Rails routing, kontrollerek
Gyakorlat

Kovács Gábor

2023. november 14.

1. Lapozás

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 két feladat. Egyelőre csak egy feladatunk van az adatbázisban, ezért vegyünk fel pár újat.

```
irb(main):041* for i in 2..6 do
irb(main):042*   Task.create number: i, description: "#{i.
    ordinalize} task", url: "http://gyakorlat.com/tasks/
#{i}", deadline: (Date.new(2023,9,21) + (2*i).weeks )
irb(main):043> end
  TRANSACTION (0.2ms)  BEGIN
  Task Create (14.1ms)  INSERT INTO `tasks` (`number`, `url`, `
      description`, `deadline`, `created_at`, `updated_at`)
      VALUES (2, 'http://gyakorlat.com/tasks/2', '2nd task', '
      2023-10-19', '2023-11-14 11:26:43.861150', '2023-11-14
      11:26:43.861150')
  TRANSACTION (2.0ms)  COMMIT
  TRANSACTION (0.1ms)  BEGIN
  Task Create (0.6ms)  INSERT INTO `tasks` (`number`, `url`, `
      description`, `deadline`, `created_at`, `updated_at`)
      VALUES (3, 'http://gyakorlat.com/tasks/3', '3rd task', '
      2023-11-02', '2023-11-14 11:26:43.881197', '2023-11-14
      11:26:43.881197')
  TRANSACTION (0.2ms)  COMMIT
  TRANSACTION (0.1ms)  BEGIN
  Task Create (0.6ms)  INSERT INTO `tasks` (`number`, `url`, `
      description`, `deadline`, `created_at`, `updated_at`)
      VALUES (4, 'http://gyakorlat.com/tasks/4', '4th task', '
      2023-11-16', '2023-11-14 11:26:43.883388', '2023-11-14
      11:26:43.883388')
```

```
  TRANSACTION (0.2ms)  COMMIT
  TRANSACTION (0.1ms)  BEGIN
  Task Create (0.5ms)  INSERT INTO `tasks` (`number`, `url`, `
      description`, `deadline`, `created_at`, `updated_at`)
      VALUES (5, 'http://gyakorlat.com/tasks/5', '5th task', '
      2023-11-30', '2023-11-14 11:26:43.885321', '2023-11-14 
      11:26:43.885321')
  TRANSACTION (0.2ms)  COMMIT
  TRANSACTION (0.1ms)  BEGIN
  Task Create (0.9ms)  INSERT INTO `tasks` (`number`, `url`, `
      description`, `deadline`, `created_at`, `updated_at`)
      VALUES (6, 'http://gyakorlat.com/tasks/6', '6th task', '
      2023-12-14', '2023-11-14 11:26:43.887071', '2023-11-14 
      11:26:43.887071')
  TRANSACTION (0.5ms)  COMMIT
=> 2..6
```

A lapozás megvalósításához felveszünk egy új függőséget a `Gemfile`-ba:

```
gem 'will_paginate'
```

majd telepítjük a hiányzó függőségeinket:

```
bundle install
```

A lapozás megvalósítása ezután egyszerűvé válik. A `tasks/index.html.erb` nézeten felvesszük a lapozó linkeket az `@tasks` példányváltozóra.

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

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

```
class TasksController < ApplicationController
  def index
    @tasks = Task.get_task_page(params[:page])
  end
end
```

A feladat modell osztályt kiegészítjük a lapot lekérdező függvénnyel, melynek egy paramétere van, a lap sorszáma. A feladatokat létrehozási idejük szerint növekvő sorrendbe rendezzük, majd lekérdezzük a lapot úgy, hogy egy lap pontosan 5 feladatot tartalmazzon.

```
class Task < ApplicationRecord
    def self.get_task_page(page)
        Task.all.order(deadline: :asc).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).

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

2. Fájlok fel- és letöltése

A megoldásokhoz csatolhatunk fájlokat, amelyet csatolmány feltöltésével valósítunk meg. Léteznek erre kész API-k, mint a PaperClip vagy a CarrierWave, most azonban fapados megoldást adunk rá.

A csatolmányok számára a megoldások (`Submission`) modellünkben korábban létrehoztunk egy, csak a memóriában létező példányváltozót, most lecseréljük azt egy kapcsolt modellre. 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.

```
kovacsg@debian:~/gyakorlat> rails g model attachment submission:
    references filename:string mime:string path:string size:
    integer
      invoke  active_record
      create    db/migrate/20231114114606_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> rails db:migrate
== 20231114114606 CreateAttachments: migrating
    ======================================
-- create_table(:attachments)
   -> 0.0231s
== 20231114114606 CreateAttachments: migrated (0.0237s)
    ==============================
```

```ruby
class CreateAttachments < ActiveRecord::Migration[6.0]
  def change
    create_table :attachments do |t|
      t.references :solution, 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.

```
kovacsg@debian:~/gyakorlat# rails g model attachment solution:
    references name:string mime:string path:string size:integer
      invoke  active_record
      create    db/migrate/20200420070231_create_attachments.rb
      create    app/models/attachment.rb
      invoke    test_unit
      create      test/models/attachment_test.rb
      create      test/fixtures/attachments.yml
```

A csatolmány egy megoldáshoz tartozik, de egy megoldáshoz több csatolmány is tartozhat, több változatot is elfogadunk, de csak az utolsót vesszük figyelembe. Ezért módosítjuk a megoldások modellünket:

```ruby
class Solution < ApplicationRecord
  has_many :attachments

  def attachment
    self.attachments.order(created_at: :desc).first
  end
end
```

A megoldásokat a fájlrendszeren a `public` könyvtárban létrehozott `data` alkönyvtárban tároljuk, mert bináris adatot az adatbázis sem tud a fájlrendszernél hatékonyabban tárolni.

A csatolmány fel- és letöltésének eseménykezelőjét a megoldások kontrollerrel valósítjuk meg, amelyben a feltöltés (`create`) az új megoldás eseménykezelője lesz, és a letöltés (`download`) pedig a `show` nézetben látható link eseményezelője lesz.

Módosítjuk a fájlfeltöltő és -letöltő útvonalakat! A megoldás beadása képernyőre a feladat adatlapja képernyőn található beadás linkre kattintással juthatunk el. Mivel a megoldás modell tartalmaz egy hivatkozást arra a feladatra, amelyre a megoldás született, a linkben át kell adnunk a feladat azonosítóját. Ezért az útvonalba fel kell vennünk egy azonosítót, amelyet `task_id`-nak nevezünk el. A fájlfeltöltésnek két paramétere lesz: a feltöltő felhasználó azonosítója, a feladat azonosítója; a letöltésnek pedig egy, a csatolmány azonosítója. Az új megoldás képernyő eseménykezelője a `create` akció lesz, amelyet elnevezünk `update`-nek létrehozva a `update_path` és `update_url` helpereket. A megoldás megtekintéséhez át kell adnunk a megoldás azonosítóját, ezért a `show` akcióra mutató linket kiegészítjük egy `:id` útvonaltaggal. A megoldás letöltéséhez ugyancsak szükségünk van a megoldás azonosítójára, ezért azt eleve így hozzuk létre, és elnevezzük `download`-nak.

```
Rails.application.routes.draw do
  get 'submissions/show/:id', to: 'submissions#show', as: '
     show_submission'
  get 'submissions/new/:task_id', to: 'submissions#new'
  post 'submissions/create', as: 'upload'
  get 'submissions/download/:id', to: 'submissions#download', as:
      'download'
end
```

A `tasks/show` nézetben módosítjuk a linket kiegészítve az aktuális feladat azonosítójának sztringgé konvertált változatával, így az új megoldás képernyőn fel tudjuk használni ezt az értéket.

```
<%= link_to "Submit solution", '/submissions/new/'+@task.id.to_s
    %>
```

Az útvonalból kapott `task_id` nevű paraméter értéke alapján előkeressük az adatbázisból a hozzá kapcsolódó feladat objektumot, és azt elérhetővé tesszük az új megoldás képernyő számára.

```
 class SubmissionsController < ApplicationController
  def new
    @submission = Submission.new
    @task = Task.find params[:task_id]
  end
end
```

Az új megoldás képernyőn (`submissions/new.html.erb`) módosítjuk beadó űrlapunkat, hogy az képes legyen a fájlfeltöltésre. Ehhez fel kell vennünk a `multipart` opciót a paraméterek közé, és rejtett paraméternek át kell adunk a feladat azonosítóját. A felhasználót nem, mert a feladatot beadó személy az aktuálisan bejelentkezett felhasználó lesz. Az eseménykezelő URL módosítjuk, hogy az a `upload_path` helpert használja.

```
<%= form_with model: @submission, url: upload_path, multipart:
    true, method: :post do |f| %>
    <%= f.hidden_field :task_id, value: @task.id %>
```

```
    <%= f.label :attachment %>
    <%= f.file_field :attachment %>
    <%= f.submit "Upload" %>
<% end %>
```

Hozzuk létre a megoldás beadása eseménykezelőjét (`create`) a megoldások kontrollerben. A megoldás objektum létrehozásához szükségünk van a feladatra és a felhasználóra, melyek idegen kulcsok a megoldás modellben. A feladatot a paraméterül átadott azonosító alapján előkereshetjük az adatbázisból, a felhasználó pedig a `ApplicationController`-ben előkeresett bejelentkezett felhasználó lesz. E párossal létrehozunk egy megoldás objektumot, majd elmentjük az adatbázisba. A feltöltött fájlra a a nézetben megadott `submission[attachment]` paraméternévvel hivatkozhatunk. Ha e paraméter rendelkezik értékkel, akkor mentsük el. Önálló nézetünk nincs, ezért a következő oldalunk az előző oldal újratöltése lesz.

```
class SubmissionsController < ApplicationController
  def create
    unless params[:submission][:attachment].nil?
        submission = Submission.new user: @session_user, task_id:
            params[:submission][:task_id]
        submission.save
        Attachment.save_file params[:submission][:attachment],
            submission
    end
    redirect_back fallback_location: hello_path
  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, amelynek beállíthatjuk a fájlnevét, a MIME típusát, és a megoldást, 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.

```
class Attachment < ApplicationRecord
  belongs_to :submission

  def Attachment.save_file(file, submission_object)
    Rails.logger.info "Attachment #{submission_object}"
    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 filename: fname, mime: file.
       content_type, submission: submission_object
    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
```

A csatolmányokat a `submissions/show.html.erb` fájban kell letölthetővé tennünk, amit egy link hozzáadásával valósítunk meg.

```
  <%= link_to "Download", download_path(@submission.id) %>
```

A letöltés akciót a megoldások kontrollerbe helyezzük el. Az akció a paraméterük kapott azonosító alapján előkeresi a csatolmányt az adatbázisból, és annak útvonal attribútuma által kijelölt fájlt visszaküldi közben jelzi a böngészőnek, hogy kínálja fel mentésre a fájlt a fájl eredeti fájlnevével. Ha nincs ilyen azonosítójú megoldás, akkor a nem létező erőforrást jelentő 404-es hibakóddal térünk vissza.

```
class SubmissionsController < ApplicationController
  def download
    a = Submission.find(params[:id]).attachment
    unless a.nil?
        send_file a.path, type: a.mime, disposition: :attachment,
            filename: a.filename
        return
    end
    head 404
  end
end
```

3. Felhasználótípusok és a menüik

Az oktató és a hallgató felhasználótípusok megkülönböztetése végett felveszünk egy új mezőt a felhasználók modellbe, a szerepet.

```
kovacsg@debian:~/gyakorlat/public/data> rails g migration
    AddRoleToUsers role:integer{1}
       invoke  active_record
       create    db/migrate/20231114123620_add_role_to_users.rb
```

A szerep alapértelmezett értéke legyen a hallgató, amelyet az 1 érték reprezentál, és módosítsuk az adatbázisban lévő összes rekord szerepét hallgatóra.

```
class AddRoleToUsers < ActiveRecord::Migration[7.0]
  def change
    add_column :users, :role, :integer, limit: 1, default: 1
    User.update_all role: 1
  end
end
```

Végrehajtjuk a migrációt.

```
kovacsg@debian:~/gyakorlat/db/migrate> rails db:migrate
== 20231114123620 AddRoleToUsers: migrating
    ======================================
-- add_column(:users, :role, :integer, {:limit=>1})
   -> 0.0085s
== 20231114123620 AddRoleToUsers: migrated (0.0089s)
    =============================
```

A modellben definiáljuk a szerep lehetséges értékeit egy enummal.

```
class User < ApplicationRecord
  enum role: { lecturer: 0, student: 1, admin: 2 }
end
```

Az enum az elnevezett értékek alapján ?-re, illetve !-re végződő gettereket és settereket hoz létre. Próbáljuk ki konzolon. Vegyünk egy felhasználót (1. sor), kérdezzük meg, hogy a felhasználó hallgató-e (2. sor), módosítsuk a szerepét oktatóra (3. sor), majd vissza (5. sor), de közben nézzük meg, hogy a művelet sikerült-e (4. sor).

```
kovacsg@debian:~/gyakorlat$ rails c
Loading development environment (Rails 7.0.8)
irb(main):001> u = User.first
  User Load (0.2ms)  SELECT `users`.* FROM `users` ORDER BY `
      users`.`id` ASC LIMIT 1
=>
#<User:0x00007f1399486170
...
irb(main):002> u.student?
=> true
irb(main):003> u.lecturer!
  TRANSACTION (0.2ms)  BEGIN
  User Exists? (0.3ms)  SELECT 1 AS one FROM `users` WHERE `users
      `.`email` = 'valaki@mail.bme.hu' AND `users`.`id` != 1
      LIMIT 1
```

```
  User Update (0.3ms)  UPDATE `users` SET `users`.`updated_at` =
     '2023-11-14 12:41:08.368208', `users`.`role` = 0 WHERE `
     users`.`id` = 1
  TRANSACTION (0.5ms)  COMMIT
=> true
irb(main):004> u
=>
#<User:0x00007f1399486170
 id: 1,
 name: "Valaki",
 email: "valaki@mail.bme.hu",
 neptun: "aaaaaa",
 encrypted_password: "[FILTERED]",
 created_at: Tue, 31 Oct 2023 12:44:40.710832000 UTC +00:00,
 updated_at: Tue, 14 Nov 2023 12:41:08.368208000 UTC +00:00,
 salt: "[FILTERED]",
 role: "lecturer">
irb(main):005> u.student!
  TRANSACTION (0.3ms)  BEGIN
  User Exists? (0.4ms)  SELECT 1 AS one FROM `users` WHERE `users
     `.`email` = 'valaki@mail.bme.hu' AND `users`.`id` != 1
     LIMIT 1
  User Update (0.4ms)  UPDATE `users` SET `users`.`updated_at` =
     '2023-11-14 12:41:34.594678', `users`.`role` = 1 WHERE `
     users`.`id` = 1
  TRANSACTION (0.7ms)  COMMIT
=> true
```

A feladatok `show` nézetén ezután felhasználótípushoz tudjuk rendelni az egyes funkciókat. A hallgató csak a saját megoldását láthatja, az oktató a feladatra érkezett összes megoldást. Ezeket linkként tesszük elérhetővé, amelyek a megoldás adatlapja (`show`) képernyőre vezet az egyes megoldások esetén.

```
<% if @session_user.lecturer? %>
    <ul>
    <% for submission in @task.submission %>
        <li><%= link_to "#{submission.user.name}",
            show_submission_path(submission.id) %></li>
    <% end %>
    </ul>
<% elif @session_user.student? %>
    <% if @task.submissions.where(user: @session_user).any? %>
    <%=link_to "My submission", show_submission_path(@task.
        submissions.where(user: @session_user).take.id) %>
    <% end %>
<% end %>
```