HTML és Rails
Gyakorlat

Kovács Gábor

2023. október 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="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 az `app/assets/stylesheets` könyvtárban az `application.css` szerkesztésével. 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,

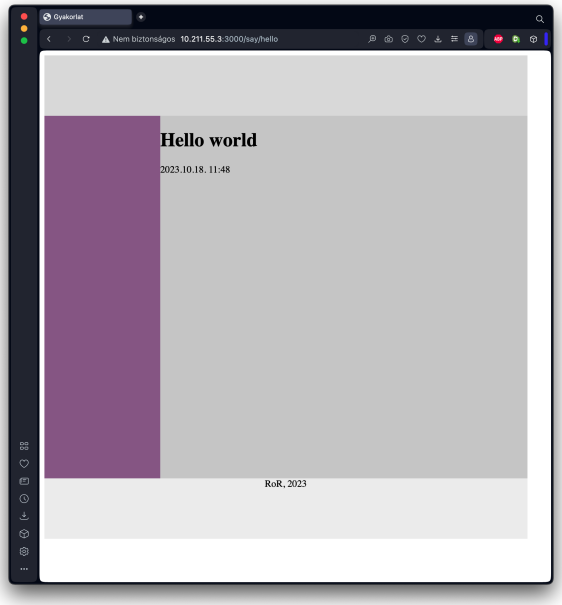
és a menütől jobbra helyezkedjen el a vízszintesen a szélesség 76%-át elfoglalva. A lábrészben a szöveget igazítsuk középre, és legyen az is 100 pixel magas, valamint a fejlécnél világosabb szürke színű.

```
 div#page {
     width:  800px;
 }
 div#header {
     height: 100px;
     background-color: #ddd;
 }
 div#footer {
     height: 100px;
     background-color: #eee;
     text-align: center;
 }
 div#main {
     height: 600px;
 }
div#menu {
  float: left;
  width: 24%;
  height: 100%;
  background-color: #906090;
}
div#content {
     float: left;
     width: 76%;
     height: 100%;
     background-color: #ccc;
}
```

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

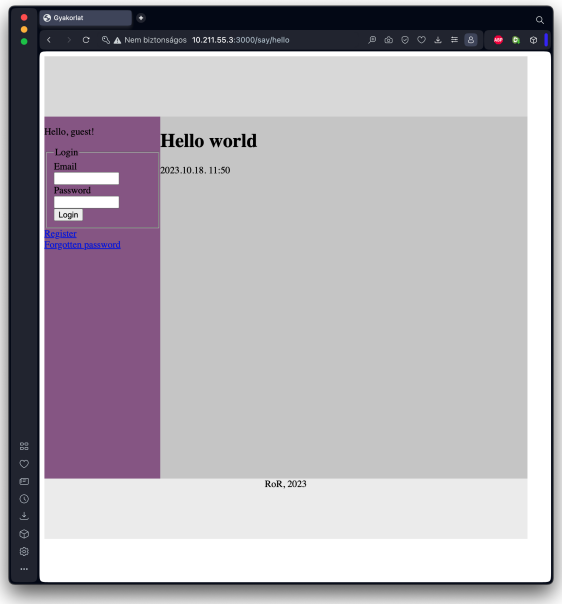
```
<p>Hello, guest!</p>
<fieldset>
    <legend>Login</legend>
    <%= form_tag '/sessions/create', method: :post do %>
        <%= label_tag 'email', "Email" %><br/>
```

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

```
<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 négy akciót definiálunk: a profiloldal megtekintését és szerkesztését, a feladatok listájának megtekintését, a megoldások listjának megtekintését, valamint a kijelentkezést.

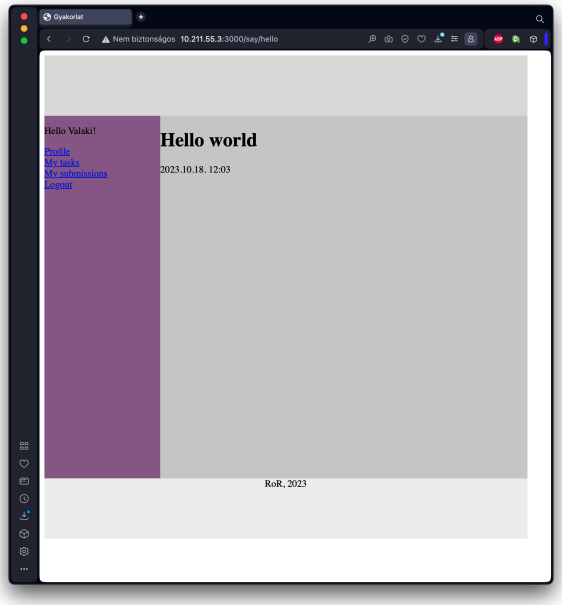
```
P>Hello Valaki!</p>

<%= link_to "Profile", '/users/edit' %><br/>
<%= link_to "My tasks", '/tasks' %><br/>
<%= link_to "My submissions", '/submissions/index' %><br/>
<%= link_to "Logout", '/sessions/destroy' %>
```

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 architektúra szerint szorosan kapcsolódik a nézethez. Hozzuk létre a felhasználó nézetét és fontosabb akcióit a következő paranccsal:

```
kovacsg@debian:~/gyakorlat/app/views/layouts> rails g
    controller users new edit forgotten
```



3. ábra. A bejelentkezett felhasználó menüje

```
      create  app/controllers/users_controller.rb
       route  get 'users/new'
              get 'users/edit'
              get 'users/forgotten'
      invoke  erb
      create    app/views/users
      create    app/views/users/new.html.erb
      create    app/views/users/edit.html.erb
      create    app/views/users/forgotten.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 az öt elem rendre a következő: egy 20 karakter széles, a felhasználó nevére vonatkozó szövegbeviteli mező a hozzá kapcsolódó címkével, egy 30 karakter széles, a felhasználó email címére vonatkozó szövegbeviteli mező a hozzá kapcsolódó címkével, egy 8 karakter széles, a felhasználó Neptun-kódjára vonatkozó szövegbeviteli mező a hozzá kapcsolódó címkével, valamint két, 20 karakter széles jelszóbeviteli mező a hozzájuk kapcsolódó címkékkel. A kétjelszómező 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>Registration</h1>
<fieldset>
    <legend>Register a new user</legend>
    <%= form_for @user, url: '/users/create', method: :post
        do |f| %>
        <div>
            <%= f.label :name %>
            <%= f.text_field :name, size: 20 %>
        </div>
        <div>
            <%= f.label :email %>
            <%= f.text_field :email, size: 30 %>
        </div>
        <div>
            <%= f.label :neptun %>
            <%= f.text_field :neptun, size: 8 %>
        </div>
        <div>
            <%= f.label :password %>
```

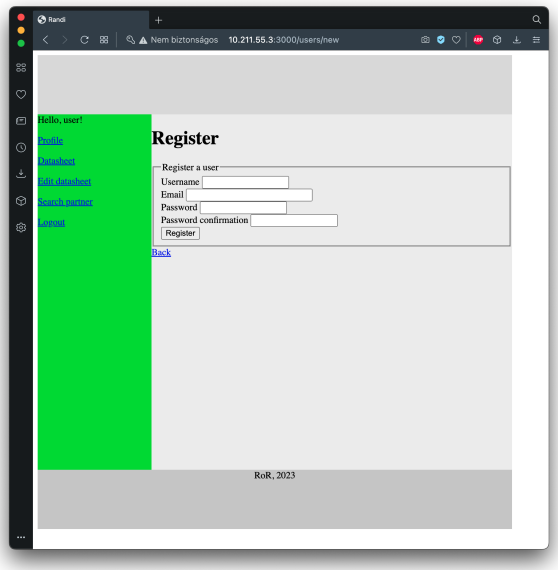
```
            <%= f.password_field :password, size: 20 %>
        </div>
        <div>
            <%= f.label :password_confirmation %>
            <%= f.password_field :password_confirmation,
                size: 20 %>
        </div>
        <%= f.submit "Register" %>
    <% end %>
</fieldset>

<%= 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, illetve módosítanunk kell az eseménykezelő URL-t.

```
<h1>Profile</h1>
<fieldset>
    <legend>Edit user profile</legend>
    <%= form_for @user, url: '/users/update', method: :put
        do |f| %>
        <div>
            <%= f.label :name %>
            <%= f.text_field :name, size: 20 %>
        </div>
        <div>
            <%= f.label :email %>
            <%= f.text_field :email, size: 30 %>
        </div>
        <div>
            <%= f.label :neptun %>
            <%= f.text_field :neptun, size: 8 %>
        </div>
        <div>
            <%= f.label :password %>
            <%= f.password_field :password, size: 20 %>
        </div>
        <div>
            <%= f.label :password_confirmation %>
            <%= f.password_field :password_confirmation,
                size: 20 %>
        </div>
        <%= f.submit "Update" %>
    <% 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.

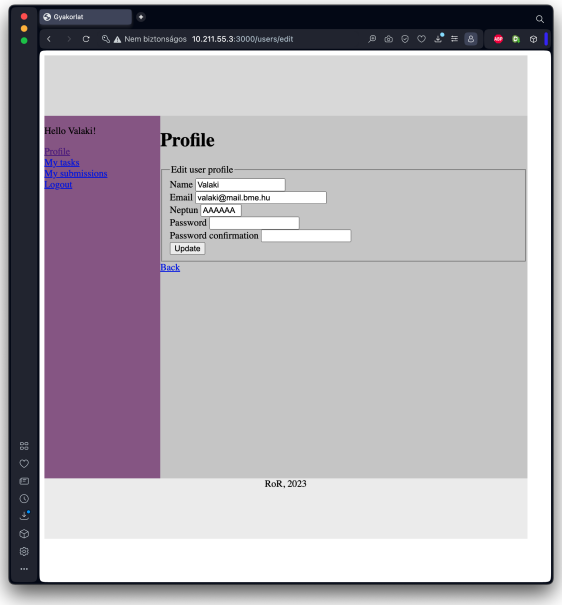
```
class UsersController < ApplicationController
  def new
    @user = User.new
  end

  def edit
    @user = User.new name: "Valaki", email: 'valaki@mail.
        bme.hu', neptun: 'AAAAAA'
  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.

Ezután alakítsuk ki az elfejetett jelszó oldalt is. Itt egyszerűbb a formunk a beléptetésnél, csak az email címet tartalmazza. Használjuk itt is a `form_for` helpert, így a `@user` példányváltozót itt is egy üres objektumra kell inicializálnunk a kontrollerben.

```
<h1>Forgotten password</h1>
<fieldset>
    <legend>Please give your email address</legend>
    <%= form_for @user, url: '/users/send_forgotten',
        method: :post do |f| %>
        <div>
            <%= f.label :email %>
            <%= f.text_field :email, size: 30 %>
        </div>
        <%= f.submit "Send" %>
    <% end %>
</fieldset>
```



5. ábra. A profil szerkesztése képernyő

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

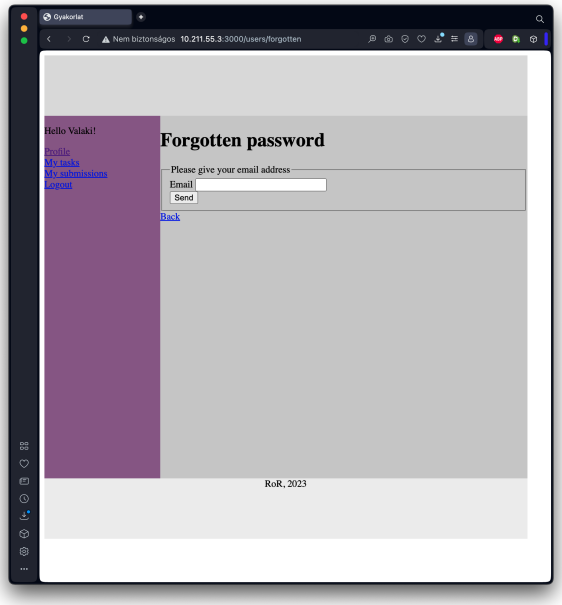
Az elfelejtett jelszó kiküldését a form eseményét kezelő kontroller akció, a `send_forgotten` teszi majd meg, amit fel kell vennünk a kontroller osztályába egyelőre üres törzzsel.

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

  def send_forgotten
  end
end
```

Az elfelejtett jelszó nézetét a 6. ábra mutatja.

Hozzuk létre a házifeladatkezelő portálunk feladatokra vonatkozó modelljét és a hozzá tartozó kontrollert egy paranccsal. A `Task` modellünkben legyen egy egy digiten ábrázolt, egész típusú, `number` nevű, egy string típusú, `url` nevű, egy string típusú, `description` nevű, és egy dátum típusú, `deadline` nevű attribútumunk rendre a sorszám, URL, a feladat szövege és



6. ábra. Az elfelejtett jelszó nézete

határidő tulajdonságokra hivatkozva.

```
kovacsg@debian:~/gyakorlat/app/views/users> rails g
    scaffold task number:integer{1} url:string description:
    string deadline:date
       invoke  active_record
       create    db/migrate/20231017110448_create_tasks.rb
       create    app/models/task.rb
       invoke    test_unit
       create      test/models/task_test.rb
       create      test/fixtures/tasks.yml
       invoke  resource_route
        route    resources :tasks
       invoke  scaffold_controller
       create    app/controllers/tasks_controller.rb
       invoke    erb
       create      app/views/tasks
       create      app/views/tasks/index.html.erb
       create      app/views/tasks/edit.html.erb
       create      app/views/tasks/show.html.erb
       create      app/views/tasks/new.html.erb
       create      app/views/tasks/_form.html.erb
```

```
      create      app/views/tasks/_task.html.erb
      invoke    resource_route
      invoke    test_unit
      create      test/controllers/tasks_controller_test.rb
      create      test/system/tasks_test.rb
      invoke    helper
      create      app/helpers/tasks_helper.rb
      invoke      test_unit
      invoke    jbuilder
      create      app/views/tasks/index.json.jbuilder
      create      app/views/tasks/show.json.jbuilder
      create      app/views/tasks/_task.json.jbuilder
```

Létrejött egy `Task` modell, egy `TasksController` kontroller és a kapcsolódó nézetek: `new` és `edit` egy-egy nézet, amelyek a közös `_form` töredékben lévő formot használják a személy adatainak létrehozására, illetve módosítására. A `show` nézet a személy adatlapját mutatja, a `index` nézet pedig az elérhető személyeket mutatja egy listában.

Hajtsuk végre a scaffold létrehozása során keletkezett migrációt.

```
kovacsg@debian:~/gyakorlat/app/views/users> rails db:
    migrate
== 20231017110448 CreateTasks: migrating
    ======================================
-- create_table(:tasks)
   -> 0.0414s
== 20231017110448 CreateTasks: migrated (0.0419s)
    =============================
```

Ezután a böngészőben nyissuk meg a feladatok nézetet (`http://localhost:3000/tasks`), próbáljuk ki a feladatok létrehozását, törlését, listázását. Nézzük meg, hogy létrejött-e a rekord az adatbázisban.

```
kovacsg@debian:~/gyakorlat/app/views/tasks$ rails db

MariaDB [gyakorlat_development]> select * from tasks;
+----+--------+-------------------------------+-------------------------+


| id | number | url                           | description
              | deadline   | created_at                 |
   updated_at                 |
+----+--------+-------------------------------+-------------------------+


|  1 |      1 | http://gyakorlat.com/tasks/1  | De nehez az
   iskolataska | 2023-10-18 | 2023-10-17 11:11:33.005323 |
```

```
    2023-10-17 11:11:33.005323 |
+----+----------+----------------------------+----------------------------+

1 row in set (0.000 sec)
```

A feladatok adatlapja képernyőn lehetséges a feladat szerkesztése és törlése minden felhasználó számára, ezeket egyelőre tegyük elérhetetlenné egy-egy kommentjel (#) beszúrásával, viszont szükségünk lesz egy megoldás beadása műveletre, amelyhez be kell szúrnunk egy új linket.

```
<div>
  <%#= link_to "Edit this task", edit_task_path(@task) %> |
  <%= link_to "Back to tasks", tasks_path %>
  <%= link_to "Submit solution", '/submissions/new' %>

  <%#= button_to "Destroy this task", @task, method: :
      delete %>
</div>
```

A megoldások tekintetében három képernyőt hozunk létre. A menüből elérhető felhasználói megoldások listáját, a feladat adatlapról elérhető új megoldás, valamint a megoldások listája képernyőről elérhető megoldás adatlap képernyőt. Hozzuk ezeket létre egy új kontroller generálásával.

```
kovacsg@debian:~/gyakorlat/app/views/tasks$ rails g
    controller submissions index show new
       create  app/controllers/submissions_controller.rb
        route  get 'submissions/index'
               get 'submissions/show'
               get 'submissions/new'
       invoke  erb
       create    app/views/submissions
       create    app/views/submissions/index.html.erb
       create    app/views/submissions/show.html.erb
       create    app/views/submissions/new.html.erb
       invoke  test_unit
       create    test/controllers/
          submissions_controller_test.rb
       invoke  helper
       create    app/helpers/submissions_helper.rb
       invoke    test_unit
```

A megoldás egy felhasználó adja be egy feladat tekintetében, hozzuk létre ezt a modellt. Másik típusra hivatkozás típusaként a `references` értéket adjuk meg a modell létrehozásakor.

```
kovacsg@debian:~/gyakorlat/app/views/tasks> rails g model
    submission user:references task:referenes
Could not generate field 'task' with unknown type '
    referenes'.
kovacsg@debian:~/gyakorlat/app/views/tasks> rails g model
    submission user:references task:references
       invoke  active_record
       create    db/migrate/20231017111719
           _create_submissions.rb
       create    app/models/submission.rb
       invoke    test_unit
       create      test/models/submission_test.rb
       create      test/fixtures/submissions.yml
kovacsg@debian:~/gyakorlat/app/views/tasks> rails db:
    migrate
== 20231017111719 CreateSubmissions: migrating
    ===============================================
-- create_table(:submissions)
   -> 0.0222s
== 20231017111719 CreateSubmissions: migrated (0.0226s)
    ======================================
```

Az új megoldás képernyőn el fogunk helyezni egy fájlbeviteli mezőt, ebből az következik, hogy a modellünknek kell rendelkeznie egy attribútummal a fájl tárolására is. Erre később egy új modellt fogunk létrehozni, de addig is felveszük a modell osztályban egy példányváltozót e célra. Ez a példányváltozó jelenleg csak a memóriában létezik, mentés művelet esetén nem kerül ki az adatbázisba az értéke.

```
class Submission < ApplicationRecord
  attr_accessor :attachment
end
```

A feladat adatlapjáról (`show` képernyő) juthatunk el a feladatbeadás képernyőre. Hozzuk létre ezt a formot a `form_with` helperrel. A feladat beadója a bejelentkezett felhasználó lesz, az pedig, hogy a megoldás melyik feladatra vonatkozik, az előző képernyőről fogjuk tudni, így a megoldás két attribútumát ki tudjuk tölteni. A megoldás modell harmadik attribútuma a feltöltött fájl, ehhez vegyünk fel egy mezőt. Az eseménykezelőnk a `create` nevű akció lesz, amelyet HTTP POST művelettel érünk majd el.

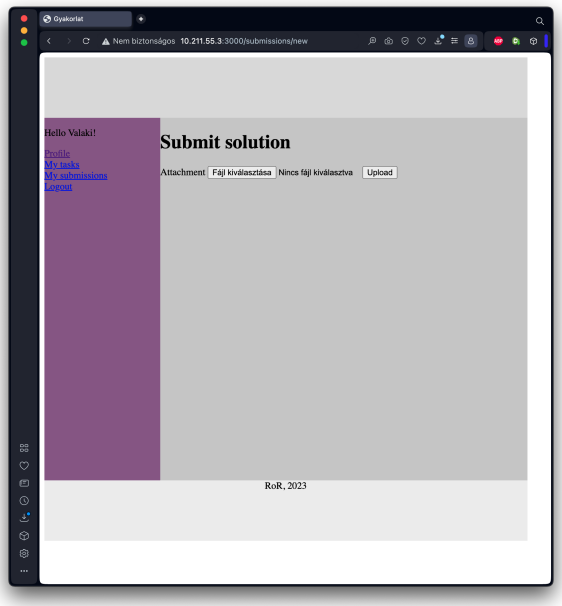
```
<h1>Submit solution</h1>
<%= form_with model: @submission, url: 'submissions/create'
    , method: :post do |f| %>
```

```
    <%= f.label :attachment %>
    <%= f.file_field :attachment %>
    <%= f.submit "Upload" %>
<% end %>
```

A `form_with model` nevű paraméteréhez egy modell osztály egy példányát kell rendelnünk, amely most a `Submission` példánya lesz, amelyet a megoldások kontrollerében kell inicialálnunk. A megoldás beadó felhasználó legyen az első felhasználó az adatbázisból, és a megoldott feladat hasonlóképp legyen az első feladat az adatbázisból.

```
class SubmissionsController < ApplicationController
  def new
    @submission = Submission.new user: User.new, task: Task
        .find(1)
  end
end
```

Az új megoldás beadása nézetének első változatát a 7. ábra mutatja.



7. ábra. Az új megoldás nézetének első változata

A bejelentkezett felhasználó a menüjéből elérheti a megoldások listáját is a harmadik menüelemre kattintva. Készítsük el ezt a képernyőt is. A képernyő neve `index` lesz, ahol egy sorszámozatlan listában (`ul`) megjelenítjük

a felhasználó összes beadott megoldását. A megoldások példányváltozón iterálunk a `each` metódussal, és annak minden egyes elemére létrehozunk egy listaelemet, amely egy linket tartalmaz a megoldás adatlapja képernyőre.

```
<h1>My submissions</h1>

<ul>
<% @submissions.each do |submission| %>
    <li><%= link_to "My submission to task #{submission.
        task.number}", '/submissions/show' %></li>
<% end %>
</ul>
```

A megoldások kontrollerében inicializáljuk ezt a listát egy tömbként, amelyben helyezzünk el a megoldások modell két példányát.

```
class SubmissionsController < ApplicationController
  def index
    @submissions = []
    s1 = Submission.new user: User.first, task: Task.first
    s2 = Submission.new user: User.first, task: Task.first
    @submissions << s1
    @submissions << s2
  end
end
```

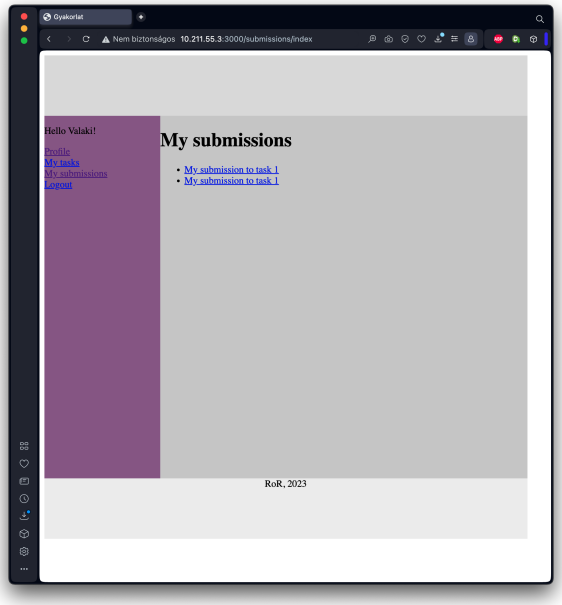
A megoldásaim listája nézetének első változatát a 8. ábra mutatja.

A megoldások listája képernyőről elérhező a megoldás adatlapja, ahol megjelenítjük a feladat sorszámát, leírását, valamint egy linket, amelyen kereszül letölthető a felhasználó saját megoldása.

```
<h1>My submission to task <%= @submission.task.number %></
    h1>
<p>Task:</p><p><%= @submission.task.number %></p><p><%=
    @submission.task.description %></p>
<%= link_to "Download", 'download' %>
<%= link_to "Back", :back %>
```

A megoldások kontrollerében inicializálnunk kell a megjelenítendő megoldás objektumát, amelyen kereszül elérhető lesz a feladat is.

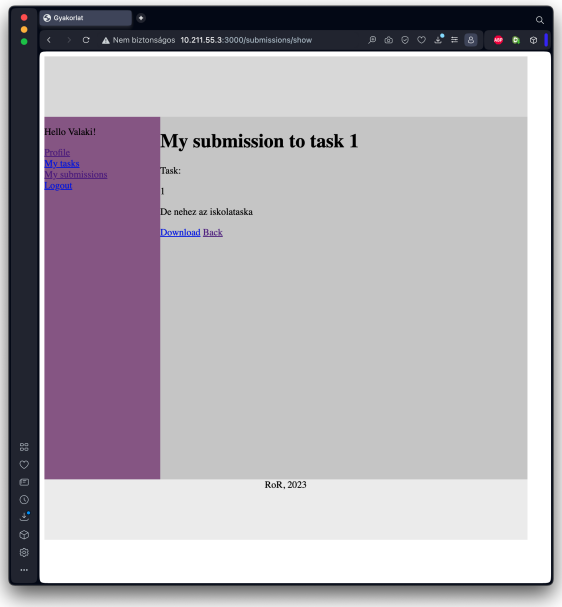
```
class SubmissionsController < ApplicationController
  def show
    @submission = Submission.new user: User.first, task:
        Task.first
  end
```



8. ábra. A megoldásaim listája nézetének első változata

end

A megoldás adatlapja nézetének első változatát a 9. ábra mutatja.



9. ábra. A megoldás adatlapja nézetének első változata