Rails routing, kontrollerek
Gyakorlat

Kovács Gábor

2017. november 7.

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.

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

  match 'users/new', to: 'users#new', via: :get, as: 'register'
  post 'users/create', to: 'users#create', as: 'create_user'

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

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


  get 'users/forgotten', to: 'users#forgotten', as: '
      forgotten_password'
  post 'users/send_forgotten', to: 'users#send_forgotten', as: '
      recover_password'

  resources :items
  get 'say/hello', to: 'say#hello', as: 'hello'
end
```

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.

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

<%= link_to 'Register', register_path %> <br/>
<%= link_to "Forgotten password", forgotten_password_path %>
</fieldset>
```

A sporteszközök kontrollerében korábban kikommenteztük az automatikusan generált kódot, és kezdezi adatokat állítottunk be, ezeket állítsuk vissza.

Tegyük lehetővé, hogy a felhasználó feltölthessen pontosan egy képet egy sporteszközhöz. 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/users> rails g model image
    path:string mime:string size:integer filename:string item:
    references
       invoke  active_record
       create    db/migrate/20171107113029_create_images.rb
       create    app/models/image.rb
       invoke    test_unit
       create      test/models/image_test.rb
       create      test/fixtures/images.yml
```

Hajtsuk végre a migrációt, majd indítsuk újra a webszerverünket.

```
kovacsg@debian:~/gyakorlat/app/views/users> rails db:migrate
== 20171107113029 CreateImages: migrating
   =============================================
-- create_table(:images)
   -> 0.0482s
== 20171107113029 CreateImages: migrated (0.0483s)
   ====================================
```

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

```
class Item < ApplicationRecord
```

```
  has_one :image
end
```

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

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

A fájlfelöltéshez módosítanunk kell a sporteszközt létrehozó, illetve szerkesztő nézet formját (`_form.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_with(model: item, local: true, html: { multipart: true }
    ) do |form| %>
  ...
  <div class="field">
    <%= form.label :image %>
    <%= form.file_field :image %>
  </div>
  ...
<% end %>
```

Ezek az eseménykezelője a `create`, illetve az `update` metódus az `items` kontrollerben. Először a kérés paraméterei közül fehérlistáznunk kell a `image` kulcsot is, de mivel az nem a sporteszköz 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 ItemsController < ApplicationController
  def create
    params.require(:item).permit(:name, :barcode, :price, :image)
    @item = Item.new(
      name: params[:item][:name],
      barcode: params[:item][:barcode],
      price: params[:item][:price])
    respond_to do |format|
      if @item.save
        format.html { redirect_to @item, notice: 'Item was
            successfully created.' }
        format.json { render :show, status: :created, location:
            @item }
        unless params[:item][:image].nil?
```

```
          Image.saveFile params[:item][:image], @item
        end
      else
        format.html { render :new }
        format.json { render json: @item.errors, status: :
            unprocessable_entity }
      end
    end
  end

  def update
    params.require(:item).permit(:name, :barcode, :price, :image)
    respond_to do |format|
      if @item.update(
          name: params[:item][:name],
          barcode: params[:item][:barcode],
          price: params[:item][:price])
        format.html { redirect_to @item, notice: 'Item was
            successfully updated.' }
        format.json { render :show, status: :ok, location: @item
            }
        unless params[:item][:image].nil?
          Image.saveFile params[:item][:image], @item
        end
      else
        format.html { render :edit }
        format.json { render json: @item.errors, status: :
            unprocessable_entity }
      end
    end
  end
end
```

A fájl mentését a kép modell egy osztálymetódusában valósítjuk meg, aminek két paramétere lesz, a feltöltött temporális fájl és az asszociált sporteszköz. 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 self.saveFile(upload, item)
    dir  = Rails.root.join("public","data")
    unless File.exists? dir
      Dir.mkdir dir
    end
```

```
    fname = upload.original_filename
    image = Image.new
    image.path = fname
    image.mime = upload.content_type
    image.item = item
    image.filename = fname
    image.save
    path = File.join(dir, image.id.to_s) #dir.join(image.id)
    File.open(path, 'wb') do |f| f.write(upload.read) end
    image.path = path
    image.size = File.size(path)
    image.save
  end
end
```

A sporteszköz mellett már korábban lefoglaltuk a helyet a kép számára, azonban annak nem adtuk meg a `src` attribútumát. Ezt a `show.html.erb` és a `_show.html.erb` fájlokban kell megtennünk.

```
    <img src="<%= download_path(@item.image.id) unless @item.
        image.nil? %>" alt="Here come the image" width="120"
        height="120" />
```

A kép URL-jének definiálnunk kell egy új útvonalat:

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

Ennek az eseménykezelője legyen az `items` kontrollerben `download` néven definiálandó metódus, ami az adatbázisból előkeresi a kérés paraméterének megfelelő csatolmányt, és visszaküldi a fájrendszerről az alapján előkeresett fájlt válaszként. Ezt az eseménykezelőt más kontrollerben is elhelyezhettük volna, architekturális szempontból talán ez volt ebben a pillanatban a legjobb döntés.

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

Mivel sporteszközből 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 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 4 sportszert engedélyezünk.

```
class Item < ApplicationRecord
  def self.get_page(page)
    Item.all.paginate(page: page, per_page: 4)
  end
end
```

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

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

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

```
<%= will_paginate @items %>

<% @items.each do |item| %>
  <%= render 'show', locals: { item: item } %>
<% end %>
```

A menüben meg kell különböztetnünk a bejelentkezett felhasználó által látható összes sportszert, valamint a korábbi és az aktuális kölcsönzéseket. Mivel az erőforrások útvonalainak konfigurálásával még nem foglalkoztunk, egyelőre használjunk egy `q` nevű paramétert erre az útvonalban. Jelentse az `a` az összes, a `c` az épp kölcsönzött és a `h` a korábban kölcsönzött sportszerek listáját.

```
<%= link_to "Browse items", "/items?q=a" %><br/>
<%= link_to "Borrowed items", "/items?q=c" %><br/>
<%= link_to "History", "/items?q=h" %><br/>
```

Az összes sporteszközt kell szűkítenünk e paraméterek és a tördelés értéke alapján, így két további paramétert veszünk fel az imént létrehozott `get_page` metódusunkba. Az egyik a szűrés típusa, a másik a felhasználó, hogy melyik felhasználó aktuális és korábbi kölcsönzéseiről van is szó. Ha

nincs szűrőfeltétel vagy az összes sportszert keressünk, akkor az `Item.all` hívást kell használnunk. Ha egy felhasználó korábbi kölcsönzéseire vagyunk kíváncsiak, akkor a felhasználó objektumhoz asszociált kölcsönzések tömbből csak azokat kérjük le az adatbázisból, melyeknél a visszaadási dátum nem nulla, az aktuális kölcsönzéseknél pedig azokat, amelyeknél a visszadási dátum nulla. Mindkét esetben az `each` metódussal iterálunk a kölcsönzéseket tartalmazó tömbön és hozzáadjuk az egyes kölcsönzések tárgyát jelentő sporteszköz objektumot egy tömbhöz. Végül a tördelést ezen a tömbön végezzük el.

```
class Item < ApplicationRecord
  def self.get_page(page, q = nil, user = nil)
    #Item.all
    if q.nil? || q == 'a' || user.nil?
      items = Item.all
    elsif !user.nil? && q == 'h'
      items = []
      user.borrows.where.not(return_time: nil).each do |b| items
          << b.item end
    elsif !user.nil? && q == 'c'
      item = []
      user.borrows.where(return_time: nil).each do |b| items << b
          .item end
    end
    items.paginate(page: page, per_page: 4)
  end
end
```

A kontrollerben is kibővül e két paraméterrel.

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

A sporteszköz adatlapján a bejelentkezett felhasználó lát két műveletet, a kölcsönzést és a visszadást, amelyeket még nem valósítottunk meg. Mindkettőhöz fel kell vennünk egy útvonalat, melyeknek két paramétere lesz, a kölcsönvevő felhasználó azonosítója és a kölcsönvett sporteszköz azonosítója. A sportszer `show` és `_show` nézeteiben mindkét információ elérhető

```
<%= link_to 'Borrow', borrow_path(@user.id, @item.id) %>|
<%= link_to 'Return', return_path(@user.id, @item.id) %>
```

Az új útvonalaink hivatkozzanak a sportszerek kontroller egy-egy műveletére. Ismét architekturális kérdés, hogy ezek itt jó helyen vannak-e. A döntés pillanatában ez az optimális.

```
  get 'borrow/:user/:item', to: 'items#borrow', as: 'borrow'
  get 'return/:user/:item', to: 'items#return', as: 'return'
```

A kontrollerben mindkét metódus előkeresi az adatbázisból a kérés paraméterei alapján a kölcsönző felhasználó és a kölcsönzendő sportszer objektumait, elvégzi a kölcsönzés műveletet, amely mivel adatmanipuláció, annak helye a kölcsönzés modellben van, majd megjeleníti az index nézetet.

```
class ItemsController < ApplicationController
 def borrow
    item = Item.find params[:item]
    user = User.find params[:user]
    Borrow.borrow item, user
    @items = Item.get_page(params[:page], params[:q], @user)
    redirect_to :index
 end

 def return
    item = Item.find params[:item]
    user = User.find params[:user]
    Borrow.return item, user
    @items = Item.get_page(params[:page], params[:q], @user)
    redirect_to :index
 end
```

A kölcsönzés modell `borrow` és `return` metódusainak megvalósítása a következő gyakorlatra maradt.

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

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