Rails routing, kontrollerek
Gyakorlat

Kovács Gábor

2016. április 11.

A gyakorlat során az előző gyakorlatokon megkezdett feladat megoldását folytatjuk. A harmadik gyakorlaton kialakított menüben a felhasználóra vonatkozó útvonalak közösek, 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 értintett linkek a felhasználói profil megtekintése, a profil szerkesztése nézetek, illetve az utóbbihoz tartozó eseménykezelők. Nevezzük el ezeket az útvonalakat a jobb karbantarthatóság végett.

```
  get 'users/:id/edit/', as: 'profile', to: 'users#edit'
  put 'users/:id', as: 'update_user', to: 'users#update'

  get 'users/:id', as: 'user', to: 'users#show'
```

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

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

<%= link_to "View profile", user_path(@user.id) %><br/>
<%= link_to "Edit profile", profile_path(@user.id) %><br/>
<%= link_to "My recipes", recipes_path %><br/>
<%= link_to "New recipe", new_recipe_path %><br />
<%= link_to "Logout", logout_path %>
</fieldset>
}

A felhasználó, az az aktuálisan
bejelentkezett felhasználó, akit a \verb!session!-ből vehetünk
elő. Mive erre minden műveletnél szükségünk lesz, a
```

```
kontrollerek ősosztályában inicializáljunk ennek megfelelően a
    példányváltozókat.

{\footnotesize
class ApplicationController < ActionController::Base
  before_filter :find_user
  protect_from_forgery with: :exception

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

A receptek kontrollerében a inicializálatlan akcióink maradtak vissza az előző gyakorlat után, és hiányoznak az eseménykezelő metódusaink. De mielőtt belekezdenénk módosítsuk a receptek útvonalait, hogy azok jobban illeszkedjenek a Rails konvenciókhoz. Mind a `recipes`, mind a `recipes/:id` útvonalat kétszer definiáljuk, de különböző HTTP metódussal férünk hozzá, így az eseményhez kötendő kontroller metódus egyértelmű marad.

```
  post 'recipes', as: 'create_recipe', to: 'recipes#create'

  get 'recipes', as: 'recipes', to: 'recipes#index'
  get 'recipes/:id/edit', as: 'edit_recipe', to: 'recipes#edit'
  put 'recipes/:id', as: 'update_recipe', to: 'recipes#update'

  get 'recipes/:id', as: 'recipe', to: 'recipes#show'
```

A receptek kontrollerben az `edit`, az `update` és `show` akciók esetén a `params` hashben megkapjuk a hivatkozott recept azonosítóját, amely alapján előkeressük az adatbázisból a hivatkozott recept objektumot.

```
class RecipesController < ApplicationController
  before_filter :find_recipe, only: [:edit, :update, :show]
  private
    def find_recipe
      @recipe = Recipe.find params[:id]
    end

    def recipe_params
      params.require(:recipe).permit([:name, :description, :image
          ])
    end
end
```

A `params` hashben lévő, a HTTP kérésből kinyert paramétereket fehérlistáznunk kell, amelyet a `recipe_params` metódusban teszünk meg, amely visszatérési értéke átadható a modell `create` és `update` metódusainak.

```
class RecipesController < ApplicationController
  private
    def recipe_params
      params.require(:recipe).permit([:name, :description, :image
          ])
    end
end
```

A kontroller akcióiban használjuk fel a két fenti metódust. A `create` és az `update` akcióhoz nem tartozik önálló nézet, ezért azok végrehajtása után navigáljuk át a felhasználót a `show` akció nézetére egy átirányítással.

```
class RecipesController < ApplicationController
  def new
    @recipe = Recipe.new
  end

  def create
    Recipe.create(recipe_params)
    redirect_to recipe_path(@recipe)
  end

  def edit
  end

  def update
    Recipe.update(recipe_params)
    redirect_to recipe_path(@recipe)
  end

  def show
  end

  def index
    @recipes = Recipe.all
  end
end
```

Tegyük lehetővé, hogy a felhasználó feltölthessen pontosan egy képet egy recepthez az elkészült ételről. Ehhez fájlfeltöltést valósítunk meg, amelyhez szükségünk lesz egy új modellre, ami a feltöltött képekről tárol adatot. Egy csatolmányról megjegyezzük a MIME-típusát, az az üzenetet, amihez 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/app/views/recipes# rails g model Image
    name:string mime:string recipe:references path:string size:
    integer
```

Hajtsuk végre a migrációt, majd indítsuk újra a webszerverünket.

```
kovacsg@debian:~/gyakorlat/app/views/recipes# rails db:migrate
== 20170411103107 CreateImages: migrating
   ===========================================
-- create_table(:images)
   -> 0.4175s
== 20170411103107 CreateImages: migrated (0.4178s)
   ==========================
```

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

```
class Recipe < ApplicationRecord
  has_one :image
end
```

A birtokolt objektum a csatolmány.

```
class Image < ApplicationRecord
  belongs_to :recipe
end
```

A fájlfelöltéshez módosítanunk kell az receptet létrehozó, illetve szerkesztő nézetet, 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 :recipe, url: update_recipe_path(@repice), html: {
    multipart: true }, method: :put do |form| %>
  ...
    <div>
      <%= form.label :image %>
      <%= form.file_field :image %>
    </div>
  ...
<% end %>
```

Ezek az eseménykezelője a `create`, illetve az `update` metódus a `recipes` kontrollerben. Az üzenet sikeres mentése esetén mentsük el az üzenethez asszociált csatolmányt is. Mivel a képet is tartalmazó recept mentése nem

csak adatbázisműveletből áll, ezért azt szét kell tördelnünk darabokra: beállítjuk a recept modell példányának attribútumait, és a sikeresen létrehoztuk, elmentjük a hozzá asszociált csatolmányt.

```
class RecipesController < ApplicationController
  def create
    #Recipe.create(recipe_params)
    @recipe = Recipe.new
    @recipe.name = params[:recipe][:name]
    @recipe.user = @user
    @recipe.description = params[:recipe][:description]
    if @recipe.save
      unless params[:recipe][:image].nil?
        Image.saveFile(params[:recipe][:image], @recipe)
      end
    end
    redirect_to recipe_path(@recipe)
  end

  def update
    #Recipe.update(recipe_params)
    @recipe.name = params[:recipe][:name]
    @recipe.description = params[:recipe][:description]
    @recipe.user = @user
    @recipe.save
    unless params[:recipe][:image].nil?
      Image.saveFile(params[:recipe][:image], @recipe)
    end
    redirect_to recipe_path(@recipe)
  end
end
```

A fájl mentését a kép modell egy osztálymetódusában valósítjuk meg, aminek két paramétere lesz, a feltöltött temporális fájl és az asszociált recept. 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 elmentjük azt az adatbázisba.

```
class Image < ApplicationRecord
  def Image.saveFile(upload, recipe)
    dir = Rails.root.join("public","images")
    unless File.exists? dir
      Dir.mkdir dir
    end
    fname = upload.original_filename
```

```
    image = Image.new
    image.mime = upload.content_type
    image.name = fname
    image.recipe = recipe
    image.save
    logger.error "Image id: #{image.id}"
    path = File.join(dir, image.id.to_s)
    File.open(path, "wb") do |f| f.write(upload.read) end
    image.path = path
    image.size = File.size(path)
    image.save
  end
end
```

Jelenítsük meg a feltöltött képet a recept felett, ha az adott üzenethez tartozik feltöltött kép is. A képet beágyazzuk, ezért azt `<img>`-ként jelenítjük meg, aminek átadjuk az elérhetőségét és adatbázisbeli azonosítóját.

```
<% unless @recipe.image.nil? %>
  <img src="<%= download_path(@recipe.image.id)%>" width="300"/>
<% end %>
```

A kép URL-jének definiálnunk kell egy új útvonalat:

```
get 'download/:id', as: 'download', to: 'recipes#download'
```

Ennek az eseménykezelője a `recipes` kontrollerben `download` néven definiálandó metódus, ami az adatbázisból előkeresi a kérés paraméterének megfelelő csatolmányt, és visszaküldi a fájrendszerről az alapján előkeresett fájlt válaszként.

```
class RecipesController < ApplicationController
  def download
    image = Image.find params[:id]
    send_file image.path, type: image.mime, disposition: :inline
  end
end
```

Mivel receptből 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 frissítjük a függőségeinket:

```
bundle update
```

A tördelés megvalósítása négy lépésből áll. A modell osztályban definiálunk egy függvény, ami az oldalszám paraméterhez előkeresi az adott oldalra

eső töredéket a felhasználó összes, létrehozási idő tekintetében csökkenő sorrendbe rendezett bejegyzése közül. Egy oldalon 2 receptet engedélyezünk.

```
  def self.getPage(page, user)
    if user.nil?
      Recipe.all.order(updated_at: :desc).paginate(page: page,
          per_page: 2)
    else
      user.recipes.order(updated_at: :desc).paginate(page: page,
          per_page: 2)
    end
  end
```

A kontrollerben ezután nem a felhasználó összes üzenetét kérjük el az adatbázisból, hanem csak az aktuális töredéket.

```
class RecipesController < ApplicationController
  def index
    @recipes = Recipe.getPage(params[:page], @user)
  end
end
```

A nézethez pedig hozzáadjuk a lapozó linkeket. A linkek alapéertelmezés szerint a `page` paraméterhez rendelt értékben adják át a megjelenítendő töredés sorszámát.

```
<ul>
  <% for recipe in @recipes do %>
  <li><%= link_to recipe.name, recipe_path(recipe) %></li>
  <% end %>
</ul>
<%= will_paginate @recipes %>
```

Negyedik lépésként a linkünk szebb kinézete végett módosítjuk az útvonalat, hogy az opcionálisan tarlamazhassa a töredéket is. Mivel a töredék sorszámát a Rails `:id`-ként is értelmezhetné, ezért bevezetünk egy statikus útvonalszakaszt, az `index`-et, és sorrendben ezt a `get 'recipes/:id'` útvonal előtt helyezzük el, hogy ez illeszkedjék előbb, mert maga az `index` is értelmezhető `id`-ként. Az `:page` paraméter opcionális, ha meg van adva, akkor bekerül a `params` hash-be.

```
  get 'recipes/index(/:page)', as: 'recipes', to: 'recipes#index'
```

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

```
root :to => "recipes#index"
```