

HTML és Rails

Gyakorlat

Kovács Gábor

2012. március 14.

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="header">
Header
</div>
<div id="menu">
</div>
<div id="main">
  <%= yield %>
</div>
<div id="footer">
  Copyright , RoR 2012
</div>
```

Második lépésként készítsük el az oldal stíluslapját, amivel ezek a helyükre kerülnek, és helyezzünk el benne minimális mennyiségű formázási információt. Az oldal szélessége legyen 800 képpont. A fejrész legyen világosszürke és 100 képpont magas. A menü a szélesség 24%-át töltse ki, legyen szürke háttérű, 600 képpont magas, balra igazított. Az oldal központi része legyen szintén 600 képpont magas, az oldal szélességének 76%-t töltse ki, világosszürke háttérrel rendelkezzen, és legyen szintén balra igazított. A lábrészben a szöveget igazítsuk középre, és legyen a lábrész magassága 50 képpont.

```
#page {
width: 800px;
}

#header {
background-color: #dddddd;
height: 100px;
}

#menu {
background-color: #d0d0d0;
height: 600px;
width: 24%;
float: left;
}

#main {
background-color: #ffffff;
height: 600px;
width: 76%;
float: right;
}

#footer {
height: 50px;
clear: both;
text-align: center;
}
```

Kétféle felhasználóra készülünk fel egyelőre, egy látogatóra, és egy belépett felhasználóra, aki korábban kereszülment egy regisztrációs folyamaton. A látogató csak böngészhet, ugyanakkor regisztrálhat. A bejelentkezett felhasználó számára több funkciót is elérhetővé teszük. 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. A Rails konvenció szerint a részlegesen renderelt állományok neve aláhúzásjellel kezdődik. Legyen a fájlunk neve ezért `_layouts.html.erb`! A formot ágyazzuk egy `fieldset`-be `Login` fejléccel, és tartalmazzon egy felhasználónévre utaló címkét és szövegbeviteli mezőt és 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 a regisztrációt és az elfelejtett jelszó visszaszerzését, ezt két 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' %>
<%= link_to "Forgotten_password", '/users/forgotten' %>
```

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

```
<%= render 'layouts/loginform' %>
```

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.

```
module ApplicationHelper
  def logged_in?
    false
  end
end
```

Ezt visszavezetve a keretbe a menü `div`-ben a következő módosítást végezzük el. Így a helper módosításával be, illetve ki tudunk lépni az oldalról.

```
<% if logged_in? %>
  Hello user
<% else %>
  <%= render 'layouts/loginform' %>
<% end %>
```

Nézzük meg a be nem lépett felhasználó regisztrációs 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 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 `show`, és a hiányzó jelszót pótló `forgotten` felület, illetve akció.

Hozzuk mindjárt létre a regisztrációs nézetet! Legyen egy címsorunk, amely elmondja a felhasználónak, hogy melyik oldalon van. Az esetleges hibüzeneteknek tartunk 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 keresztül definiáljuk fogjuk az űrlap elemeit. Legyen az öt elem rendre a következő: egy húsz karakter széles felhasználónév szövegbeviteli mező a hozzá kapcsolódó címkével, egy email cím szövegbeviteli mező a hozzá kapcsolódó címkével, két darab jelszóbeviteli mező a hozzájuk kapcsolódó címkével, a két jelszómező eltérő azonosítóval rendezzék, és végül egy nyomógomb. Az oldal alján helyezünk el egy visszalépés gombot azon felhasználóknak akik véletlenül tévedtek ide. A visszalépés Railsben a `:back` URL-lel lehetséges, ami vagy a HTTP fejrészből kivett hivatkozó oldalra mutat, vagy JavaScripttel valósul meg.

```
<h3>Registration</h3>
<%= flash[:notice] %>

<div>
  <fieldset>
    <legend>Register a new user</legend>
    <%= form_for :user, :url => { :controller => "users",
      :action => "create" } do |form| %>
      <div>
        <%= form.label :username %><br />
        <%= form.text_field :username, :size=>20 %>
      </div>
      <div>
        <%= form.label :email %><br />

```

```

    <%= form.text_field :email %>
  </div>
  <div>
    <%= form.label :password %>:<br />
    <%= form.text_field :password , { :size=>20 } %>
  </div>
  <div>
    <%= form.label :password_confirmation %>:<br />
    <%= form.text_field :password_confirmation ,
      :size=>20 %>
  </div>
  <%= submit_tag "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.

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.

```

<fieldset>
  <legend>Register a new user</legend>
  <form accept-charset="UTF-8" action="/users/create"
    method="post"><div style="margin:0;padding:0;display
  :inline"><input name="utf8" type="hidden" value="&#
  x2713;" /><input name="authenticity_token" type="
  hidden" value="iUr0v03mfCcW+h2t9CRHIPnje0IYvfmQJiUp/
  wyG6f4=" /></div>
  <div>
    <label for="user_username">Username</label>:<br />
    <input id="user_username" name="user[username]"
      size="20" type="text" />
  </div>
  <div>
    <label for="user_email">Email</label>:<br />

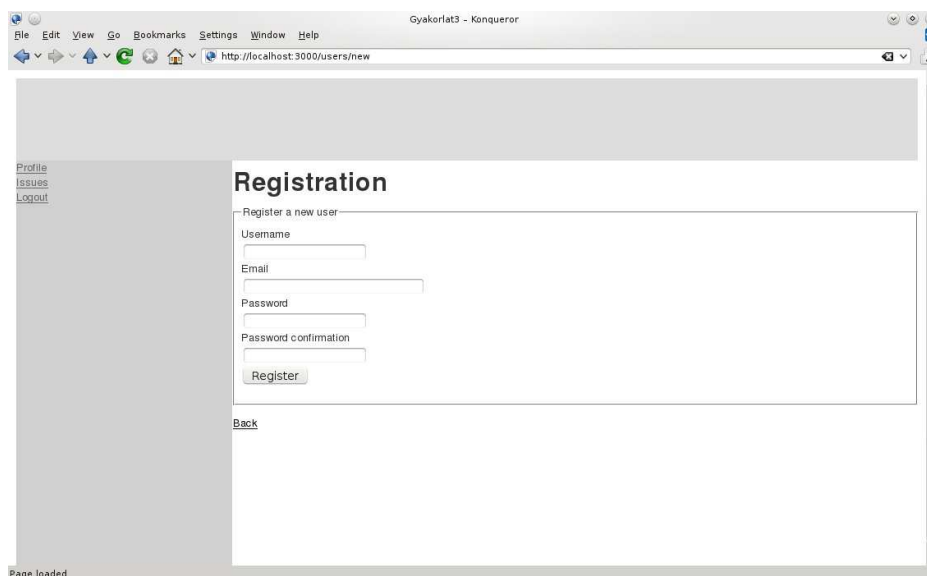
```

```

        <input id="user_email" name="user [email]" size="30"
            type="text" />
    </div>
    <div>
        <label for="user_password">Password</label>:<br />
        <input id="user_password" name="user [password]"
            size="20" type="password" />
    </div>
    <div>
        <label for="user_password_confirmation">Password
            confirmation</label>:<br />
        <input id="user_password_confirmation" name="user [
            password_confirmation]" size="20" type="password
            " />
    </div>
    <input name="commit" type="submit" value="Register" /
    >
</form> </fieldset>
</div>
<a href="javascript:history.back()">Back</a>

```

Az 1. ábra a regisztrációs oldal képernyőképét mutatja. Láthatóan sikerült az elrendezést megvalósítanunk, és a formokat létrehozunk.



1. ábra. A létrehozott regisztrációs oldal

Az új analógiájára hozzuk létre az elfelejtett jelszó oldalt is. Itt egyszerűbb

a formunk, csak az email címet tartalmazza, és a címkék különböznek az előző példához képest.

```
<fieldset>
  <legend>Forgotten password</legend>
  Please, give your email address<br />
  <%= form_tag '/users/send_reminder', :method => :post do
    %>
    <%= text_field_tag 'email' %>
    <%= submit_tag "Reminder" %>
  <%= end %>
</fieldset>
<%= link_to "Back", :back %>
```

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 kilépés vagy az új megjegyzés felvitele. Az első a `users` kontroller `edit` akciójára és nézetére mutat, a második az `issues` kontroller `new` akciójára és nézetére, a harmadik a `sessions` kontroller `destroy` akciójára. Az utóbbi kettőhöz új modellt és kontrollert is létre kell hoznunk.

```
<%= link_to "Profile", url_for(:controller=>"users",
  :action=>"edit") %><br />
<%= link_to "Issues", "/issues" %> <br />
<%= link_to "Logout", url_for(:controller=>"sessions",
  :action=>"destroy") %> <br />
```

A profil szerkesztése lényegében megegyezik az új felhasználó létrehozásával, vagyis a regisztrációval a különbség a feldolgozó akcióban és a címkékben áll.

```
<h3>Profile</h3>
<%= flash[:notice] %>

<div>
  <fieldset>
    <legend>Edit user profile</legend>
    <%= form_for :user, :url => {:action =>"update"}
      do |form| %>
      <div>
        <%= form.label :username %><br />
        <%= form.text_field :username, :size=> 20 %>
      </div>
    </div>
```

```

    <%= form.label :email %><br />
    <%= form.text_field :email %>
  </div>
  <div>
    <%= form.label :password %><br />
    <%= form.password_field :password, :size=> 20 %>
  </div>
  <div>
    <%= form.label :password_confirmation %><br />
    <%= form.password_field :password_confirmation,
      :size=> 20 %>
  </div>
  <%= submit_tag "Update" %>
<% end %>
</fieldset>
</div>
<%= link_to "Back", :back %>

```

A teendők modelljét, kontrollerét és nézetét egyetlen paranccsal hozzuk létre. A `scaffold` automatikusan generálja a modellt és a migrációt a paraméterlistában megadott azonosítók és típusok alapján. Úgy nevezett resourceful útvonalakat definiál a `routes.rb`-ban, amelyek eltérnek az eddig megszokottaktól (parancssoron a `rake routes` paranccsal nézhetjük meg, hogy mi változott). Négy nézetet hoz 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. Ezen kívül létrejönnek a modell és kontroller generálásnál már megszokott helper modulok és a teszt infrastruktúra.

```

rails generate scaffold Issue label:string \
  description:text priority:integer \
  deadline:datetime status:integer

```

Hajtsuk végre az adatbázis migrációt!

```

rake db:migrate

```

Legyen a teendők tábla a következő. Legyen a teendőknek egy címkéje (`label`), ami a teendő rövid definícióját tartalmazza, ez egy string típusú attribútum. Legyen egy szöveg típusú `description` attribútum, ami a teendő részletes leírását tartalmazza. A teendőnek legyen egy prioritás, amit egy egy és öt közötti egész számmal ábrázolunk. A teendő lezárásának időpontja

`datetime` típusú, legyen az attribútum neve `deadline`. A teendő készültségi állapotát szintén egész típusú számmal reprezentáljuk a `status` attribútumban. Ezen kívül a szokásos időpecsétek automatikusan generálódnak.

```
class CreateIssues < ActiveRecord::Migration
  def change
    create_table :issues do |t|
      t.string :label
      t.text :description
      t.integer :priority
      t.datetime :deadline
      t.integer :status

      t.timestamps
    end
  end
end
```

A `/issues` útvonalon elérhető teendők listája nézet egy táblázatot tartalmaz, ahol minden egyes modell attribútumhoz egy oszlop tartozik. A teendők a táblázat soraiban jelennek meg, ez meg is felel az elképzeléseinknek. Minden egyes teendőre három művelet jött létre automatikusan, a `Show`, amely a `show` nézetre navigál minket, az `Edit`, amely az `edit` nézetre visz, és a `Destroy`, amely törli az adatbázisból és táblázatból az aktuális sort egy Javascripttel megvalósított felugró ablakon való megerősítés után. Az oldal alján egy link található az új teendő felvétele oldalra. Az `edit` műveletet tegyük átmenetileg elérhetetlenné!

Teendő létrehozása (`new`) és szerkesztése (`edit`) ugyanazon form beágyazott renderelésével történik. A form a modell minden egyes attribútumára a típusának megfelelő szerkesztőmezőt tartalmaz. A `string` és egész típushoz szövegbeviteli mező, a `szöveg` típushoz szövegmező, a dátum típushoz öt legördülő menükből összerakott beviteli mező tartozik.

Módosítsuk az automatikusan generált teendők listájának nézetét (`index`), adjunk hozzá egy szűrőket tartalmazó részt. Definiáljuk egy szűrési feltételt a prioritásokra vonatkozóan. A form vonatkozzon az `index` akcióra, és helyezzünk el benne egy címkét egy legördülő menüt, amely az `1..5` tartományból vehet fel értékeket, és tartalmazza az üres értéket, valamint egy nyomógombot!

```
<fieldset>
  <legend>Filter</legend>
  <%= form_tag "/issues", :method => :post do %>
    <%= label_tag :priority %>
```

```

    <%= select_tag :priority, options_for_select
      ([1,2,3,4,5]), :include_blank=>true %>
    <%= submit_tag "Filter" %>
  <% end %>
</fieldset>

```

Egészítsük ki a **show** nézetet is! Tegyük lehetővé egy-egy formmal a teendő átrendelését, a csatolmányok menedzsmentjét és kommentek a teendőhöz fűzését. A csatolmányok és a kommentek listáját egyelőre nem tudjuk megjeleníteni, mert még nem hoztuk létre a szükséges modelleket.

```

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

```

A csatolmány feltöltésénél figyeljünk arra, hogy a formunk `:multipart` opciója `true` értékű legyen! A fájlbeviteli mezőt a `file_field_tag` helperrel adjuk hozzá. A formot a `upload` kontroller metódus fogja kezelni.

```

<div>
  <h3>Attachments</h3>
  <fieldset>
    <legend>Upload a new attachment</legend>
    <%= form_tag '/issues/upload', :multipart => true do %>
      <%= label_tag 'File' %>
      <%= file_field_tag 'file' %>
      <%= submit_tag "Upload" %>
    <% end %>
  </fieldset>
</div>

```

Az új komment hozzáadását megvalósító formba egy szövegmezőt helyezünk el, ami 60 oszlop széles és 10 sor magas. A kommentek listáját később adjuk hozzá.

```

<div>
  <h3>Comments</h3>
  <fieldset>
    <legend>Leave a comment</legend>

```

```

<%= form_tag '/comments/new', :method => :post do %>
  <%= label_tag 'Comment' %><br/>
  <%= text_area_tag 'comment', '', {:cols=>60, :rows=>10}
  %>
  <%= submit_tag "Send" %>
<% end %>
</fieldset>
</div>

```

A következő lépésben összekapcsoljuk a kontrollereket és a nézeteket. A kontrollerek példányváltozóit felhasználhatjuk a nézetekben, így dinamikus, a kéréstől függő adatot is elő tudunk állítani.

Az első módosítást a `users` controller `edit` akciójában tegyük meg, definiáljunk egy `@user` azonosítójú példányváltozót, amelyre a `edit` nézet form helperében a `:user` attribútummal hivatkozunk.

```

def edit
  @user = User.new(:email=>'kovacsg@tmit.bme.hu',
                  :username=>'kovacsg')
end

```

Ezután a `edit` nézetben automatikusan beíródik a form input mezőibe az aktuális felhasználónév, illetve az email cím. A jelszó és a jelszó megerősítése mezők értelemszerűen üresek maradnak.

Az automatikusan generált teendők controllerünk az adatbázisból dolgozna, de jelenleg ott nincs adatunk, ezért adjuk át helyettük lokálisan definiált objektumokat a nézeteknek! Az `index` akcióban kommentezzük ki az összes rekordot listázó hívást, és használjuk helyette egy új példányt, állítsuk be az `id` attribútumát, majd helyezzük el a nézetnek átadandó enumerációban.

```

def index
  #@issues = Issue.all
  issue = Issue.new(:deadline=>Time.now, :status=>1,
                  :priority=>5, :label=>"Todo",
                  :description=>"Urgent_todo")
  issue.id = 1
  @issues = [ issue ]
  ...
end

```

Járjunk el hasonlóan a `show` akció esetén is. Az `@issue` példányváltozó alapértelmezés szerint egy az `id` paraméter alapján az adatbázisból kikeresett rekordnak megfelelő objektum. Cseréljük ezt le egy lokális objektumra!

```
def show
  #@issue = Issue.find(params[:id])
  @issue = Issue.new(:deadline=>Time.now, :status=>1,
    :priority=>5, :label=>"Todo",
    :description=>"Urgent_todo")
  ...
end
```

Így már van megjeleníthető adatunk a nézeteken!