HTML és Rails
Gyakorlat

Kovács Gábor

2023. április 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="page">
  <div id="header"></div>
  <div id="main">
    <div id="menu">
    </div>
    <div id="content">
      <%= yield %>
    </div>
  </div>
  <div id="footer">
    RoR, 2023
  </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 800 pixel széles. A fejrész legyen világosszürke és 100 képpont magas, és helyezzünk el benn egy banner képet. Az oldal központi része legyen 600 pixel magas. A menüsávot a központi részben, a fejléc alatt helyezzük el, és az vízszintesen a szélesség 24%-át foglalja el, magasságát a központi rész magassága definiálja. Az oldal tartalmi része világosszürke

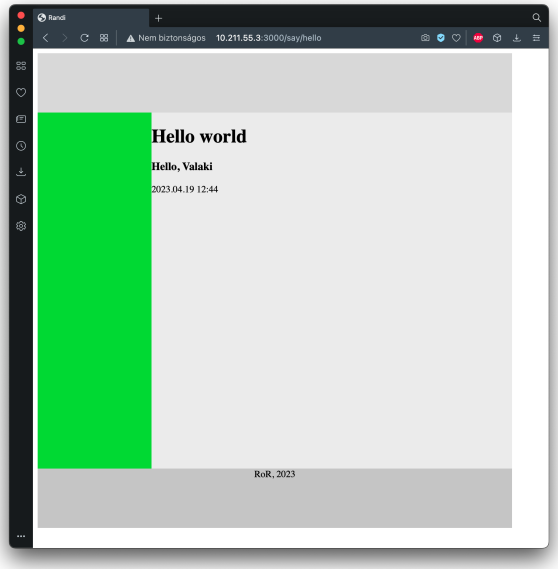
háttérrel rendelkezzen, és a menütől jobbra helyezkedjen el a vízszintesen a szélesség 76%-át elfoglalva. A lábrészben a szöveget igazítsuk középre, és legyen az is 100 pixel magas, valamint a fejlécnél világosabb szürke színű.

```
div#page {
        width: 800px;
}
div#header {
        height: 100px;
        background-color: #ddd;
}
div#footer {
        height: 100px;
        background-color: #ccc;
        text-align: center;
        clear: both;
}
div#main {
          height: 600px;
}
div#menu {
        float: left;
        width: 24%;
        height: 100%;
        background-color: #0d0;
}
div#content {
        float: left;
        width: 76%;
        height: 100%;
        background-color: #eee;
}
```

Az így kialakított elrendezést például az 1. ábra mutatja.

Két felhasználótípusra készülünk fel egyelőre, egy látogatóra és egy bejelentkezett felhasználóra, az utóbbiak korábban keresztülmentek egy regisztrációs folyamaton. A látogató csak böngészhet, bejelentkezhet és jelszóemlékeztetőt kérhet. A bejelentkezett felhasználók számára több funkciót is elérhetővé teszünk.

```
kovacsg@debian:~/randi/app/views/layouts> touch _guest.html
    .erb
kovacsg@debian:~/randi/app/views/layouts> touch _user.html.
    erb
```



1. ábra. Az oldal elrendezésének kialakítása

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 Rails konvenció szerint a részlegesen renderelt állományok neve aláhúzásjellel kezdődik. Legyen a fájlunk neve ezért `_guest.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 14 karakter hosszúra korlátozzuk. A be nem jelentkezett felhasználónak tegyük lehetővé az elfelejtett jelszó visszaszerzését, ezt egy link hozzáadásával tesszük meg.

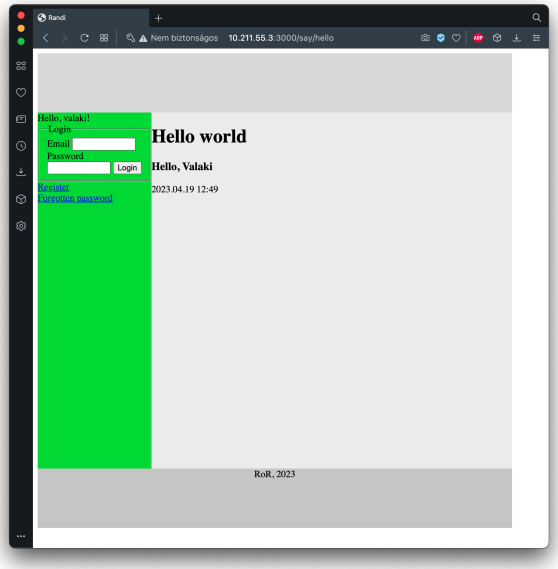
```
Hello, valaki!

<fieldset>
  <legend>Login</legend>
  <%= form_tag '/sessions/create', method: :post do %>
    <%= label_tag 'email', 'Email' %>
```

```
    <%= text_field_tag 'email', '', size: 14 %>
    <%= label_tag 'password', 'Password' %>
    <%= password_field_tag 'password', '', size: 14 %>
    <%= 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.

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

A vendégfelhasználó menüjének megvalósítását a 2. ábra mutatja.



2. ábra. A vendégfelhasználó menüje

Modellezzük azt az esetet is, amikor egy felhasználó már bejelentkezett. Ezt egy, a `helpers/application_helper` állományban elhelyezendő saját helper metódussal tesszük meg `logged_in?`. Itt egyelőre manuálisan állítjuk, hogy be van-e jelentkezve a felhasználó. A metódus értelemszerűen boolean visszatérési értékű.

```
module ApplicationHelper
  def logged_in?
    true
  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 a `logged_in?` függvény visszatérési értékének módosításával, hogy ellenőrizhessük, hogy a megfelelő menü jelenik-e meg a vendég és a felhasználó számára.

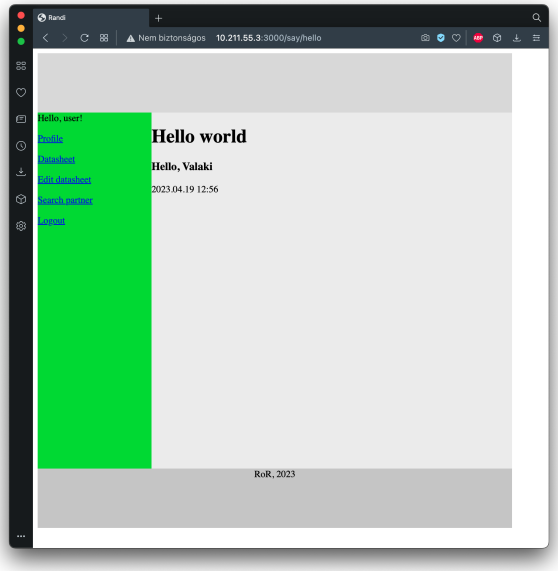
```
<div id="menu">
 <% if logged_in? %>
   <%= render 'layouts/user' %>
 <% else %>
   <%= render 'layouts/guest' %>
 <% end %>
</div>
```

A bejelentkezett hallgató felhasználó menüjét a vendéghez hasonlóan beágyazott nézettel hozzuk létre (`_user.html.erb`). Egyelőre öt akciót definiálunk: a profiloldal megtekintését, a felhasználó adatlapjának megtekintését és szerkesztését, a felhasználói adatlaplista megtekintését, valamint a kijelentkezést.

```
Hello, user!

<p><%= link_to 'Profile', '/users/edit' %></p>
<p><%= link_to 'Datasheet', '/people/1' %></p>
<p><%= link_to "Edit datasheet", '/people/1/edit' %></p>
<p><%= link_to "Search partner", '/people' %></p>
<p><%= link_to "Logout", '/sessions/destroy' %></p><p>Hello
    , user!</p>
```

A bejelentkezett hallgató felhasználó menüjének megvalósítását a 3. ábra mutatja.

Nézzük meg a be nem lépett felhasználó létrehozásának 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:



3. ábra. A bejelentkezett felhasználó menüje

```
kovacsg@debian:~/randi/app/views/layouts$ rails g
   controller users new edit forgetten
      create  app/controllers/users_controller.rb
       route  get 'users/new'
              get 'users/edit'
              get 'users/forgetten'
      invoke  erb
      create    app/views/users
      create    app/views/users/new.html.erb
      create    app/views/users/edit.html.erb
      create    app/views/users/forgetten.html.erb
      invoke  test_unit
      create    test/controllers/users_controller_test.rb
      invoke  helper
      create    app/helpers/users_helper.rb
      invoke    test_unit
```

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`, és az elfelejtett jelszó esetén az email címet elkérő `forgotten` nézet. Az 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 kontrollerébe.

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 egy modell objektum vagy annak neve szimbólum formájában, második paramétere a formhoz kötött akció, amely legyen a `users` kontroller (ezt nem kell leírnunk, mert az új felhasználó létrehozása akció kontrollere ugyanaz) `create` akciója, a harmadik paramétere a HTTP metódus, ami POST. A metódus blokkjának van egy paramétere a `form`, amin kereszül definiáljuk fogjuk az űrlap elemeit. Legyen a négy elem rendre a következő: egy 20 karakter széles, a felhasználónévre vonatkozó szövegbeviteli mező a hozzá kapcsolódó címkével, egy. A két jelszómező eltérő azonosítóval rendelkezzék, az egyik prefixe `_confirmation`-re végződjék.

Ha a felhasználó meggondolná magát, és megsem kívánná regisztrálni magát, egy `Back` feliratú linkkel biztosítjuk számára a lehetőséget az előző oldalra való visszatérésre.

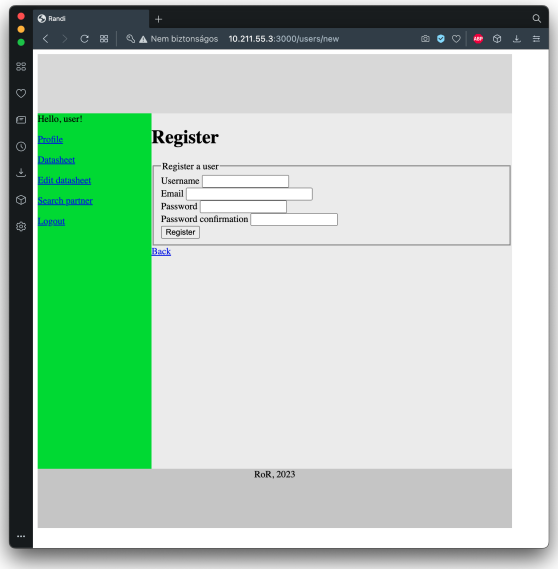
```
<h1>Register</h1>

<fieldset>
  <legend>Register a user</legend>
  <%= form_for @user, url: '/users/create', method: :post
      do |form| %>
    <div>
      <%= form.label :username %>
      <%= form.text_field :username, size: 20 %>
    </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_confirmation, size:
          20 %>
    </div>
```

```
    <%= form.submit "Register" %>
  <% end %>
</fieldset>

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

Ahhoz, hogy az űrlap megjelenjen, a kontrollerben inicializánunk kell a `@user` példányváltozót.

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

A felhasználói regisztráció nézetét a 4. ábra mutatja.



4. ábra. A regisztráció nézete

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 felhasználói profil szerkesztésének nézetében (`edit.html.erb`) található form szinte teljesen megegyezik az új felhasználót létrehozó formmal. A nézetben a feliratokat kell még átírnunk regisztrációról profil szerkesztésére.

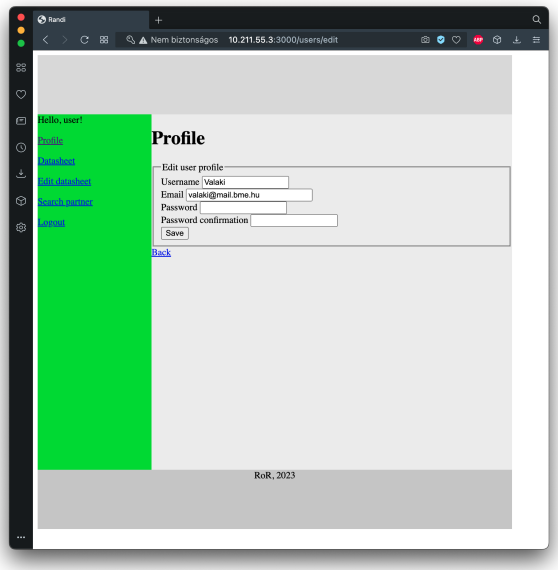
```
<h1>Profile</h1>

<fieldset>
  <legend>Edit user profile</legend>
  <%= form_for @user, url: '/users/update', method: :put do
      |form| %>
    <div>
      <%= form.label :username %>
      <%= form.text_field :username, size: 20 %>
    </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_confirmation, size:
          20 %>
    </div>
    <%= form.submit "Save" %>
  <% end %>
</fieldset>

<%= 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. A `show` akcióban is inicializáljuk a `@user` példányváltozót, a `index` akcióban pedig a `@users` példányváltozót, amely egy User típusú objektumokat tartalmazó tömb.

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

  def edit
    @user = User.new username: 'Valaki', email: '
        valaki@mail.bme.hu'
  end
end
```

A felhasználói profiloldal szerkesztésének nézetét az 5. ábra mutatja. Láthatjuk, hogy a Rails automatikusan inicializálta a form mezőit, ahol a hozzájuk tartozó érték elérhető volt – a jelszó mezők kivételével.



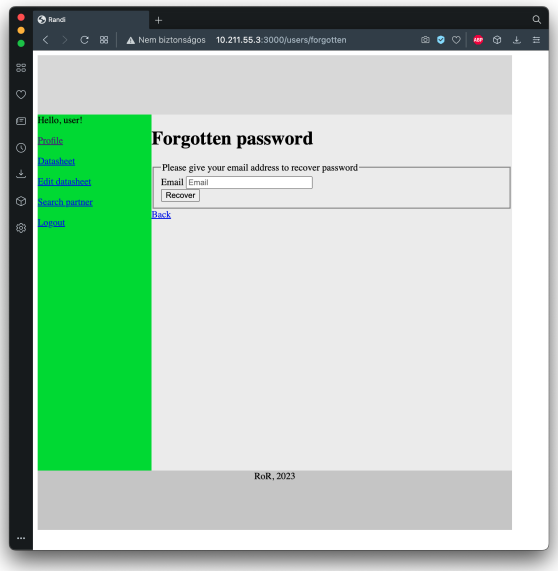
5. ábra. A profil szerkesztése képernyő

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

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

<fieldset>
  <legend>Please give your email address to recover
      password</legend>
  <%= form_tag '/users/send_forgotten', method: :post do %>
    <div>
      <%= label_tag :email, 'Email' %>
      <%= text_field_tag :email, '', placeholder: 'Email',
          size: 30 %>
    </div>
    <%= submit_tag "Recover" %>
  <% end %>
</fieldset>

<%= 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.

Az elfelejtett jelszó nézetét a 6. ábra mutatja.



6. ábra. Az elfelejtett jelszó nézete

Az előző gyakorlaton már létrehoztuk az adatlapok modelljét, és automa-

tikusan generált nézeteit. Most az `index`, illetve a `show` képernyőket részben újraírjuk.

A `people/index.html.erb` nézetben eredetileg az adatlapokhoz tartozó töredékeket ágyaztunk be, azonok egy `for` ciklussal végigszaladva a kontroller `index` akciójában definiált `@people` példányváltozó elemein. Most az alapértelmezett módon generált linkeket, vagyis az új adatlap és az adatlap mutatása linkeket töröljük. Az utóbbit pedig átmozgatjuk az egyes felhasználók adatait mutató adatlap töredékbe.

```
<p style="color: green"><%= notice %></p>

<h1>People</h1>

<div id="people">
  <% @people.each do |person| %>
    <%= render person %>
    <p>
      <%#= link_to "Show this person", person %>
    </p>
  <% end %>
</div>

<%#= link_to "New person", new_person_path %>
```

A `_person.html.erb` töredékben elhelyezünk egy profilképet, és mellette az automatikusan generált adatlistát, mindkettőt egy-egy divbe foglalva. A profilkép és a felhasználónév legyen klikkelhető.

```
<div id="<%= dom_id person %>">
  <a href="<%= people_path person %>">
  <div class="profile-image">
    <img src="" width="160" height='160'>
  </div>
  </a>
  <div class="profile-data">
  <p>
    <strong>Name:</strong>
    <a href="<%= people_path person%>"><%= person.name %></
      a>
  </p>

  <p>
    <strong>Birthdate:</strong>
    <%= person.birthdate %>
```

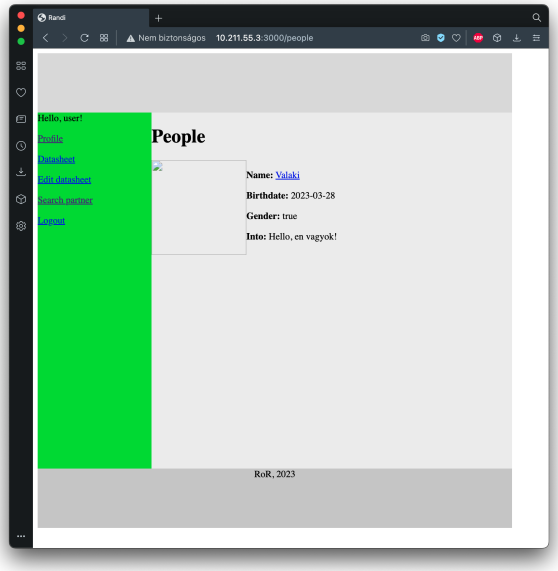
```
  </p>

  <p>
    <strong>Gender:</strong>
    <%= person.gender %>
  </p>

  <p>
    <strong>Into:</strong>
    <%= person.into %>
  </p>
  </div>
</div>
```

Az egymás mellett való elhelyezéshez a két új divhez formázást rendelünk.

```
div.profile-image {
        width: 160px;
        height: 160px;
        float: left;
}
div.profile-data {
        width: 300px;
        height: 160px;
        float: left;
}
div.profile-actions {
        clear: both;
}
```

Az adatlaplista nézetét a 7. ábra mutatja.

Az adatlapképernyő a `show.html.erb` fájlban található, ahol amellett, hogy ugyanez az adatlap jelenik meg, amit az imént módosítottunk, műveleteket is végrehajthatunk az adatlaphoz tartozó felhasználóval kapcsolatban. Az alapértelmezett műveleket, vagyis a felhasználói adatlap szerkesztésének, az adatlap törlésének lehetőségét ki kell zárnunk, kikommentezzük ezeket a sorokat. Helyettük felvesszük az ismerősnek jelölés és az ismerős kapcsolat törlésének lehetőségét, valamint a komment üzenetekkel való kommunikációt. Az előbbieket két link létrehozásával tesszük meg, amelyek a későbbiekben nem lesznek majd egyszerre láthatóak, erről majd később kell gondoskodnunk, amikor tudjuk, hogy ki kinek az ismerőse. A kommentelő és az adatlap tulajdonosa közötti üzeneteket jelenítsük meg külön bekezdésekben benne a kommentelő felhasználónevével, az üzenet lokalizált időpontjával és az üzenet törzsével. Az üzenetlista alá tegyünk ki egy új üzenet létrehozó formot,



7. ábra. Az adatlaplisták nézete

amelyet később Javascripttel kezelünk.

```
<p style="color: green"><%= notice %></p>

<%= render @person %>

<div class="profile-actions">
  <%# link_to "Edit this person", edit_person_path(@person
      ) %> |
  <%# link_to "Back to people", people_path %>

  <%# button_to "Destroy this person", @person, method: :
      delete %>

  <% @comments.each do |comment| %>
    <div class="commentbox">
      <p><%= comment.user.username %> left a message at <%=
          l comment.created_at %></p>
      <p><%= comment.comment %> </p>
    </div>
  <% end %>

  <%= form_tag 'comments/add', method: :post do %>
```

```
    <%= text_area_tag :comment, '', rows: 5, columns: 80 %>
    <%= submit_tag :send %>
  <% end %>

  <%= link_to "Mark as friend", 'people/add_friend' %>
  <%= link_to "Unfriendly", 'people/remove_friend' %>
</div>
```

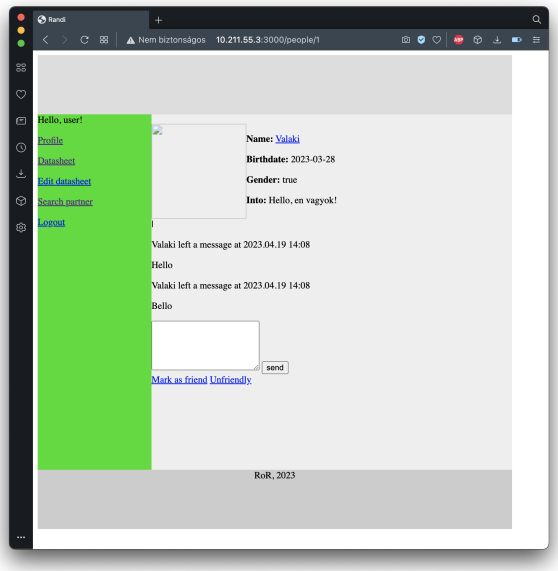
A kommentelhetőséghez szükségünk van a `@comments` példányváltozóra. A komment adatstruktúra attribútumai a megjelenítendő adatokból következnek, vagyis kell egy hivatkozás egy felhasználóra, és egy mező az üzenet törzse számára, továbbá az időpecsétek automatikusan létrejönnek. A modell generálásakor ha más típusra akarunk hivatkozni, akkor a `references` típust kell megadnunk. Hajtsuk végre a migrációt, amint létrejött a modell.

```
kovacsg@debian:~/randi/app/controllers> rails g model
    comment user:references comment:text
       invoke  active_record
       create    db/migrate/20230418113023_create_comments.
           rb
       create    app/models/comment.rb
       invoke    test_unit
       create      test/models/comment_test.rb
       create      test/fixtures/comments.yml
kovacsg@debian:~/randi/app/controllers> rails db:migrate
== 20230418113023 CreateComments: migrating
    ===============================================
-- create_table(:comments)
    -> 0.0284s
== 20230418113023 CreateComments: migrated (0.0288s)
    ======================================
```

Most már van adatstruktúránk, amelyből egy tömböt kell létrehoznunk a kontroller `show` akciójában. Statikus adatokat veszünk fel, később azonban lecseréljük adatbázisból származókra.

```
class PeopleController < ApplicationController
  # GET /people/1 or /people/1.json
  def show
    c1 = Comment.new user_id: 1, comment: 'Hello',
        created_at: Time.now
    c2 = Comment.new user_id: 1, comment: 'Bello',
        created_at: Time.now
    @comments = [c1, c2]
  end
```

Az adatlap nézetét a 8. ábra mutatja.



8. ábra. Az adatlap nézete