_path
    else
      flash[:notice] = "Email address in use or password do not
          match"
      redirect_back(fallback_location: register_path)
    end
  end

  def update
    @user.update(name: user_params[:name], password: user_params
        [:password], email: user_params[:email],
        password_confirmation: user_params[:password_confirmation
        ])
    unless params[:user][:attachment].nil?
      Attachment.save_file params[:user][:attachment], @user
    end
    redirect_to action: :show
  end
end
```

A fájl mentését a csatolmány modell egy osztálymetódusában valósítjuk meg, aminek két paramétere lesz, a feltöltött temporális fájl és az asszociált felhasználó. Először definiáljuk a fájl mentésének a helyét, és ellenőrizzük, hogy létezik-e az, ha nem, akkor létrehozzuk a könyvtárat. Létrehozunk egy új csatolmány példányt, és inicializáljuk annak mezőit a feltöltött temporális fájl attribútumai alapján. Eltároljuk az adatokat az adatbázisban, majd elmentjük a fájlt az adatbázisazonosítóval megegyező névvel a kijelölt helyre. Végül lekérdezzük a létrejött fájl méretét és elérési útját, és elmentjük azokat is az adatbázisba.

```
class Attachment < ApplicationRecord
```

```
  def self.save_file(upload, user)
    dir = Rails.root.join('public', 'data')
    unless File.exists? dir
      Dir.mkdir dir
    end
    fname = upload.original_filename
    # fname.gsub!(/^|w|.|_/,"_")
    a = Attachment.new
    a.filename = fname
    a.mime = upload.content_type
    a.user = user
    a.save
    path = File.join(dir, a.id.to_s)
    File.open(path, 'wb') do |f| f.write(upload.read) end
    a.size = File.size(path)
    a.path = path
    a.save
  end
end
```

A felhasználó profilképét a `show.html.erb` fájlban kell megjelenítenünk, ha a felhasználónak van feltöltött képe. A fájl letöltését egy újonnan felveendő, `download` nevű útvonalon keresztül tesszük lehetővé felhasználva a helpert. Két paraméterre mindenképp szükség lesz, a felhasználóra és a csatolmány azonosítójára (erre csak akkor, ha egy felhasználónak több csatolmánya is lehet).

```
<div>
  <% unless @user.attachment.nil? %>
    <img width="200px" height="200px" src="<%= download_path(
        @user.id, @user.attachment.id) %>"></img>
  <% end %>
</div>
```

A kép URL-jének definiálnunk kell egy új útvonalat:

```
get 'download/:id/:aid', to: 'users#download', as: 'download'
```

Ennek az eseménykezelője legyen az `users` kontrollerben `download` néven definiálandó metódus. Mivel ez egy konkrét felhasználóra vonatkozik, a felhasználó objektumát előkerestető `before_filter`-t itt is módosítjuk. Maga a letöltő akció az adatbázisból előkeresi a kérés paraméterének megfelelő csatolmányt, és visszaküldi a fájrendszerről az alapján előkeresett fájlt válaszként.

```
class UsersController < ApplicationController
  before_action :find_user, only: [:edit, :update, :show, :
      download]
```

```
  def download
    a = Attachment.find params[:aid]
    send_file a.path, type: a.mime, disposition: :inline
  end
end
```

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

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