HTML és Rails
Gyakorlat

Kovács Gábor

2013. március 18.

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">
Tidy up!
</div>

<div id="menu">
<%= render '/layouts/loginform' %>
</div>

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

<div id="footer">
Copyright RoR, 2014
</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 és 800 pixel magas. A fejrész legyen világosszürke és 100 képpont magas, és nagy betűs szöveget helyezhessünk el benne. A menüsáv a fejléc alatt legyen szürke hátterrel 20 pixel magassággal. Az oldal központi része legyen 600 képpont magas, világosszürke

háttérrel rendelkezzen. A lábrészben a szöveget igazítsuk középre, és legyen a lábrész magassága 100 képpont.

```
body {
    width: 1024px;
    height: 800px;
}

div#header {
    background-color: #cccccc;
    font-size: 54px;
    height: 100px;
}

#menu {
    background-color: #dddddd;
    height: 20px;
}

#main {
    height: 600px;
    background-color: #eeeeee;
}

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

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 `_loginform.html.erb`! 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 18 hosszúra korlátozzuk. A be nem jelentkezett felhasználónak tegyük lehetővé az elfelejtett jelszó visszaszerzését, ezt egy link hozzáadásával tesszük meg.

```
<%= form_tag '/sessions/create', :method => :post do %>
<%= label_tag :username, "Username" %>
<%= text_field_tag :username, '', :size=>18 %>
<%= label_tag :password, "Password" %>
<%= password_field_tag :password, '', :size=>18 %>
<%= submit_tag 'Login' %>
<%= link_to "Forgotten password", '/forgotten/forgotten' %>
<% end %>
```

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 menüt megvalósító `loginform`-ban a következő módosítást végezzük el. Így a helper módosításával be, illetve ki tudunk lépni az oldalról. „Jelenkteztessük” 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 %>
```

Hozzuk létre a felhasználók által elvégzendő 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 adatstruktúráját úgy alakítsuk ki, hogy feljegyezhessük magunknak azok végrehajtási határidejét, a készültségi állapotát, amit egész számmal reprezentálunk, a feladat szöveges leírását, és a feladat rövid nevét. A parancs után hajtsuk végre az adatbázis migrációját, és nézzük meg az eredményt!

```
rails generate scaffold task deadline:datetime status:
  integer description:text title: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. Az, hogy e linkek közül melyek lesznek elérhetők a vendég felhasználók, a bejelentkezett felhasználók és az adminisztrátor számára, egy későbbi döntés kérdése.

A `new` és a `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.

A `show` nézet a feladat adatstruktúra mezőinek értékeit írja ki, nem abban a formátumban, ahogy az nekünk majd kelleni fog.

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
  t1 = Task.new
  t1.id = 1
  t1.deadline = Time.now + 1.week
  t1.title = 'Porszivozas'
  t1.state = 0
  t1.description = 'Lattad a szoba sarkat?'
  t2 = Task.new
  t2.id = 2
  t2.deadline = Time.now + 1.day
  t2.title = 'Mosogatas'
  t2.state = 0
  t2.description = 'Ehes vagyok, es nincs tanyer'
  @tasks = [ t1, t2 ]
end
```

Hasonlóan az `edit` és a `show` nézetek `@task` kontroller példányvátozóit is inicializáljuk, 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. Mivel a `@task` automatikusan inicializálódik a kontroller `set_task` privát metódusa által az adatbázis alapján, kommentezzük ki azt a sort.

```
def edit
  t1 = Task.new
  t1.id = 1
  t1.deadline = Time.now + 1.week
  t1.title = 'Porszivozas'
  t1.state = 0
  t1.description = 'Lattad a szoba sarkat?'
  @task = t1
end

def set_task
  #@task = Task.find(params[:id])
end
```

A felhasználók modelljét az előző gyakorlaton létrehoztuk, azonban akkor nem vettük figyelembe, hogy a felhasználók regisztációja milyen módon történjék. Mivel egy új felhasználó regisztrációja csak az adminisztrátor által történhet meg, ezért stratégiát váltunk, töröljük a korábban létrehozott modellünket, és az adminisztráció szempontjából könnyebben karbantartható módon hozzuk újra létre. Az attribútumok megegyeznek az előző gyakorlaton felvettekkel kiegészítve azt az előző alkalommal elfejetett `password` mezővel. A struktúrából azonban változatlanul hiányzik egy kétértékű mező, az adminisztrátori jogosultságokat jelző tulajdonság.

```
rails generate scaffold user username:string password:
   string first_name:string family_name:string email:string
    last_login:datetime
```

A parancs futtatásával létrejött az `users` kontroller és a hozzá kapcsolódó nézetek. A felhasználók listáját táblázatosan megjelenítő `index` nézet kifejezetten alkalmas lesz a felhasználók adminisztrációjának ellátására, a `show` profiloldalt azonban módosítanunk kell.

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 külön kontrollerbe egyetlen, `forgotten` nevű nézetbe.

```
rails generate controller forgotten forgotten
```

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

```
<h1>Forgotten password</h1>
<%= form_tag '/forgotten/send_reminder' do %>
<%= label_tag :email, "Please give your email address" %>
<%= text_field_tag :email, @user.email, :size=>40 %>
<%= submit_tag "Send reminder" %>
<% end %>
```

Az elfelejtett jelszó kiküldését a form eseményét kezelő kontroller akció, a `send_reminder` teszi majd meg, amit fel kell vennünk a kontroller osztályába egyelőre üres törzzsel.

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, az aktuális teendők megtekintése. Ezen funkciókhoz hozzunk létre egy-egy linket a menüben. A kilépés művelet mutasson a még nem létező `sessions` kontroller `destroy` akciójára.

```
<%= link_to "View profile", '/users' %>
<%= link_to "View todos", '/tasks' %>
<%= link_to "Logout", '/sessions/destroy' %>
```