# HTML és Rails
# Gyakorlat

Kovács Gábor

2013. október 16.

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"></div>

<div id="menu"></div>

<div id="main">
<%= yield %>
</div>

<div id="footer">
RoR, 2013
</div>
```

Második lépésként készítsük el az oldal stíluslapját, amivel ezek a helyükre kerülnek, és helyezzünk el benne minimális mennyiségű formázási információt. A fejrész legyen világosszürke és 100 képpont magas. A menü a szélesség 24%-át töltse ki, legyen szürke hátterű, 400 képpont magas, balra igazított. Az oldal központi része legyen szintén 400 képpont magas, az oldal szélességének 75%-t töltse ki, világosszürke háttérrel rendelkezzen, és legyen szintén balra igazított. A lábrészben a szöveget igazítsuk középre, és legyen a lábrész magassága 100 képpont.

```
body {
  width: 1024px;
}

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

#menu {
  background-color: #dddddd;
  height: 400px;
  width: 24%;
  float: left;
}

#main {
  background-color: #eeeeee;
  height: 400px;
  width: 76%;
  float: left;
}

#footer {
  background-color: #cccccc;
  clear: both;
  height: 100px;
  text-align: center;
}
```

Kétféle felhasználóra készülünk fel egyelőre, egy látogatóra, és egy belépett felhasználóra, aki korábban keresztülment 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, vagyis a neptun kódra 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 paramé-

tere 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=>18 %><br/>
  <%= label_tag 'password', "Password" %><br/>
  <%= password_field_tag :password, '', :size=>18 %><br/>
  <%= submit_tag "Login"%>
  <% end %>
</fieldset>
<%= link_to "Register", '/users/new' %><br/>
<%= 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!<br/>
      <% else %>
      <%= render 'layouts/loginform' %>
      <% 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 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.

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 tizennyolc karakter széles felhasználónévre vonatkozó szövegbeviteli mező a hozzá kapcsolódó címkével, egy harminc karakter széles email cím szövegbeviteli mező a hozzá kapcsolódó címkével, két darab 18 karakter széles jelszóbeviteli mező a hozzájuk kapcsolódó címkével, a két jelszómező eltérő azonosítóval rendelkezzék, az egyik prefixe `_confirmation`-re végződjék, egy szövegbeviteli mező a bankszámlaszámnak, és végül egy nyomógomb. Az oldal alján helyezzünk el egy visszalépés gombot azon felhasználóknak, akik véletletül 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 => {:action => 'create' } do |
        form| %>
    <div>
      <%= form.label :username %>:
      <%= form.text_field :username, :size=>18 %>
```

```
    </div>
    <div>
      <%= form.label :email %>:
      <%= form.text_field :email, :size=>30 %>
    </div>
    <div>
      <%= form.label :password %>
      <%= form.password_field :password, :size=>18 %>
    </div>
    <div>
      <%= form.label :password_confirmation %>
      <%= form.password_field :password_confirmation, :size
          =>18 %>
    </div>
    <div>
      <%= form.label :bank_account %>
      <%= form.password_field :bank_account, :size=>24 %>
    </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.

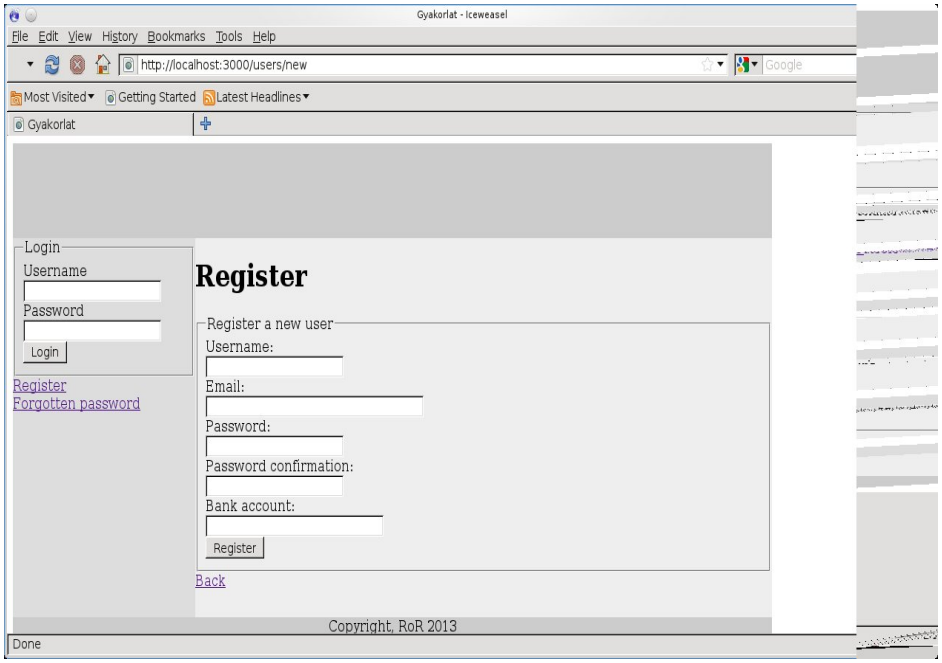
Láthatjuk, hogy a `:password_confirmation` szimbólumból a Rails automatikusan a *Password confirmation* szöveget állította elő. A stringek és a szimbólumok így ezen elv mentén felcserélhetők a form helperek argumentumlistájában.

```
<div>
  <fieldset>
    <legend>Register a new user</legend>
    <form accept-charset="UTF-8" action="/users/create"
        class="new_user" id="new_user" method="post"><div
        style="margin:0;padding:0;display:inline"><input
        name="utf8" type="hidden" value="&#x2713;" /><input
        name="authenticity_token" type="hidden" value="
```

```
        IOYYdCvoEvugQc6Ac1/6MR34d204fzzK9dnoshT6ZEM=" /></
        div>
         <div>
             <label for="user_username">Username</label>:<br
                 />
             <input id="user_username" name="user[username]"
                  size="18" 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="18" type="password" />
         </div>
         <div>
             <label for="user_password_confirmation">
                 Password confirmation</label>:<br/>
             <input id="user_password_confirmation" name="
                 user[password_confirmation]" size="18" type=
                 "password" />
         </div>
         <div>
             <label for="user_bank_account">Bank account</
                 label>:<br/>
             <input id="user_bank_account" name="user[
                 bank_account]" size="24" type="text" />
         </div>
         <input name="commit" type="submit" value="Register"
              />
</form>
</fieldset>
</div>

<a href="http://localhost:3000/users/edit">Back</a>
</div>
```

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




1. ábra. A létrehozott regisztrációs oldal

Az új analógiájára hozzuk létre az elfejetett jelszó oldalt is. Itt egyszerűbb a formunk, csak az email címet tartalmazza, és a címkék különböznek az előző példához képest.

```
<h1>Forgotten password</h1>
<fieldset>
  <legend>Please, give your email address</legend>
  <%= form_for @user, :url=>{:action=>"send_forgotten"} do
      |form| %>
    <div>
      <%= form.label :email %>: <br/>
      <%= form.text_field :email %>
      <%= form.submit "Send" %>
    </div>
  <% 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, új fogadás kezdeményezése, a saját korábbi fogadások

megtekintése, az aktuális fogadások megtekintése és az aktuális fogadható események listája. Az első a `users` kontroller `edit` akciójára és nézetére mutat, a kilépés a `sessions` kontroller `destroy` akciójára, a többi pedig egyelőre nem definiált kontrollerek nem definiált akcióira mutatnak. Az utóbbiakhoz új modellt és kontrollert is létre kell hoznunk. Az események modelljét nevezzük `event`-nek, a fogadások modelljét pedig `bid`-nek, ekkor a Rails konvenciói szerint az összes, aktuálisan fogadható eseményt a `/events` útvonalon láthatjuk majd kis módosítás után. A bejelentkezett felhasználó a `/bids/new` nézeten kezdeményezhet majd új fogadást, a `/bids` nézeten megtekintheti saját aktuális, illetve korábbi fogadásait, végül a saját profil oldalának megtekintésével láthatja az aktuális egyenlegét.

```
Hello, user!<br/>
<%= link_to "Edit profile", '/users/edit' %><br/>
<%= link_to "View events", '/events' %><br />
<%= link_to "New bid", '/bids/new' %><br />
<%= link_to "My bids", '/bids' %><br />
<%= link_to "My bids history", '/bids' %><br />
<%= link_to "My balance", '/users/show' %><br />
<%= link_to "Logout", '/session/destroy'%>
```

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

```
<h1>Edit profile</h1>
<p><%= flash[:notice] %></p>

<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=>18 %>
      </div>
      <div>
          <%= form.label :email %>:<br/>
          <%= form.text_field :email, :size=>30 %>
      </div>
      <div>
          <%= form.label :password %>:<br/>
          <%= form.password_field :password, :size=>18 %>
```

```
        </div>
        <div>
            <%= form.label :password_confirmation %>:<br/>
            <%= form.password_field :password_confirmation,
                :size=>18 %>
        </div>
        <div>
            <%= form.label :bank_account %>:<br/>
            <%= form.text_field :bank_account, :size=>24 %>
        </div>
        <%= submit_tag "Update" %>
    <% end %>
  </fieldset>
</div>
<%= link_to "Back", :back %>
```

A felhasználó a saját profilja megtekintésekor láthatja az aktuális egyenlegét.

```
<h1>Balance of <%= @user.username %></h1>
<%= @user.account %>
```

Mivel a felhasználói nézetek mind szükségessé teszik a felhasználó kontrollerében a `@user` példányváltozó inicializáltságát részben a `form_for` metódus, részben követlen használat miatt, végezzük el a szükséges módosításokat. Továbbá adjuk hozzá a kontrollerhez a nézetek formjainak eseménykezelő metódusait, amelyek elnevezésükben kövessék a Rails konvencióit, ha lehetséges. Tehát az új felhasználó létrehozása eseményt a `create`, a felhasználó adatainak módosítása eseményt pedig a `update` akció kezelje le. Ezután ellenőrizhetjük, hogy `edit` metódusban beállított értékek beíródnak-e a nézet formjának megfelelő mezőjébe.

```
class UsersController < ApplicationController
  def new
    @user = User.new
  end

  def create
  end

  def edit
    @user = User.new
    @user.username = 'valaki'
    @user.email = 'valaki@bme.hu'
```

```
    @user.bank_account = 11111111
    @user.password = 'titok'
  end

  def update
  end

  def show
    @user = User.new
    @user.username = 'valaki'
    @user.email = 'valaki@bme.hu'
    @user.bank_account = 11111111
    @user.password = 'titok'
    @user.account = 1
  end

  def forgotten
    @user = User.new
  end

  def send_forgotten
  end
end
```

Az események modelljét, kontrollerét és nézetét egyetlen paranccsal hozzuk létre. A `scaffold` automatikusan gerenálja 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 megszokottaktók (parancssoron a `rake routes` paranccsal nézhetjük meg, hogy mi változott). Négy nézezet 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 event eventtime:datetime
   description:text oddstrue:float oddsfalse:float
```

Hajtsuk végre az adatbázis migrációt!

```
rake db:migrate
```

Legyen az események tábla a következő. Legyen minden eseménynek egy időpont típusú bekövetkezési ideje (`eventtime`), egy szöveg típusú leírása (`description`) és egy-egy lebegőpontos típusú értéke, ami az esemény bekövetkezte (`oddstrue`), illetve be nem következte (`oddsfalse`) esetén adható nyeremény szorzóját adja meg. Ezeken kívül a szokásos időpecsétek automatikusan generálódnak.

```
class CreateEvents < ActiveRecord::Migration
  def change
    create_table :events do |t|
      t.datetime :eventtime
      t.text :description
      t.float :oddstrue
      t.float :oddsfalse

      t.timestamps
    end
  end
end
```

Az `/events` útvonalon elérhető események listája nézet egy táblázatot tartalmaz, ahol minden egyes modell attribútumhoz egy oszlop tartozik. Az események 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. Ezekre egyelőre még ne klikkeljünk rá, mert adatbázisműveletet vonnának maguk után. Az oldal alján egy link található az új feladat felvétele oldalra. Módosítsuk a Show és Edit `link_to` függvényhívását úgy, hogy az megfeleljen az útvonalainknak: `"/events/show/#{event.id}"`, illetve `"/events/edit/#{event.id}"`.

Feladat 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 kettő (`time`), három (`date`) vagy öt (`datetime`) legördülő menükből összerakott beviteli mező tartozik függően attól, hogy az időpontot is állítani akarjuk-e, vagy csak a dátumot.

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 automatikusan generált események kontrollerünk 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, hogy a weboldalon megjelenő linkek helyesek legyenek, majd helyezzük el a nézetnek átadandó enumerációban.

```
def index
    #@events = Event.all
    event1 = Event.new
    event1.eventtime = Time.now + 1.week
    event1.description = "Holyday"
    event1.oddstrue = 1.1
    event1.oddsfalse = 9.0

    event2 = Event.new
    event2.eventtime = Time.now + 2.weeks
    event2.description = "Test"
    event2.oddstrue = 8.3
    event2.oddsfalse = 1.1

    @events = [ event1, event2 ]
    ...
end
```

Hasonlóan járhatunk el a `show`, `edit` akciók esetén is. Így máris van megjeleníthető adatunk a nézeteken! Az `new` akció esetén szerencsénk van, ott egy új példányt találunk inicializálatlan attribútumokkal.

```
  def show
    #@event = Event.find(params[:id])
    @event = Event.new
    @event.eventtime = Time.now + 1.week
    @event.description = "Holyday"
    @event.oddstrue = 1.1
    @event.oddsfalse = 9.0
    ...
  end
```

Az eseményekhez hasonló módon hozzuk létre a fogadások modelljét, majd hajtsuk végre a migrációt. Egyelőre csak egy attribútumot rendeljünk hozzá, a fogadás időpontját. A fogadások ezen kívül rendelkezni fognak egy felhasználó referenciájával és legalább egy esemény referenciájával. Ennek megvalósítása azonban a következő gyakorlatra marad.

```
rails generate scaffold bid bidtime:datetime
rake db:migrate
```