Rails routing, kontrollerek
Gyakorlat

Kovács Gábor

2011. április 11.

Legelőször az útvonalakat tartalmazó `config/routes.rb` fájlban állítsuk át a webszerverünk gyökerét a teendők nézet `index` oldalára.

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

A gyakorlat során az előző gyakorlatokon megkezdett feladat megoldását folytatjuk. Először a teendőkhöz kapcsolt csatolmányokat tesszük rendbe. A csatolmányok a teendőkhöz kapcsolódnak, jelenleg egy csatolmánnyal kapcsolatban annak a fájlrendszeren tárolt nevét és a csatolmányt tulajdonló teendő azonosítóját jegyezzük meg. A csatolmány azonban rendelkezik típussal is, lehet szöveg, kép vagy egyén formátumú, ezért jegyezzük meg a MIME típusát. A teendőhöz fűzött csatolmányt nem feltétlenül a teendő tulajdonosa hozta létre, ezért a létrehozó felhasználóra is generálunk egy idegen kulcsot. A következő kép parancs elvégzi a szükséges módosításokat a sémán.

```
rails generate migration AddUserIdAndMimeToAttachments
  \
  user_id:integer mime:string
rake db:migrate
```

A csatolmányt létrehozó felhasználó és a csatolmány között modell szinten is hozzuk létre a kapcsolatot. Bár a felhasználók és a csatolmányok között egy-több reláció van, mivel az egyes felhasználók által létrehozott csatolmányokra nem vagyunk kíváncsiak, csak a csatolmányt létrehozó felhasználóról szeretnénk információt megjeleníteni, a kapcsolat csak egy irányban lesz navigálható. Ezért csak `Attachment` modell osztályt módosítjuk.

```
belongs_to :user
```

Csatolmányok kezelésére most már megvan az eszköztárunk. A következő lépésben lehetővé tesszük egy bejelentkezett felhasználó számára, hogy

az aktuálisan megjelenített teendőhöz feltöltsön egy új csatolmányt. Először hozzuk létre a nézetet! Jelenítsük meg mindjárt az adatbázisban elérhető csatolmányokat egy táblázatban a `show` nézet csatolmányok számára fenntartott részében. A csatolmányokat a `downloads` útvonalon tesszük elérhetővé a teendő és a csatolmány azonosítójának megadásával. A csatolmányok enumerációját kezdetben állítsuk be az összes csatolmányra a `show` kontroller akcióban.

```
<div>
 <h3>Attachments</h3>
 <table>
   <% @attachments.each do |a| %>
     <tr>
       <td><%= link_to a.file, "/downloads/#{@issue.id
          }/#{a.id}" %></td>
       <td><%= a.user.username %></td>
       <td><%= a.updated_at %></td>
     </tr>
   <% end %>
 </table>
</div>
```

A nézetnek a címkék mellett egy fájlbeviteli mezőt és egy nyomógombot kell tartalmaznia, amelynek eseményét a még nem létező `upload` akció kezeli le. Fájlfeltöltés esetén a `multipart` HTML opciót be kell állítanunk. A nyomógomb feliratának előállításakor felhasználjuk a `ActiveSupport` modul `humanize` metódusát. Mindezek mellett jelenítsük meg a fájlfeltöltéssel kapcsolatos feldolgozás során generált flash üzeneteket.

```
 <%= flash[:file] %>
<fieldset>
  <legend>Upload a new attachment</legend>
  <%= form_for :upload, :url=>{:action=>'upload', :id=>
      @issue.id},
        :html=>{:multipart => true} do |form|%>
    <%= form.label :file %>
    <%= form.file_field 'file' %>
    <%= form.submit 'upload'.humanize %>
  <% end %>
</fieldset>
```

A fájlfeltöltést a teendők kontrollerének `upload` akciója végzi el, ami először ellenőrzi, hogy a felhasználó be van-e jelentkezve, majd megvizsgálja,

hogy a feltöltési akcióhoz tartozik-e feltöltött állomány. Ha mindkét feltétel teljesül, akkor a fájlt a modell osztályban definiálandó `save_attachment` osztálymetódussal eltároljuk a fájlrendszeren. Az egyik feltétel nem teljesül, flash üzenet segítségével tudatjuk a felhasználóval a hiba okát. Végül mivel a feltöltéshez nem tartozik külön nézet átirányítjuk a kérést az aktuális teendő `show` akciójára. Itt használhattuk volna a `render` metódust, azonban akkor helytelen URL jelenne meg a böngésző címsorában.

```
def upload
  if !session[:user].nil?
    if !params[:upload].nil?
      Attachment.save_file session[:user], params[:id
          ], params[:upload]
      flash[:file] = "File uploaded successfully"
    else
      flash[:file] = "Empty file field"
    end
  else
    flash[:file] = "Please log in to upload file"
  end
  @issue = Issue.find params[:id]
  redirect_to issue_url(@issue)
end
```

A fájl elmentéséhez három paraméterre van szükségünk. A létrehozó felhasználó azonosítójára, a teendő azonosítójára, melyhez a csatolmány tartozik és magára a csatolmányra, amit a HTTP kérés paraméterei között találhatunk meg. A feltöltött fájlokat tároljuk a Rails alkalmazás `public` könyvára alatt egy `data` könyvárban a teendő azonosíta szerint alkönyvtárakra szétválogatva. A `public` és a `data` tagok statikusak az elérési úton, azonban a teendő azonosítójának megfelelő könyvtár létezését minden kérés során ellenőriznünk kell, és amennyiben nem lézezik, létre kell hoznunk. A kövezkező lépés a feltöltött állomány nevének kinyerése a HTTP kérés paraméterei közül és a weboldal elleni támadásra alkalmas karakterek lecserélése a fájlnévben: a soreleji üres karaktereket és . karaktereket cseréljük le aláhúzásjelre. Végül fájlművelettel elmentjük az adatokat, és létrehozzuk az adatbázisbeli rekordot.

```
def self.save_file(user,issue,upload)
  dir = Rails.root.join("public","data",issue)
  if !File.exists? dir
    Dir.mkdir dir
```

```
  end
  fname = File.basename(upload['file'].
      original_filename)
  fname.gsub!(/^\w\.\_/,"_")
  path = File.join(dir, fname)
  File.open(path,"w") {|f| f.write(upload['file'].read)
      }
  a = Attachment.new
  a.file = fname
  a.user_id = user
  a.issue_id = issue
  a.mime = upload['file'].content_type
  a.save
end
```

A teendők `show` nézetében lehetővé tettük a korábban feltöltött csatolmányok letöltését, az akciót azonban egy jelenleg nem létező kontrollerre mutat, ezért módosítanunk kell az útvonalak tábláját a `routes.rb`-ban. Rendeljük a `/downloads` útvonal prefixhez az `issues` kontroller `download` akcióját, az útvonal következő két darabjából pedig képezzünk HTTP kérés paramétert, az első legyen a teendő azonosítója, a második a csatolmány azonosítója.

```
match '/downloads/:id/:aid' => 'issues#download'
```

A letöltés akció a kérés paraméterei alapján kikeresi a fájlrendszerről és az adatbázisból a HTTP kérés paramétereinek megfelelő rekordot és fájlt, ha van ilyen, és csatolmányként elküldi a felhasználónak. Végül átirányítja a kérést az aktuális teendő `show` akciójára.

Egy teendőhöz sok csatolmány, illetve komment tartozhat, akárcsak egy felhasználónak sok teendője lehet. Ha ezek száma egy határon túlnyúlik, akkor az oldaluk magassága kezelhetetlen lehet, ezért e tartalmak tekintetében megvalósítjuk a tördelés funkciót. A tördeléshez a `will_paginate` gemet használjuk, a `Gemfile`-ba beszúrjuk az alábbi sort, majd kiadjuk a `bundle install` parancsot:

```
gem 'will_paginate', '~> 3.0.0'
```

A tördeléshez három forrást kell módosítanunk. A teendők modell osztályban létrehozunk egy-egy metódust, amelyek a paraméterként átadott oldalszámhoz rendelik az adatbázisbeli csatolmányok, kommentek részhalmazát időben csökkenő sorrendben. Csatolmányból egy oldalon egyet jelenítsünk csak meg, kommentből tizet. A tördeléshez a frissen telepített gem `paginate` vagy a `page ActiveRecord` kiegészítését használjuk.

```
class Issue < ActiveRecord::Base
  def get_attachments_page(page)
    attachments.order('updated_at desc').paginate(:page
      =>page,:per_page=>1)
  end
  def get_comments_page(page)
    comments.order('updated_at desc').paginate(:page=>
      page, :per_page=>10)
  end
end
```

A teendők kontroller `show` akciójában módosítsuk a `@attachments` példányváltozó értékét, ne az összes rekordot jelenítsük meg, hanem csak a HTTP paraméterben megkapott oldalszámhoz tartozót. Az oldalszámra az alapértelmezett `page` helyett a `attachment` névvel hivatkozunk, ugyanis ezen az oldalon nem csak a csatolmányokat, hanem a kommenteket is tördeljük.

```
def show
  @attachments = @issue.get_attachments_page params[:
    attachment]
end
```

A `show` nézetbe mindössze egyetlen sort kell beszúrnunk, ami hozzáadja a lapozást megvalósító linkeket. A `param_name` opcióval felüldefiniáljuk az alapértelmezett `page` kérés paraméternevet.

```
<%= will_paginate @attachments, :param_name => '
    attachment' %>
```

Ugyanezeket a lépéseket végezzük el a felhasználó teendői vonatkozásában is. Először hozzunk létre néhány új teendőt az aktuális felhasználónk számára véletlen határidő és prioritás értékekkel egy Rails konzolon.

```
irb(main):021:0> for j in 1..25
irb(main):022:1> i = Issue.new
irb(main):023:1> i.user_id = 2
irb(main):024:1> i.priority = (rand*5).to_i
irb(main):025:1> i.label = "Teendo #{j}"
irb(main):026:1> i.description = "Surgosen csinald meg"
irb(main):027:1> i.deadline = (rand*10).days.ago
irb(main):028:1> i.status = 1
irb(main):029:1> i.save
irb(main):030:1> end
```

Az előzőek mintájára a felhasználó teendői tördelést a reláció egy oldalán, vagyis a `User` modell osztályban végezzük el ugyanúgy egy paraméterrel. A teendőket prioritás szerint rendezzük csökkenő sorrendbe egy oldalon 10 teendőt megjelenítve.

```
class User < ActiveRecord::Base
  def get_issues_page(page)
    issues.order('priority desc').paginate(:page=>page,
        :per_page=>10)
  end
end
```

A teendők kontroller `index` akciójában a listázandó teendők meghatározásakor figyelembe vesszük, hogy a felhasználó be van-e jelentkezve. Ha igen, akkor a saját teendőit látja, ha nincs, akkor az összeset. Az oldalszámra a `page` HTTP kérés paraméterrel hivatkozunk.

```
if !session[:user].nil?
  u = User.find session[:user]
  @issues = u.get_issues_page params[:page]
else
  @issues = Issue.all
end
```

A nézetben annyival van egyszerűbb dolgunk, hogy nem kell átneveznünk a paraméter nevét.

```
<%= will_paginate @issues %>
```

A teendők listájában a teendő határideje az adatbázisbeli dátumformátumot használja. Ezt átalakíthatjuk az `ActiveSupport` helperével irodalmi formátumúvá:

```
<%= issue.deadline.strftime("%d/%m/%Y %H:%M") %>
```

A nézetek tervezésekor létrehoztunk egy legördülő menüt a teendő más felhasználóhoz rendelése céljából. Valósítsuk meg az eseménykezelőt. Mivel most már rendelkezésünkre áll a legördülő menü mögött álló modell is, hozzárendelhetjük az aktuális nézet felhasználóit választható elemként. Az eseménykezelő a `reassign` akció, amelyet a teendő azonosítójával paraméterezett URL-en keresztül érünk el.

```
<fieldset>
  <legend>Actions</legend>
  <%= link_to 'Edit', edit_issue_path(@issue) %>
```

```
  <%= form_tag "/issues/reassign/#{@issue.id}", :method
      =>:post do %>
    <%= label_tag :responsible_user %>:
    <%= select("issue", "user_id", User.all.collect {|p
        | [p.username, p.id] }, {:include_blank=>true})
      %>
    <%= submit_tag "Reassign" %>
  <% end %>
</fieldset>
```

A teendők kontrollerben meg kell valósítanunk még a feladat-átrendelés akciót, amely a teendő felhasználó asszociációját módosítja egy másik létező felhasználó objektumra, végül átirányítja a kérést a teendő `show` oldalára.

```
def reassign
  p = params[:issue]
  responsible_user = User.find p['user_id']
  @issue = Issue.find params[:id]
  if !responsible_user.nil?
    @issue.user = responsible_user
    @issue.save
  end
  redirect_to issue_url(@issue)
end
```