Rails routing, kontrollerek
Gyakorlat

Kovács Gábor

2020. november 10.

1. Útvonalak módosítása

A gyakorlat során az előző gyakorlatokon megkezdett feladat megoldását folytatjuk. A `routes.rb`-ben a felhasználók kontroller akcióira mutató útvonalakat tesszük először rendbe. Az útvonalak közül több egy konkrét felhasználóra vonatkozik, ezért az útvonalban meg kell jelennie a felhasználó azonosítójának. Ilyen az adatlap, a profil szerkesztése nézet

```
Rails.application.routes.draw do
  get 'users/new', to: 'users#new', as: 'registration'
  match 'users/create', to: 'users#create', as: 'register', via:
      [:post, :get]
  get 'users/edit/:id', to: 'users#edit', as: 'profile'
  put 'users/update/:id', to: 'users#update', as: 'update_profile
      '
  get 'users/forgotten', to: 'users#forgotten', as: '
      forgotten_password'
  put 'users/recover', to: 'users#recover', as: 'recover'
  get 'users/show/:id', to: 'users#show', as: 'show_user'
end
```

Minden útvonalhoz rendelünk az `as` opcióval egy (pontosabban kettő) egyedi útvonalhelpert (`_url` és `_path` szuffixekkel), amelyre az összes nézetben lecseréljük az első paraméterek sztringjeit. Ahol az útvonalsztringben szimbólum áll, ott a helper annyi paraméterrel rendelkezik, ahány szimbólum van az útvonalban. A bejelentkezett felhasználó menüje így néz ki például:

```
<p>Hello <%= @user.name %>!</p>
<p><%= flash[:notice] %></p>
<%= link_to "Profile", profile_path(@user.id) %><br/>
<%= link_to "My videos", videos_path %><br/>
<%= link_to "Logout", logout_path %>
```

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 `@videos` példányváltozóra.

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

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

```
class VideosController < ApplicationController
  def index
    @videos = Video.get_videos_page(params[:page])
  end
end
```

A videó modell osztályt kiegészítjük a töredéket lekérdező függvénnyel, melynek egy paramétere van, a töredék sorszáma. A videókat létrehozási idejük szerint csökkenő sorrendbe rendezzük, majd lekérdezzük a töredéket úgy, hogy egy töredék pontosan 2 elemet tartalmazzon.

```
class Video < ApplicationRecord
  def self.get_videos_page(page)
    Video.all.order(created_at: :desc).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 paramétert. Mivel ez nem egy konkrét téma példányra vonatkozik, az `on` opció értéke `collection` lesz (egyébként `member`-t használnánk).

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

3. Fájlok fel- és letöltése

A videókhoz csatolnunk kell 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 :video, 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, a gyakorlaton ezt két részletben tettük meg, mert elsőre lehagytuk a `mime` paramétert.

```
kovacsg@debian:~/gyakorlat/db/migrate$ rails db:migrate
== 20201110115054 CreateAttachments: migrating
   ===============================
-- create_table(:attachments)
   -> 0.0301s
== 20201110115054 CreateAttachments: migrated (0.0306s)
   ======================

kovacsg@debian:~/gyakorlat/db/migrate$
kovacsg@debian:~/gyakorlat/db/migrate$ rails g migration
    AddMimeToAttachments mime:string
```

```
Running via Spring preloader in process 18051
      invoke  active_record
      create    db/migrate/20201110120544_add_mime_to_attachments
          .rb
kovacsg@debian:~/gyakorlat/db/migrate$ rails db:migrate
== 20201110120544 AddMimeToAttachments: migrating
   ==============================
-- add_column(:attachments, :mime, :string)
   -> 0.0171s
== 20201110120544 AddMimeToAttachments: migrated (0.0174s)
   =====================
```

A csatolmány egy videó tartozik, ezért módosítjuk a megoldások modellünket. A másik irányú kapcsolat a `references` típusú attribútum miatt automatikusan létrejött.

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

A csatolmány fel- és letöltésének eseménykezelőjét a videó kontrollerrel valósítjuk meg, amelyben két akciót értint a feltöltés (`create`) és a letöltés (`download`).

Módosítjuk a fájlfeltöltő és -letöltő útvonalakat! A letöltés a videók alá tartozik, egy konkrét videópéldányhoz rendelt (`on: :member`), és az újonnan bevezetendő `download` függvény szolgálja ki.

```
Rails.application.routes.draw do
  resources :videos do
    get 'page/:page', on: :collection, to: 'videos#index', as: '
        page'
    get 'download', on: :member, to: 'videos#download', as: '
        download'
  end
end
```

A videó létrehozása űrlapunkon már ott van a fájlfeltöltés (`videos/new.html.erb`), viszont a formot bináris adatok átküldésére is alkalmassá kell tennünk. Ehhez fel kell vennünk a `multipart` opciót a paraméterek közé, a videó azonosítója a modellből elérhető. A felhasználó állíthatóságára nincs szükségünk, ezért azt eltávolítjuk a formból, így csak a cím, a fájl és a tagek beviteli mezők maradnak.

```
<%= form_with(model: video, local: true, html: {multipart: true})
    do |form| %>
  ...
<% end %>
```

Hozzuk létre a feltöltés eseménykezelőjét a videók kontrollerben, ott is a létrehozás akcióban. A videó objektum létrehozásához csak a címre van szükségünk. A videó tulajdonosa az aktuálisan bejelentkezett felhasználó. Ha a videót sikeresen el tudtuk menteni az adatbázisba, akkor a fájl paraméter alapján létre tudjuk hozni a videó példányhoz tartozó csatolmány objektumot, és a videót hozzá tudjuk rendelni a felhasználóhoz.

```
class SolutionsController < ApplicationController
  def create
    @video = Video.new title: params[:video][:title]
    @video.user = @user

    respond_to do |format|
      if @video.save
        attachment = Attachment.save_file params[:video][:file],
            @video
        @user.videos << @video
        format.html { redirect_to @video, notice: 'Video was
            successfully created.' }
        format.json { render :show, status: :created, location:
            @video }
      else
        format.html { render :new }
        format.json { render json: @video.errors, status: :
            unprocessable_entity }
      end
    end
  end

  private
    # Only allow a list of trusted parameters through.
    def video_params
      params.require(:video).permit(:title, :file)
    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 létrehozuk egy új csatolmány objektumot, aminek beállíthatjuk a fájlnevét, a MIME típusát, és a videót, 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.

```ruby
class Attachment < ApplicationRecord
  belongs_to :video

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

A videókat a `video` HTML tagek `src` attribútumában használjuk az `index` és a `show` nézetekben. A videó azonosítója az `index` nézetben az iteráció objektumából, a `show` nézeten pedig a példányváltozóból jön.

```
<video width="250" controls class="video" id="video_<%= video.id
    %>">
   <source src="<%= download_video_path(video.id) %>" type="video
       /mp4" />
</video>
```

A letöltés akciót a videók kontrollerbe helyezzük el. A videó példányváltozó beállítására a letöltéshez is szükségünk van, ezért bővítjük a `before_action` listáját. A letöltéshez előkeressük e videó csatolmányát, és a `send_file` művelettel visszaküldjük a felhasználónak.

```ruby
class VideosController < ApplicationController
  before_action :set_video, only: [:show, :edit, :update, :
      destroy, :download]
  def download
    a = @video.attachment
    unless a.nil?
      send_file a.path, type: a.mime, filename: a.filename,
          disposition: :inline
    end
  end
end
```