HTML és Rails
Gyakorlat

Kovács Gábor

2015. március 17.

A gyakorlat célja, hogy kialakítsuk a félév során megoldandó feladat képernyőit HTML-ben, ahol lehet Rails metódusok felhasználásával.

Az első lépés a webalkalmazásunk keretének kialakítása ezt a nézetek között az alkalmazásszintű nézetben, vagyis a `layouts/application.html.erb`-ben tehetjük meg. Rendezzük el úgy az oldalunkat, hogy legyen benne egy fejrész, egy központi rész, amely bal oldalon egy keskeny menüsávból áll, jobb oldalon a tartalomból, és egy lábrész. Az elrendezést `div`-ekkel valósítjuk meg, mindegyikhez egyedi `id`-t rendelve.

```
<div id="page">
<div id="header"></div>
<div id="menu"></div>
<div id="main">
<%= yield %>
</div>
<div id="footer"> Copyright, RoR, 2015</div>
</div>
```

Második lépésként készítsük el az oldal stíluslapját, amivel ezek a helyükre kerülnek, és helyezzünk el benne minimális mennyiségű formázási információt. Az oldal legyen 1024 pixel széles. A fejrész legyen világosszürke és 100 képpont magas. A menüsávot a fejléc alatt helyezzük el, és az vízszintesen a szélesség 24%-át foglalja el, és legyen 600 pixel magas. Az oldal tartalmi része legyen 600 képpont magas, világosszürke háttérrel rendelkezzen, és a menütől jobbra helyezkedjen el a vízszintesen a szélesség 76%-át elfoglalva. A lábrészben a szöveget igazítsuk középre.

```
#page {
    width : 1024px;
}
#header {
```

```
    background-color: #dddddd;
    height: 100px;
}
#menu {
    background-color: #cccccc;
    height: 600px;
    width: 24%;
    float: left;
}
#main {
    background-color: #eeeeee;
    height: 600px;
    width: 76%;
    float: left;
}
#footer {
    background-color: #dddddd;
    text-align: center;
    clear: both;
}
```

Háromféle felhasználóra készülünk fel egyelőre, egy látogatóra, és egy belépett felhasználóra, aki korábban keresztülment egy regisztrációs folyamaton, és egy adminisztrátorra. A látogató csak böngészhet, ugyanakkor regisztrálhat. A bejelentkezett felhasználó vagy az adminisztrátor számára több funkciót is elérhetővé teszünk.

Kezdjük a látogató menüjével. Helyezzünk el a menüben egy a belépést lehetővé tevő formot. Ezt részleges rendereléssel tesszük meg. A formot tartalmazó fájl alkalmazás szinten kezeljük, ezért a `layouts` könyvtárban helyezzük el, így a be nem jelentkezett felhasználó bármelyik oldalon bejelentkezhet. A form tartalmazzon egy felhasználónévre utaló címkét és egy szövegbeviteli mezőt, valamint egy a jelszóra utaló címkét és jelszóbeviteli mezőt, továbbá egy Login feliratú nyomógombot. A formot a `form_tag` Rails helperrel valósítjuk meg, aminek első paramétere a formot kezelő URL, vagyis a form `action` attribútuma, illetve adjuk meg, hogy HTTP POST-tal kívánjuk elküldeni. A form mezőit rendre a `label_tag`, `text_field_tag`, `password_field_tag` és `submit_tag` helperekkel hozzuk létre, és a beviteli mezőket 20 hosszúra korlátozzuk. A be nem jelentkezett felhasználónak tegyük lehetővé az elfelejtett jelszó visszaszerzését, ezt egy link hozzáadásával tesszük meg.

```
<fieldset>
  <legend>Login</legend>
```

```
  <%= form_tag 'sessions/create', :method => :post do %>
    <%= label_tag 'username', "Username" %><br/>
    <%= text_field_tag 'username', '', :size =>20 %> <br/>
    <%= label_tag 'password', "Password" %><br/>
    <%= password_field_tag 'password', '', :size =>20 %> <
        br/>
    <%= submit_tag "Login" %>
  <% end %>
</fieldset>
<%= link_to "Register", '/users/new' %><br/>
<%= link_to "Forgotten␣password", '/users/forgotten' %><br/
    >
```

Ezután a `menu` azonosítójú `div`-ben meghivatkozhatjuk ezt az oldalt. A Rails konvenció szerint az aláhúzásjelet el kell hagynunk.

```
<%= render partial:"layouts/menu" %>
```

Modellezzük azt az esetet is, amikor egy felhasználó már bejelentkezett. Ezt egy saját helper metódussal tesszük meg, amit a `helpers/application_helper` állományba helyezünk el. Itt egyelőre manuálisan állítjuk, hogy be van-e jelentkezve a felhasználó. A metódus értelemszerűen boolean visszatérési értékű. A belejentkezett felhasználóról pedig egy-egy függvénnyel eldöntjük, hogy felhasználó vagy adminisztrátor-e.

```
module ApplicationHelper
  def logged_in?
    true
  end

  def is_student?
    false && logged_in?
  end

  def is_admin?
    !is_student? && logged_in?
  end
end
```

Ezt visszavezetve a keretbe a menüt megvalósító `menu`-ban a következő módosítást végezzük el. Így a helper módosításával be, illetve ki tudunk lépni az oldalról. „Jelentkeztessük” be és ki a felhasználót, hogy ellenőrizhessük, hogy a megfelelő menü jelenik-e meg a vendég és a felhasználó számára.

```
<% if logged_in? %>
```

```
  ...
<% else %>
  ...
<% end %>
```

Nézzük meg a be nem lépett felhasználó regisztációs folyamatát! Az előző gyakorlat alkalmával már létrehoztuk a felhasználó modellünk kezdetleges változtatás, így arra már tudhatunk hivatkozni egy Rails formban, amely az MVC tervezési minta szerint szorosan kapcsolódik a nézethez. Hozzuk létre a felhasználó nézetét és fontosabb akcióit a következő paranccsal:

```
rails generate controller users new edit show forgotten
```

A parancs futtatásával létrejött az `users` kontroller és a hozzá kapcsolódó nézetek köztük az új felhasználó létrehozását lehetővé tevő `new`, a felhasználói profil szerkesztését megvalósító `edit`, a felhasználói adatait megjelenítő `show`, és a hiányzó jelszót pótló `forgotten` felület, illetve akció. Tervezői kérdés, hogy az elfelejtett jelszó kezelését a felhasználók kontrollere részének tekintjük, vagy önálló kontrollert hozunk létre számára. A gyakorlaton amellett döntöttünk, hogy az elfelejtett jelszó kerüljön a felhasználók kontrollenébe.

Hozzunk mindjárt létre a regisztrációs nézetet! Legyen egy címsorunk, amely elmondja a felhasználónak, hogy melyik oldalon van. Az esetleges hibaüzeneteknek tartsunk fenn helyet. Ezután egy `fieldset`-ben definiáljuk egy formot, amely ez esetben egy konkrét, létező modellhez van kötve. Ezt a `form_for` Rails helperrel tehetjük meg. Ennek első paramétere a modell neve szimbólum formájában, második paramétere a formhoz kötött akció, amely legyen a `users` kontroller `create` akciója a konvenciót követve. A metódus blokkjának van egy paramétere a `form`, amin kereszül definiáljuk fogjuk az űrlap elemeit. Legyen a hat elem rendre a következő: egy húsz karakter széles felhasználónévre vonatkozó szövegbeviteli mező a hozzá kapcsolódó címkével, egy-egy harminc karakter széles névre, illetve email címre vonatkozó szövegbeviteli mező a hozzá kapcsolódó címkével, két darab 20 karakter széles jelszóbeviteli mező a hozzájuk kapcsolódó címkével, a két jelszómező eltérő azonosítóval rendelkezzék, az egyik prefixe `_confirmation`-re végződjék, és végül egy nyomógomb. Az oldal alján helyezzünk el egy visszalépés gombot azon felhasználóknak, akik véletletül tévedtek ide. A `:back` szimbólummal hivatkozott értékben a Rails a `HTTP_REFERER` HTTP fejléc elem vagy a Javascript `history` attribútuma alapján tárolja a megelőző HTTP kérésben használt URI-t.

```
<h1>Registration</h1>

<div>
```

```
  <fieldset>
    <legend>Register a new user</legend>
    <%= form_for @user, :url => { :action => 'create' } do
        |form| %>
    <div>
      <%= form.label :username %>:
      <%= form.text_field :username, :size=>20 %>
    </div>
    <div>
      <%= form.label :name %>:
      <%= form.text_field :name, :size=>30 %>
    </div>
    <div>
      <%= form.label :email %>:
      <%= form.text_field :email, :size=>30 %>
    </div>
    <div>
      <%= form.label :password %>:
      <%= form.password_field :password, :size=>20 %>
    </div>
    <div>
      <%= form.label :password_confirmation %>:
      <%= form.password_field :password_confirmed, :size=>
          20 %>
    </div>
    <%= form.submit "Register" %>
    <% end %>
  </fieldset>
</div>

<%= link_to "Back", :back %>
```

A létrejött oldal HTML forrását tekintve a következőt látjuk. A formok mezőinek `name` és `id` attribútuma tartalmazza a modell nevét és a mező nevét. A név attribútum Ruby hash mintájára készült el, a modell nevének hash-ére hivatkozik a mező Rails forrásban megadott neve. Az általunk megadott mezőkön kívül létrejött két hidden mező is, amelyek a form használójának hitelesítését hivatottak ellenőrizni. A visszalépés itt JavaScripttel valósul meg. A forrást megtekintve láthatjuk, hogy a `:password_confirmation` szimbólumból a Rails automatikusan a *Password confirmation* szöveget állította elő. A stringek és a szimbólumok így ezen elv mentén felcserélhetők a form helperek argumentumlistájában.

A form eseményét a Rails konvenció szerint a `create` kontroller metódus

fogja kezelni. Ez még nem létezik, ezért definiáljuk azt egyelőre üres törzzsel.

A felhasználói profil szerkesztésének nézetében (`edit.html.erb`) található form szinte teljesen megegyezik az új felhasználót létrehozó formmal. A felhasználónév módosítását kell inaktívvá tennünk, illetve az eseménykezelő kontroller akciót kell módosítanunk. A form eseményét a Rails konvenció szerint a `update` kontroller metódus fogja kezelni. Ez még nem létezik, ezért definiáljuk ezt is egyelőre szintén üres törzzsel. Ezen kívül a nézetben a feliratokat kell még átírnunk regisztrációról profil szerkesztésére.

```
<h1>Profile</h1>

<div>
  <fieldset>
    <legend>Edit user profile</legend>
    <%= form_for @user, :url => { :action => 'update', :id
        => @user.id } do |form| %>
    <div>
      <%= form.label :username %>:
      <%= @user.username %>
    </div>
    ...
    <%= form.submit "Update" %>
    <% end %>
  </fieldset>
</div>

<%= link_to "Back", :back %>
```

Mivel a `form_for` Rails helper metódust használtuk a form létrehozására a `new` és az `edit` nézetekben, ezért szükséges a megfelelő kontroller akciókban a `@user` példányváltozó inicializálása. Ezeket egyelőre ne adatbázisból tegyük meg, hanem statikus tartalommal töltsük fel. Míg a `new` esetén a felhasználó még nem lézetik az adatbázisban, attribútumai inicializálatlanok, ezért elégséges egy frissen létrehozott példány használata, addig az `edit` esetén már ki kell töltenünk a struktúra mezőit beleértve az adatbázisbeli azonosító `id` attribútumot is.

```
def new
  @user = User.new
end

def edit
    @user = User.new
    @user.id = 1
```

```
    @user.username = 'valaki'
    @user.name = 'Vala Ki'
    @user.email = 'valaki@mail.bme.hu'
    @user.password = 'titok'
end
```

A felhasználói profilon legyen megtekinthető a felhasználó neve, felhasználóneve és email címe.

```
<h1>Show user profile</h1>
<p>Name: <%= @user.name %></p>
<p>Username: <%= @user.username %></p>
<p>Email: <%= @user.email %></p>
```

A `show` nézet számára inicializálnunk kell a `@user` példányváltozót.

```
def show
    @user = User.new
    @user.id = 1
    @user.username = 'valaki'
    @user.name = 'Vala Ki'
    @user.email = 'valaki@mail.bme.hu'
end
```

Ezután alakítsuk ki az elfejetett jelszó oldalt is. Itt egyszerűbb a formunk a beléptetésnél, csak az email címet tartalmazza.

```
<h1>Forgotten password</h1>

<div>
<fieldset>
  <legend>Please give your email address</legend>
  <%= form_tag 'users/send_forgotten' do %>
    <%= label_tag 'email', "Email address" %><br/>
    <%= text_field_tag 'email', '', :size=>30 %>
    <%= submit_tag "Send" %>
  <% end %>
</fieldset>
</div>

<%= link_to 'Back', :back %>
```

Az elfelejtett jelszó kiküldését a form eseményét kezelő kontroller akció, a `send_forgotten` teszi majd meg, amit fel kell vennünk a kontroller osztályába egyelőre üres törzzsel. Ha a felhasználó meggondolná magát, és megsem

kívánná elküldetni magának a jelszavát, egy `Back` feliratú linkkel biztosítjuk számára a lehetőséget az előző oldalra való visszatérésre.

```
<h1>Request forgotten password</h1>

<fieldset>
<legend>Please give your user name</legend>
<%= form_for @user, :url => { :action => 'send_forgotten'
    }, :method => :post do |f| %>
 <div>
    <%= f.label :username %>
    <%= f.text_field :username, :size => 14 %>
 </div>
 <%= f.submit 'Send' %>
<% end %>
</fieldset>

<%= link_to 'Back', :back %>
```

A formok eseménykezelőihez a generate controller parancsunk nem generált útvonalakat, ezért azokat – egyelőre paraméterek nélkül – felvesszük a `config/routes.rb`-be:

```
post 'users/create'
post 'users/update'
post 'users/send_forgotten'
```

Térjünk át a belépett felhasználó használati eseteire. Módosítsuk az alkalmazás szintű helperünk visszatérési értékét `true`-ra, és rendereljünk a belépett felhasználó menüjébe is használható linkeket. Ilyen például a profil szerkesztése, a feladatbeadás, a beadott megoldás módosítása, illetve a kilépés. Ezen funkciókhoz hozzunk létre egy-egy linket a menüben. A kilépés művelet mutasson a még nem létező `sessions` kontroller `destroy` akciójára. A felhasználói profil szerkesztéséhez használjuk a `edit` akciót, ennek azonban át kell majd később adnunk a bejelentkezett felhasználó azonosítóját, ami még nem áll a rendelkezésünkre, így e link még inaktív lesz. Az adminisztrátor számára lehetővé tesszük az új feladat nézetre való navigációt és a feladat szerkesztését, míg felhasználók számára a feladatok index nézetére való ugrást, ahol a megoldás feltölthető.

```
Hello, world!<br/>
<%= link_to "Profile", '/users/edit' %><br/>
<% if is_admin? %>
<%= link_to "New task", '/tasks/new' %><br/>
<% end %>
```

```
<% if is_admin? %>
<%= link_to "Edit task", '/tasks/edit' %><br/>
<% elsif is_student? %>
<%= link_to "Submit task", '/tasks/' %><br/>
<% end %>
<%= link_to "Logout", '/sessions/destroy' %>
```

Hozzuk létre a feladatok modelljét és nézeteit! A strukturált megjelenítésért csaláshoz folyamodunk, és mindezt egyetlen paranccsal, a `scaffold`-dal tesszük meg. Ez létrehozza a modell osztályt, a kapcsolódó adatbázis-migrációt, a tesztadatok fájlját, a modell egységtesztjeit, a kontroller osztályát, a kapcsolódó nézetek kezdeti változatát, a kontrollerek funkcionális tesztjeit, illetve a nézetekhez kapcsolódó útvonalakat. A parancs úgy nevezett resourceful útvonalakat definiál a `routes.rb`-ban, amelyek eltérnek az eddig megszokottaktók (parancssoron a `rake routes` paranccsal nézhetjük meg, hogy mi változott). A feladatok adatstruktúrája tartalmazza a feladat sorszámát, a feladat határidejét, a feladat leadási idejét és értékelését. Az utóbbi két attribútumot a következő gyakorlaton átmozgatjuk egy beadás modellbe. A parancs után hajtsuk végre az adatbázis migrációját, és nézzük meg az eredményt!

```
rails generate scaffold Task number:integer deadline:
    datetime submitted_at:datetime status:string
```

```
rake db:migrate
```

Négy nézet jött létre: `index`, `new`, `edit`, `show`, amelyek egyszerű táblázatos formában jelenítik meg az összes vagy az azonosító HTTP kérés paraméterben megadott modell példány adatait. A nézetekhez kapcsolódó kontroller akciókon kívül további három metódus jön létre a kontrollerben: `create`, `update`, `destroy`. Ezek együttesen megfelelnek a RESTful HTTP filozófiájának.

Nézzük meg a létrejött nézeteket! Az `index` nézet egy fejléccel rendelkező táblázatban megjeleníti a `Task` osztály példányait külön sorokban, az oszlopokban az attribútumok találhatók. A táblázatban három extra oszlop található fejléc nélkül, amelyek a `show`, a `edit` nézetre mutató linkeket, illetve egy az objektumok törlő, a `destroy` akcióra mutató linket tartalmaz. E három link paraméterezve van a feladat azonosítójával, így az mindig a táblázat megfelelő sorában megjelenő feladatra vonatkozik. Az oldal alján egy új feladat létrehozását lehetővé tevő nézetre mutató link található. E linkeket csak az adminisztrátor számára tesszük elérhetővé. A felhasználók egy fájlfeltöltés formot láthatnak helyettük, a vendég felhasználók pedig semmit.

A `new` és az `edit` nézetek egy linktől és egy címkétől eltekintve azonosok, mindkét nézet megjeleníti a `form` nevű részleges nézetet. A `form` nézet egymás utáni `<div>` elemekben a `Task` adatstruktúrája mezői típusainak megfelelő adatbeviteli mezők jelennek meg egy a mező nevével megegyező címke után. Minden típus előre definiált adatbeviteli mezőre képeződik le, így az időpont öt darab legördülő menüre, a string és az egész szövegbeviteli mezőre. Az oldalak tetején megjelenhetnek az esetleges hibaüzenetek. Az `edit` nézetet módosítsuk annyiban, hogy az adminisztrátor szerkeszthesse csak a feladat adatait, és a felhasználók számára tegyünk lehetővé fájlfeltöltést.

```
<% if is_admin? %>
<%= render 'form' %>
<% elsif is_student? %>
<fieldset>
  <legend>Submit task</legend>
  <%= form_for @task, :url =>
    { :action => 'create', :uid => @user.id, :id => @task.id
        } do |form| %>
    <%= form.label :file %>:
    <%= form.file_field :file %>
    <%= form.submit "Upload" %>
  <% end %>
</fieldset>
<% end %>
```

A `show` nézet a feladat adatstruktúra mezőinek értékeit írja ki.

Töltsük fel az `index` nézet táblázatát adatokkal, a nézetben táblázatunk törzsét kiíró `for` ciklusban láthatjuk, hogy a `@tasks` kontroller példányváltozó elemein iterálunk végig. A kontrollerben megnézve e változót, azt látjuk, hogy ez az adatbázisból betölti az összes rekordját, azonban mivel jelenleg nincsenek ott adataink, inicializáljuk másképpen e változót.

```
def index
    #@tasks = Task.all
    task1 = Task.new
    task1.id = 1
    task1.number = 1
    task1.deadline = Time.now
    task2 = Task.new
    task2.id = 2
    task2.number = 2
    task2.deadline = Time.now
    @tasks = [ task1, task2 ]
end
```

Hasonlóan az `edit` és a `show` nézetek `@task` kontroller példányvátozóit is inicializáljuk a `set_task` nevű privát metódusban, majd ellenőrzizhetjük, hogy megjelennek-e a nézeten ezek az adatok. Az `edit` nézet esetén azt látjuk, hogy az adatstruktúra mezői értéke alapján inicializálódtak a form adatbeviteli mezői..

```
def set_task
  #@task = Task.find(params[:id])
    task1 = Task.new
    task1.id = 1
    task1.number = 1
    task1.deadline = Time.now
    @task = task1
end
```

Hivatkozások