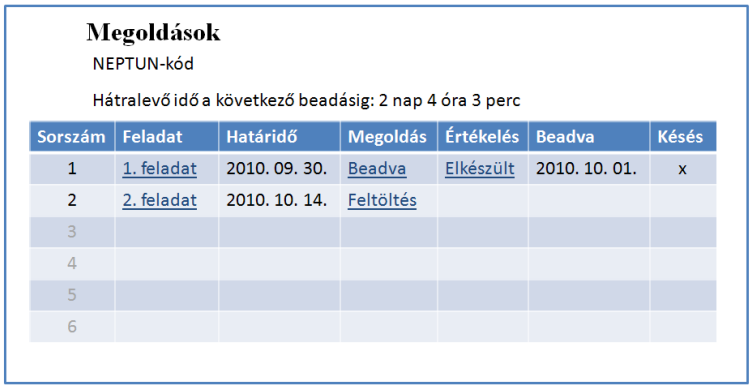
Rails MVC, session
Gyakorlat

Kovács Gábor

2010. október 29.

Az előző gyakorlaton megkezdett példát folytatjuk a megoldások beadásának megvalósításával. Az elképzelt képernyőképet a 1. ábra mutatja. A tábla adatai a tervünk alapján két modellből származnak, a feladatok modellből és a megoldások modellből. A feladatok modell adatai az összes `student` típusú felhasználóra vonatkoznak, a megoldások viszont felhasználónként egyediek.



1. ábra. Megoldások képernyő

Először hozzuk létre a feladatok modelljét, nézetét és kontrollerét. Ehhez használjuk a Rails `scaffold` parancsát, amely mindezt egyszerre megteszi. Az alábbi parancs egy `Task` modellt hoz létre egy olyan migrációval, ami

egy `number` nevű egészt, amely a feladat sorszámát adja meg, egy `url` nevű stringet, amely egy URL a feladat kiírására, és egy `deadline` nevű dátum objektumot tartalmaz, amely a feladat beadási határideje. Négy nézet jön létre a `Task`-ok listáját megjelenítő `index`, az új `Task`-ot létrehozó `new`, a `Task`-ok szerkesztésre használható `edit`, és az egy konkrét `Task` adatait megjelenítő `show`. A `new` és a `edit` egy HTML `form`-ot tartalmaz, amelyet a kontroller kezel le. A kontroller e négy akción kívül még három további akciót definiál: az új `Task` létrehozása esemény lekezelő `create` akciót, a frissítés lekezelő a `update` akciót, és a törlést megvalósító `destroy` akciót.

```
rails generate scaffold Task number:integer url:string
   deadline:datetime

rake db:migrate
```

Módosítsuk a nézetet úgy, hogy a feladatok szerkesztése, törlése csak az oktató szerepkörű felhasználók számára legyen lehetséges, és biztosítsuk a feladat beadás akciót a hallgató szerepkörű felhasználók számára. Ahogy az előző gyakorlaton láttuk, ezt úgy tudjuk megvalósítani, hogy a `session` hash-en keresztül szerzünk `user` objektumra referenciát. A `task_controller.rb`-hez az alábbi kódrészletet adjuk hozzá.

```
before_filter :find_user
private
def find_user
  @user = User.find_by_id(session[:user]) if session[:
     user]
end
```

A felhasználó referenciáját felhasználva a nézetben módosíthatjuk az elérhető akciók halmazát. Például a `index.html.erb`-ben az automatikusan generált linkek csak tesszük elérhetővé, ha a felhasználó nem `student` típusú. A `student` felhasználóknak elérhetővé tesszük a `Bead` műveletet, amelyet a hamarosan megírandó `solutions` kontroller `new` akciója valósít meg. A műveletnek a `params` hash-en keresztül átadjuk a beadni szándékozott `task` példány `id`-jét is.

```
<% if @user then
    if @user.student then %>
<td><%= link_to "Bead", :controller => "solutions", :
   action => "new", :id => task.id%></td>
<% else %>
...
```

```
<%end end %>
```

Az előző gyakorlaton módosítottuk a webalkalmazásunk általános kinézetét (`#application.html.erb`), egy menüt adtunk hozzá a layouthoz. Most ezt a menüt kibővítjük úgy, hogy az elérhetővé tegye a feladatok listáját a bejelentkezett felhasználók számára. A `if logged_in?` blokk törzsében elhelyezünk egy új linket, amely a feladatok `index` oldalára mutat.

```
<% if logged_in? %>
  <%= link_to "Feladatok", :controller => "tasks", :
      action => "index" %>
<% end %>
```

A feladatok táblázat adatainak másik részét a `Solution` modellből vesszük, amelyet most újból definiálunk.

```
rails g model Solution
```

A migrációban a `solutions` tábla oszlopainak a következőket adjuk meg. A tábla kapcsolótábla szerepét tölti be a `users` és a `tasks` táblák között, ezért két mindkét tábla kulcsát (`user_id` és `task_id`) tartalmazza. Ezen kívül megjegyezzük a beadott feladat verziószámát, a feltöltött fájl nevét, a beadás esetleges késedelmét és hiányát.

```
t.integer :user_id
t.integer :task_id
t.integer :version
t.string :filename
t.boolean :late, :default => false
t.boolean :missing, :default => true
t.timestamps
```

A megoldás modell osztályához (`Solution`) hozzáadunk egy-egy több-egy relációt a `belongs_to` metódussal a `User` és a `Task` modellekre.

```
belongs_to :user
belongs_to :task
```

Az adatbázis táblák közötti relációkat a `Task` és a `User` osztályban is reflektáljuk. Mindkettőhöz hozzáadunk egy egy-több relációt a `has_many` metódussal a `Solution` modellre vonatkozóan. A `task.rb` alábbi kódrészlete második sorában közvetett referenciát adunk meg a `User` modellre, amelyet a `solutions` kapcsolótáblán keresztül valósítunk meg. E referencia azt mondja meg, hogy az aktuális feladatra mely felhasználók adtak be megoldást.

```
has_many :solutions
has_many :users, :through => :solutions
```

A `user.rb` módosításas ezzel analóg módon történik. A `tasks` referencián keresztül a felhasználó az összes elérhető feladatot láthatja.

```
has_many :solutions
has_many :tasks, :through => :solutions
```

A migrációt elvégezve a Rails konzolon ellenőrizhetjük, hogy a relációk valóban létrejöttek az osztályok között is. A `User` osztály példányai rendelkeznek `solutions` és `tasks` getterekkel és setterekkel.

```
rake db:migrate
```

A feladat beadása fájlfeltöltéssel történik. Bár léteznek erre kész Rails függvénykönyvtárak, mi most egy fapados megoldás fogunk használni. A fájlfeltöltés logikailag a feladat beadáshoz kapcsolódik, így a `Solution` osztályban valósítjuk meg. Mielőtt megvalósítanánk a feltöltést létrehozunk egy könyvtárat a Rails keretrendszerben a webszerver adatait tartalmazó `public` könyvtár alatt, ahol a feltöltött fájlokat tárolni fogjuk, legyen ennek a neve mondjuk `data`. A fájlfeltöktés megvalósító statikus `saveFile` nevű metódus egy hash-t vár paraméterül, amelynek van egy `file` kulccsal azonosított értéke. A metódus törzsének első sora az előbbi könyvtár abszolút elérési útját definiálja. A második sor ezt összefűzi az `upload` hash `file` kulccsal jelölt elem eredeti fájlnevével. A harmadik sor megnyitja a második sorban megadott elérési úttal reprezentált fájlt írásra, miközben a `open` metódus blokkjában azonnal kiírja az `upload` hash `file` kulcsú elemének tartalmát.

```
def saveFile(upload)
  dir="/home/kovacsg/feladat3/public/data" #elotte
     letrehozni
  path = File.join(dir,File.basename(upload['file'].
     original_filename))
  File.open(path,"w") { |f| f.write(upload['file'].read
     ) }
end
```

A megoldások leírására alkamas modellünket létrehoztuk, készítsük el hozzá a nézetet és a kontrollert! A kontrollernek legyen egy akciója, az új megoldást feltöltő `new`.

```
rails g controller solutions new
```

A `new.html.erb` fájlban definiált fájlfeltöltési nézet egy `form`-ot tartalmaz, amelyre a `:upload` szimbólummal hivatkozunk, és eseményét a kontroller `create` metódusával kezeljük le. Az esemény egy HTTP POST üzenet, amelyben a HTML `form`-ot el kell látnunk a `enctype="multipart/form-data"` attribútummal jelezvén, hogy fájlt töltünk fel, ezt a `:html => {:multipart => true }` hash érték adja meg. A `form` maga egy fájlkiválasztó mezőt (`file_field`) és egy nyomógombot tartalmaz, amelynek felirata attól függ, hogy a felahsználó adott-e már be feladatot. Ennek eldöntésére egy `submitted?` nevű segédmetódust definiálunk a kontrollerben.

```
<h1>Feladat beadás<h1>
<% form_for(:upload,:url=>{:action=>"create"}, :method
   =>"post", :html => {:multipart => true }) do |form|
   %>
<p><label for="upload">Megoldás feltöltése</label> : <
   %= form.file_field('file') %></p>
<% if submitted? then %>
<%= submit_tag "Módosít" %>
<% else %>
<%= submit_tag "Bead" %>
<% end %>
<% end %>
```

A kontroller osztályban még mielőtt megvalósítanánk a feladat beadását lekezelő `create` metódust inicializáljuk annak a felhasználónak és annak a feladatnak a példányváltozóját, amelyekre a beadás vonatkozik. Minden egyes metódus meghívása előtt végrehajtjuk a `before_filter` metódusnak átadott metódusnév által definiált függvényt, a privát `find_user_and_task`-ot. Ez a metódus a `session` hash `:user` kulcsa alapján kikeresi az aktuális felhasználót, és a `params` hash `:id` kulcs alapján kikeresi az aktuális feladatot.

```
class SolutionsController < ApplicationController
  before_filter :find_user_and_task

  def index
    render :file=>'app/views/solutions/new.html.erb'
  end

  def new
  end
```

```
  private
  def find_user_and_task
    @user=User.find_by_id(session[:user]) if session[:
        user]
    @task=Task.find_by_id(params[:id]) if params[:id]
  end
end
```

A nézetben felhasználtunk egy segédmetódust a kontrollerből, a `submitted?`-et. Ezt segédmetódusként deklaráljuk a `helper_method` metódussal. A metódus törzse a `solutions` kapcsolótáblából kikeresi, hogy a `@user` és `@task` példányváltozók `id`-ihez tartozik-e megoldás.

```
class SolutionsController < ApplicationController
  helper_method :submitted?

  protected
  def submitted?
    solution = Solution.find_by_user_id_and_task_id(
        @user.id,@task.id)
    solution.nil?
  end
end
```

A fájlfeltöltés akciót lekezelő `create` metódus akkor hajtódik végre, ha van feltöltött fájl, a felhasználót azonosítottuk a session-ből, és a feladatot is megállapítottuk. A metóduk először meghívja a `Solution` modell statikus `saveFile` nevű metódusát a POST üzenet `:upload` paraméterével. Ezután az adatbázisban már korábban beadott feladatot keres. Ha nem talált, akkor inicializál egyet. Ezután növeli a verziószámot, módosítja a késés és a hiány attribútumokat, és beállítja a feltöltött fájl nevét. Végül elmenti az adatbázisba az objektumot.

```
class SolutionsController < ApplicationController
  def create
    if !params[:upload].nil? && !@user.nil? && !@task.
        nil?
      Solution.saveFile(params[:upload])
      solution = Solution.find_by_user_id_and_task_id(
          @user.id,@task.id)
      if solution.nil?
        solution = Solution.new
        solution.user_id=@user.id
```

```
            solution.task_id=@task.id
            solution.version=0
          end
          solution.version=solution.version+1
          solution.missing=false
          solution.late=false
          upload=params[:upload]
          solution.filename=File.basename(upload['file'].
              original_filename)
          solution.save
        end
      end
    end
```