Rails routing, kontrollerek
Gyakorlat

Kovács Gábor

2021. április 27.

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: 'register'
  post 'users/create'
  get 'users/edit/:id', to: 'users#edit', as: 'edit_profile'
  put 'users/update/:id', to: 'users#update', as: 'update_profile
      '
  get 'users/show/:id', to: 'users#show', as: 'profile'
  get 'users/forgotten'
  post 'users/send_forgotten'
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 projects", items_path %><br/>
<%= link_to "Logout", logout_path %>
```

2. Tördelés

A következő dolgunk a témák nézet végleges változatának kialakítása. A projektek listája (`items/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.

Először is vegyünk fel ehhez elég projektet, de még azelőtt a projektek kontrollerében (`ItemsController`) állítsuk vissza az automatikusan generált állapotot, amely az adatbázisból olvassa az adatokat.

```
class ItemsController < ApplicationController
  def index
    @items = Item.get_items_page(params[:page])
  end

  private
    def set_item
      @item = Item.find(params[:id])
    end
end
```

A projekteket létrehozó kódrészletet átmásoljuk a `db/seeds.rb` fájlba a reprodukálhatóság végett. A tulajdonos minden esetben egy, az adatbázisban létező felhasználó.

```
kovacsg@debian:~/gyakorlat> rails c
irb(main):011:0> for i in 1..10 do
irb(main):012:1* Item.create name: "Project #{i}", description: "
    Project #{i}", user_id: 2
irb(main):013:1> end
```

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 `items/index.html.erb` nézeten felvesszük a lapozó linkeket az `@items` példányváltozóra.

```
<%= will_paginate @items %>
```

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

```
class ItemsController < ApplicationController
  def index
    @items = Item.get_items_page(params[:page])
  end
end
```

A projekt modell osztályát kiegészítjük a töredéket lekérdező függvénnyel, melynek egy paramétere van, a töredék sorszáma. A projekteket létrehozási idejük szerint csökkenő sorrendbe rendezzük, majd lekérdezzük a töredéket úgy, hogy egy töredék pontosan 5 elemet tartalmazzon.

```
class Item < ApplicationRecord
  def Item.get_items_page(page)
    Item.all.order(created_at: :desc).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).

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

3. Fájlok fel- és letöltése

A projektek erőforrásaihoz csatolmányokat, vagyis fizikailag létrejövő fájlokat rendelünk, amit csatolmány feltöltésével valósítunk meg. Léteznek erre kész API-k, mint a PaperClip, a CarrierWave vagy az ActiveStorage, most azonban fapados megoldást adunk rá.

Először létrehozunk egy csatolmány modellt. Egy csatolmányról megjegyezzük a MIME-típusát, azt az erőforrást, amelyhez kapcsolódik, a feltöltött fájl elérési útvonalát a fájlrendszeren, az eredeti nevét és méretét. A csatolmányokat a fájlrendszeren a `public` könyvtárban létrehozott `data` alkönyvtárban tároljuk, mert bináris adatot az adatbázis sem tud a fájlrendszernél hatékonyabban tárolni. A gyakorlaton ezt két migrációval valósítottuk meg, az erőforrások referenciáját utólag adtuk hozzá a modellhez.

```
kovacsg@debian:~/gyakorlat/app/views/items> rails g model
    attachment name:string path:string mime:string size:integer
Running via Spring preloader in process 9727
      invoke  active_record
      create    db/migrate/20210427110237_create_attachments.rb
      create    app/models/attachment.rb
      invoke    test_unit
      create      test/models/attachment_test.rb
      create      test/fixtures/attachments.yml
kovacsg@debian:~/gyakorlat/app/views/items> rails g migration
    AddResourceToAttachments resource:references
Running via Spring preloader in process 9865
      invoke  active_record
      create
      db/migrate/20210427110330_add_resource_to_attachments.rb
```

Az `attachments` táblát létrehozó migráció:

```
class CreateAttachments < ActiveRecord::Migration[6.1]
  def change
    create_table :attachments do |t|
      t.string :name
      t.string :path
      t.string :mime
      t.integer :size

      t.timestamps
    end
  end
end
```

Az idegen kulcsot az `attachments` táblához hozzáadó létrehozó migráció az alábbi. Mivel még nincsenek erőforrás rekordjaink az adatbázisban, a `null` opció `false` értékre állítása nem fog hibát okozni.

```
class AddResourceToAttachments < ActiveRecord::Migration[6.1]
  def change
    add_reference :attachments, :resource, null: false,
        foreign_key: true
  end
end
```

Majd végrehajtjuk a migrációt.

```
kovacsg@debian:~/gyakorlat/db/migrate> rails db:migrate
== 20210427110237 CreateAttachments: migrating
   ================================
-- create_table(:attachments)
   -> 0.0256s
```

```
== 20210427110237 CreateAttachments: migrated (0.0260s)
   ==========================

== 20210427110330 AddResourceToAttachments: migrating
   ==========================
-- add_reference(:attachments, :resource, {:null=>false, :
   foreign_key=>true})
   -> 0.0422s
== 20210427110330 AddResourceToAttachments: migrated (0.0425s)
   ==========================
```

Vegyünk fel Rails konzolon néhány erőforrást minden egyes projekthez, és a reprodukálhatóság végett, mentsük és a kódrészletet a `db/seeds.rb` fájlba.

```
irb(main):017:0> Item.all.each do |item| for i in 1..10 do item.
   resources << Resource.create(name: "Resource #{i}", version:
   '1.0', item: item) end end
```

A csatolmány egy erőforráshoz tartozik, egy-egy kapcsolatban áll vele, ezért módosítjuk a két modell osztályt.

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

Mivel az idegen kulcsot utólagos migrációval hoztuk létre, a csatolmányok modellben a kapcsolat a `references` típusú attribútum miatt nem jött automatikusan létre, azt pótolnunk kell.

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

A csatolmány fel- és letöltésének eseménykezelőjét az erőforrásook kontrollerében 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 :resources do
    get 'download', to: 'resources#download', as: 'download', on:
        :member #:collection
    post 'upload', to: 'resources#upload', as: 'upload', on: :
       member
  endend
```

Az erőforrás adatlapja nézeten korábban létrehoztuk a fájlfeltöltés műveletet (`resources/show.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.

```
<fieldset>
  <legend>Upload resource</legend>
  <%= form_tag(upload_resource_path(@resource.id), method: :post,
      multipart: true) do %>
    <%= label_tag 'File' %>
    <%= file_field_tag :attachment %>
    <%= submit_tag 'Upload' %>
  <% end %>
</fieldset>
```

Hozzuk létre a feltöltés eseménykezelőjét az erőforrások kontrollerében, ott is a feltöltés akcióban. A csatolmány erőforráshoz kapcsolásához szükségünk van az erőforrás példányára, ezért módosítjuk a `before_action` szűrőfeltételének listáját, hogy az `upload` akció előtt is lefusson az aktuális erőforrás keresése. A csatolmány tulajdonosa ez az erőforrás lesz. A felhasználót végül az erőforrás adatlapjára irányítjuk vissza.

```
class ResourcesController < ApplicationController
  before_action :set_resource, only: %i[ show edit update destroy
      upload ]
  def upload
    a = Attachment.save_file params[:attachment], @resource
    redirect_to @resource
  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 az erőforrást, 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 Attachment < ApplicationRecord
  def Attachment.save_file(file, resource)
    return if file.nil?
    dir = Rails.root.join('public', 'data')
    unless File.exists? dir
      Dir.mkdir dir
```

```
    end
    fname = file.original_filename
    a = Attachment.create resource: resource, name: fname, path:
        '', mime: file.content_type
    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)
  end
end
```

Az erőforrás adatlapján (`show.html.erb`) megjelenítjük a csatolmányt egy letölthető linkként, amennyiben van. A link a letöltés akcióra mutat.

```
<p>
  <strong>Download:</strong>
  <%= link_to "Download", download_resource_path(@resource.id) %>
</p>
```

A letöltés akciót az erőforrások kontrollerébe helyezzük el. Az erőforrás 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, amennyiben létezik a csatolmány, ellenkező esetben pedig visszairánítju az erőforrás adatlapjára. A csatolmányt mentésre kínáljuk a `disposition` opció `attachment` értékre állításával, a `inline` érték esetén a böngésző megkísérli megnyitni a fájlt a `type` opcióban szereplő MIME-típus alapján.

```
class ResourcesController < ApplicationController
  before_action :set_resource, only: %i[ show edit update destroy
      upload download ]
  def download
    a = @resource.attachment
    unless a.nil?
      send_file a.path, type: a.mime, filename: a.name,
          disposition: :attachment #:inline
    else
      redirect_to @resource
    end
  end
end
```