HTML és Rails
Gyakorlat

Kovács Gábor

2024. március 26.

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, 2024</div>
</div>
```

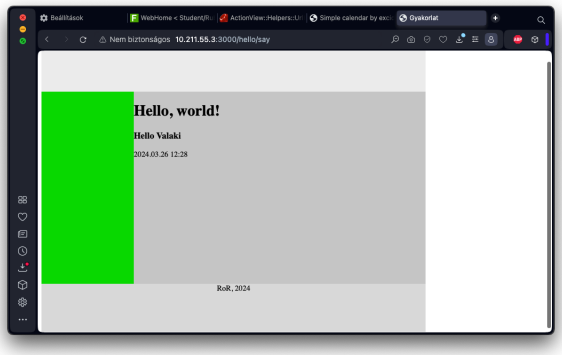
Második lépésként készítsük el az oldal stíluslapját az `app/assets/stylesheets` könyvtárban az `application.css` szerkesztésével. Helyezzünk el benne minimális mennyiségű formázási információt. Az oldal legyen 800 pixel széles. A fejrész legyen világosszürke és 100 képpont magas, és helyezzünk el benn 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
{
    height: 800px;
    width: 800px;
}
div#header
{
    height: 100px;
    background: #eeeeee;
}
div#main {
    height: 600px;

}
div#menu {
    width: 24%;
    float: left;
    background: #0d0;
    height: 100%;
}
div#content {
    width: 76%;
    float: left;
    background: #cccccc;
    height: 100%;
}
div#footer
{
    height: 100px;
    background: #ddd;
    clear: both;
    text-align: center;
  }
```

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.



1. ábra. Az oldal elrendezésének kialakítása

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

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 felhasználó email címére 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.

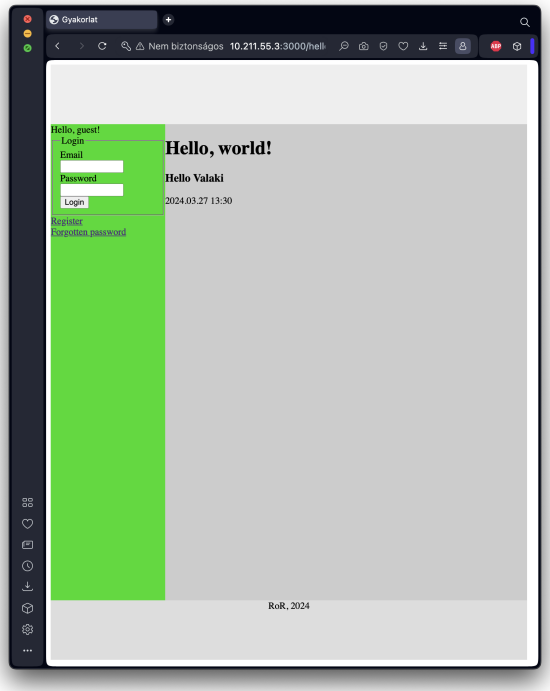
```
Hello, guest!

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

```
        <%= text_field_tag 'email', '', size: 14 %>
        <%= 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 öt akciót definiálunk: a csapatok listájának megtekintését, a saját csapat adatlapjának megtekintését, a foglalható pályák megtekintését, a profiloldal megtekintését és szerkesztését, valamint a kijelentkezést.

```
Hello, user!

<p><%= link_to "Teams", '/teams' %></p>
<p><%= link_to "My team", '/teams/1' %></p>
<p><%= link_to "Fields", 'fields' %></p>
<p><%= link_to "Profile", '/users/edit' %></p>
<p><%= link_to "Logout", '/sessions/destroy' %></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 architektúra szerint szorosan kapcsolódik a nézethez. Hozzuk létre a felhasználó nézetét és fontosabb akcióit a következő paranccsal:

3. ábra. A bejelentkezett felhasználó menüje

```
kovacsg@debian:~/gyakorlat/app> rails g controller users
   new edit forgotten
      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 a felhasználó nevére vonatkozó szövegbeviteli mező a hozzá kapcsolódó címkével, egy a felhasználó email címére vonatkozó szövegbeviteli mező a hozzá kapcsolódó címkével, valamint két 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>Register</h1>

<fieldset>
    <legend>Register a new user</legend>
    <%= form_for @user, url: '/users/create', method: :post
        do |form| %>
        <div>
            <%= form.label :name %>
            <%= form.text_field :name %>
        </div>
        <div>
            <%= form.label :email %>
            <%= form.text_field :email %>
        </div>
        <div>
            <%= form.label :password %>
```

```
            <%= form.password_field :password %>
        </div>
        <div>
            <%= form.label :password_confirmation %>
            <%= form.password_field :password_confirmation
                %>
        </div>
        <%= form.submit "Register" %>
    <% end %>
</fieldset>
<%= link_to "Back", :back %>
```

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

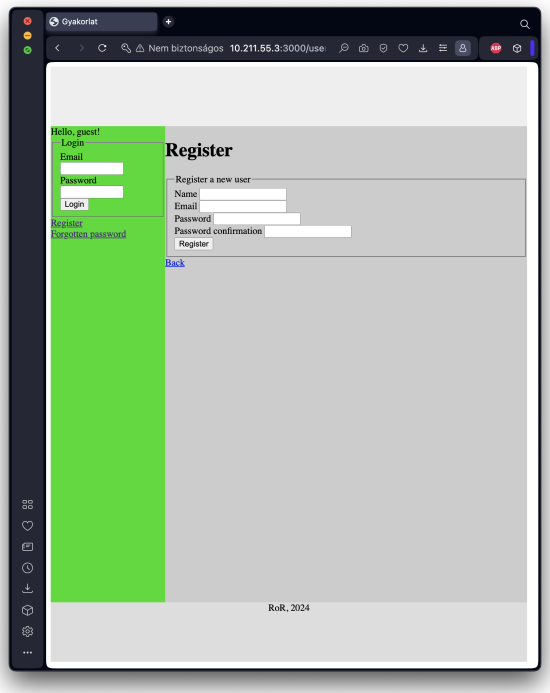
```
class UsersController < ApplicationController
  def new
    @user = User.new
  end
end
```

A felhasználói regisztráció nézetét a 4. ábra mutatja.

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 |form| %>
        <div>
```



4. ábra. A regisztráció nézete

```
            <%= form.label :name %>
            <%= form.text_field :name %>
        </div>
        <div>
            <%= form.label :email %>
            <%= form.text_field :email %>
        </div>
        <div>
            <%= form.label :password %>
            <%= form.password_field :password %>
        </div>
        <div>
            <%= form.label :password_confirmation %>
            <%= form.password_field :password_confirmation
                %>
        </div>
        <%= form.submit "Update" %>
    <% end %>
```

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

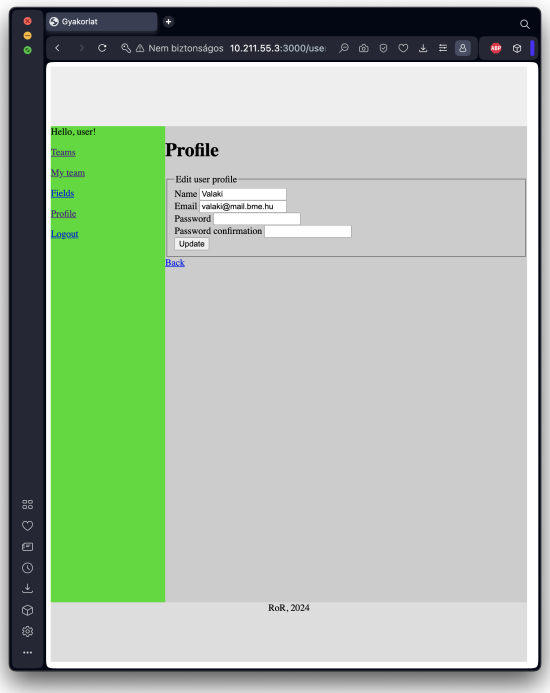
Mivel a `form_for` Rails helper metódust használtuk a form létrehozására a `new` és az `edit` nézetekben, ezért szükséges a megfelelő kontroller akciókban a `@user` példányváltozó inicializálása. Ezeket egyelőre ne adatbázisból tegyük meg, hanem statikus tartalommal töltsük fel. Míg a `new` esetén a 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'
  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>Recover forgotten password</legend>
    <%= form_for @user, url: '/users/recover', method: :
        post do |form| %>
        <div>
            <%= form.label :email %>
            <%= form.text_field :email %>
        </div>
        <%= form.submit "Recover" %>
    <% end %>
</fieldset>
```



5. ábra. A profil szerkesztése képernyő

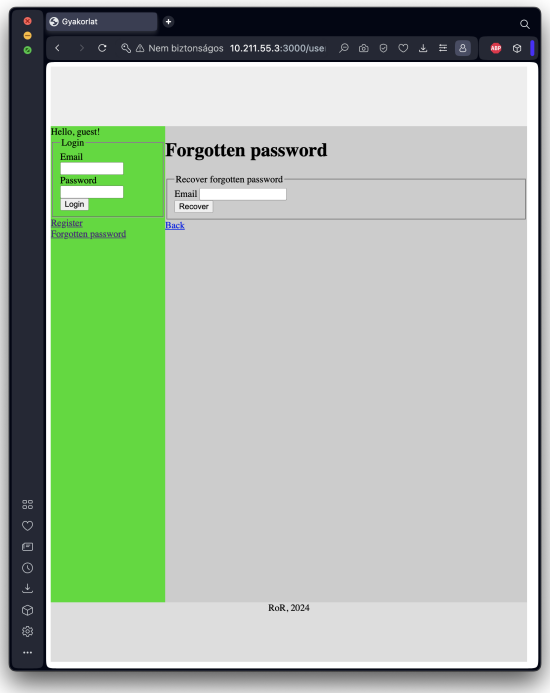
```
<%= link_to "Back", :back %>
```

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

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

  def recover
  end
end
```

Az elfelejtett jelszó nézetét a 8. ábra mutatja.

A bejelentkezett felhasználó menüjének első eleme a csapatok listája képernyő. Hozzuk létre a portálunk csapatokra vonatkozó modelljét és a hozzá tartozó kontrollert egy paranccsal. A `Team` modellünkben legyen egy string



6. ábra. Az elfelejtett jelszó nézete

típusú, `name` nevű, amely a csapat nevét tartja nyilván, és egy felhasználóra hivatkozó, `user` nevű attribútumunk. A gyakorlaton a felhasználóra először hibásan `manager` névvel hivatkoztunk, ezért utólag kiegészítéseket kellett tennünk a modell osztályunkban.

```
kovacsg@debian:~/gyakorlat/app/views/users> rails g
   scaffold team name:string user:references
      invoke  active_record
      create    db/migrate/20240326115734_create_teams.rb
      create    app/models/team.rb
      invoke    test_unit
      create      test/models/team_test.rb
      create      test/fixtures/teams.yml
      invoke  resource_route
       route    resources :teams
      invoke  scaffold_controller
      create    app/controllers/teams_controller.rb
      invoke    erb
      create      app/views/teams
```

```
      create    app/views/teams/index.html.erb
      create    app/views/teams/edit.html.erb
      create    app/views/teams/show.html.erb
      create    app/views/teams/new.html.erb
      create    app/views/teams/_form.html.erb
      create    app/views/teams/_team.html.erb
      invoke  resource_route
      invoke  test_unit
      create    test/controllers/teams_controller_test.rb
      create    test/system/teams_test.rb
      invoke  helper
      create    app/helpers/teams_helper.rb
      invoke    test_unit
      invoke  jbuilder
      create    app/views/teams/index.json.jbuilder
      create    app/views/teams/show.json.jbuilder
      create    app/views/teams/_team.json.jbuilder
```

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

Hajtsuk végre a scaffold létrehozása során keletkezett migrációt.

```
kovacsg@debian:~/gyakorlat/app/views/users> rails db:
    migrate
== 20240326115734 CreateTeams: migrating
    ===========================================
-- create_table(:teams)
   -> 0.0083s
== 20240326115734 CreateTeams: migrated (0.0087s)
    ================================
```

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

Mivel először hibásan `manager`-nek hívtuk a felhasználónkat a scaffold létrehozásakor, a képernyők `user` helyett `manager`-ként hivatkoznak a felhasználóra, ezért a csapat modell osztályban szükségünk lesz néhány új metódusra. A `manager` és a `manager_id` térjen vissza rendre a `user`, illetve a `user_id` értékével.

```
class Team < ApplicationRecord
```

```
  belongs_to :user

  def manager
    user
  end

  def manager_id
    user_id
  end
end
```

A csapatokat egyelőre statikus adatokkal szolgáljuk ki a kontrollerből. A `index` akcióban az adatbázis-műveletet lecseréljük egy tömb összeállítására, amely két csapat típusú objektumot tartalmaz. A csapat adatlapjának megtekintése (`show`) és szerkesztése (`edit`) képernyőkhöz tartozó akciók számára a kontroller törzsének első sorában található `before_action` függvényhívásban szereplő `set_team` nevű függény inicializálja a megjelentítendő csapatot. Ez a `before_action`-ben deklarált függvény akkor fut le, ha az akció neve szerepel az `only` paraméter utáni tömbben, és mind a kettő ilyen. A `set_team` függvény törzsében is lecserléjük az adatbázis-lekérdezést egy helyben példányosított objektumra.

```
class TeamsController < ApplicationController
  before_action :set_team, only: %i[ show edit update
     destroy ]

  # GET /teams or /teams.json
  def index
    #@teams = Team.all
    team1 = Team.new name: "Ballabasok", user: User.first ,
        id: 1
    team2 = Team.new name: "Ketballabasok", user: User.
        first , id: 2
    @teams = [team1, team2]
  end

  private
    # Use callbacks to share common setup or constraints
        between actions.
    def set_team
      #@team = Team.find(params[:id])
      @team = Team.new name: "Ballabasok", user: User.first
          , id: 1
    end
```

```
end
```

Az új csapat létrehozása, illetve a csapat adatlapjának szerkesztése képernyőkön ugyanazta a formot látjuk, amely a `app/views/teams/_form.html.erb` fájlban van. Ebben a felhasználóra az azonosítójával hivatkozhatunk, cseréljük ez le egy legördülő menüre. A `manager` címkét átírtuk `user`-re, és a `text_field` helpert lecseréltük `select`-re. Az utóbbinak második paraméterként átadhatjuk az opciókat a `@user` példányváltozóban.

```
<div>
  <%= form.label :user, style: "display: block" %>
  <%= form.select :user, @users %>
</div>
```

Az opciókat a csapatok kontrollerében állíthatjuk be, ahol felvesszük a `@user` példányváltozót, és egy kétdimenziós tömböt rendelünk hozzá. A belső dimenzió első eleme lesz a címke, a második az érték.

```
class TeamsController < ApplicationController
  # GET /teams/new
  def new
    @team = Team.new
    @users = [["Valaki", 1], ["Senki", 2]]
  end

  # GET /teams/1/edit
  def edit
    @users = [["Valaki", 1], ["Senki", 2]]
  end
end
```

A csapat adatlapon `_team.html.erb` a csapat menedzseréről az azonosítója helyett több információt is megjeleníthetünk, amelyek segítenek a mérkőzésszervezésben: a menedzser nevét kattinthatóvá tesszük, és a link a menedzser email címére való levélírást vált ki.

```
<div id="<%= dom_id team %>">
  <p>
    <strong>Name:</strong>
    <%= team.name %>
  </p>

  <p>
    <strong>Manager:</strong>
    <%= mail_to team.manager.email, team.manager.name %>
```

```
  </p>
</div>
```

A csapatok a mérkőzéseket pályákon játszák, ezért szükségünk lesz a pályák adatait karbantartó képernyőkre is. Ezeket is `scaffold`-dal hozzuk létre. A pályákról nyilvántartjuk a nevüket és a címüket, mindkettők string típusú attribútumként. Hajtsuk egyből végre a migrációt.

```
kovacsg@debian:~/gyakorlat/app/views/users> rails g
    scaffold field name:string address:string
       invoke  active_record
       create    db/migrate/20240326121204_create_fields.rb
       create    app/models/field.rb
       invoke    test_unit
       create      test/models/field_test.rb
       create      test/fixtures/fields.yml
       invoke  resource_route
        route    resources :fields
       invoke  scaffold_controller
       create    app/controllers/fields_controller.rb
       invoke    erb
       create      app/views/fields
       create      app/views/fields/index.html.erb
       create      app/views/fields/edit.html.erb
       create      app/views/fields/show.html.erb
       create      app/views/fields/new.html.erb
       create      app/views/fields/_form.html.erb
       create      app/views/fields/_field.html.erb
       invoke    resource_route
       invoke    test_unit
       create      test/controllers/fields_controller_test.
          rb
       create      test/system/fields_test.rb
       invoke    helper
       create      app/helpers/fields_helper.rb
       invoke      test_unit
       invoke    jbuilder
       create      app/views/fields/index.json.jbuilder
       create      app/views/fields/show.json.jbuilder
       create      app/views/fields/_field.json.jbuilder
kovacsg@debian:~/gyakorlat/app/views/users> rails db:
    migrate
== 20240326121204 CreateFields: migrating
    ==========================================
```

```
-- create_table(:fields)
   -> 0.0117s
== 20240326121204 CreateFields: migrated (0.0124s)
   ==============================
```

Hozzunk létre egy pályát a webfelületen. A pálya adatlapján `app/views/fields/show.html.erb` oldalon megjelenik a pálya neve és címe, és ez alá helyezzük el a pálya fogalhatóságát. A foglalhatóságot táblázatosan jelenítjük meg, amelynek oszlopai lesznek az aktuális hét napjai a sorai pedig a nap órái 8-tól 22-ig. Ahhoz, hogy a pályának legyenek szabad időpontjai először módosítjuk a pálya modell osztályt, felveszünk benne egy `availability` nevű példányváltozót, amely ezeket a szabad időpontokat fogja tárolni.

```
class Field < ApplicationRecord
    attr_accessor :availability
end
```

A pálya kontrollerének `show` akciójában inicializájuk az elérhetőségek tömbjét a nézet számára. A `@field` példányvátozónk az aktuális pálya adatait tartalmazza, és a `set_field` metódusban kerül inicializálásra. A `@matches` példányvátozó azon mérkőzéseket tartalmazza, amelyekre már foglalások történtek. A pálya fent létrehozott `availability` nevű attribútumához kell értéket rendelnünk. Ez egy 15x7 méretű tömb, amely elemeihez 5% valószínűséggel rendeljük hozzá az egyik meccset, ha nincs meccs hozzárendelve egy időponthoz, akkor a pálya foglalható, amit üres stringgel jelölünk a nézet számára.

```
class FieldsController < ApplicationController
  def show
    @matches = ['HUN-TUR', 'HUN-KOS', "BL-KBL"]
    @availability = {}
    for hour in 8..22 do
      @availability[hour] = {}
      for day in Date.today.beginning_of_week..Date.today.
          end_of_week
        szam = rand(100)
        @availability[hour][day] = szam < 5 ? @matches[szam
            %3] : ""
      end
    end
    @field.availability = @availability
  end
end
```

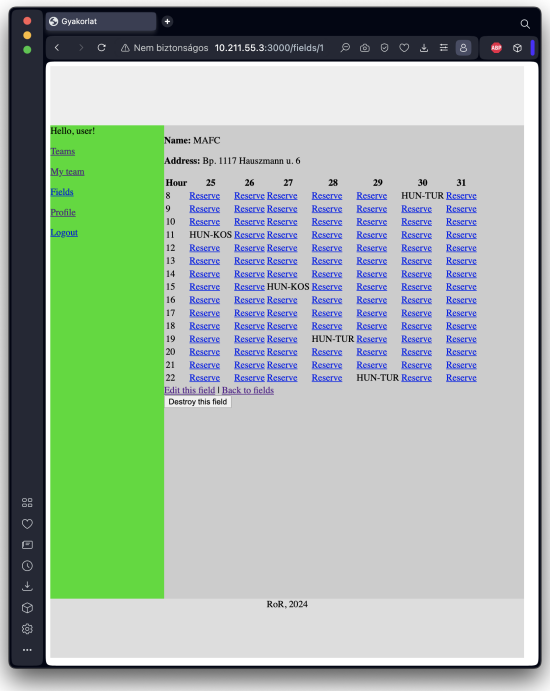
A nézetben egy táblát teszünk ki az adatlap automatikusan generált részei alá. A tábla felrészében a hét napjainak dátumai találhatók, az első oszlopban pedig az órán 8 és 22 között. Ha üres sztringet találunk az egyik cellához tartozóan, akkor oda egy a foglalási képernyőre mutató linket teszünk ki, egyébként pedig a mérkőzés adatait.

```
<table>
    <thead>
        <tr>
            <th>Hour</th>
            <% for col, val in field.availability[field.
                availability.keys[0]] do %>
                <th><%= col.day %></th>
            <% end %>
        </tr>
    </thead>
    <tbody>
    <% for row in field.availability.keys do %>
        <tr>
            <td><%= row %></td>
            <% for col, val in field.availability[row] do
            %>
                <td>
                    <% if val.empty? %>
                        <%= link_to "Reserve", '/fields/
                            reserve' %>
                    <% else %>
                        <%= val %>
                    <% end %>
                </td>
            <% end %>
        </tr>
    <% end %>
    </tbody>
</table>
```

Az pálya adatlapja képernyő ezután így néz ki:

Az adatlapról a foglalás képernyőre juthatunk át, azonban olyan képernyőnk nincs. Az első dolgunk, hogy a fogalalás link kattintható legyen, fel kell vennünk egy útvonalat a `config/router.rb` fájlban. Arról, hogy itt mi történik valójában, később fogunk csak beszélni, egyelőre fogadjuk el magyarázat nélkül, hogy ezt így kell csinálni.

```
Rails.application.routes.draw do
```



7. ábra. Az elfelejtett jelszó nézete

```
  resources :fields do
    get 'reserve', on: :member
  end
end
```

Az a pályák kontrollerében fog egy `reserve` nevű függvényt keresni, amely nem létezik, ezért hozzuk létre. Mivel ez a függvény egy konkrét pálya fefoglalásáról szól, szükségünk van a pálya példányára, amelyet a `set_field` metódus keres elő, ezért a kontroller törzsének első sorában található `before_action` függvényhívás `only` paraméterének tömbjéhez hozzáadjuk ezt a függvényazonosítót is.

Foglalás típusunk egyelőre nincs, helyettesítsük egy egyszerű objektummal, amely magát a foglalást fogja reprezentálni. A foglalás egy nap egy órájára vonatkozik az aktuális pályára valamely mérkőzés számára az aktuálisan bejelentkezett felhasználó által. Ezeket a paramétereket kell átadnunk a kontrollerből a nézetnek. A pálya és a bejelentkezett felhasználó automatikusan elérhető lesz, a többit a `reserve` függvény törzsében inicializáljuk egy statikus értékre. Mérkőzés típusunk sincsen egyelőre, a mérkőzések listája

ezért egy kétdimenziós tömb lesz.

```
class FieldsController < ApplicationController
  before_action :set_field, only: %i[ show edit update
      destroy reserve ]

  def reserve
    @reservation = Object.new
    @hour = 8
    @day = Date.today.day
    @matches = [['HUN-TUR', 1], ['HUN-KOS', 2], ["BL-KBL",
        3]]
  end
end
```

Foglalás képernyőnk sincsen ezért létrehozunk egy `reserve.html.erb` fájlt a pálya képernyőinek könyvtárában. Képernyő egy formot tartalmaz, amelyen a felhasználó kiválaszthatja, hogy az aktuális pályát mely mérkőzés számára akarja lefoglalni. A dátum és az időpont a pálya adatlapon kiválaszott cella alapján már ismert, értékeiket `hidden` típusú beviteli mezőben tároljuk, a mérkőzés számára pedig egy legördülő menüt teszünk ki.

```
<h1>Reserve field</h1>
<fieldset>
    <legend>Reserve field for match</legend>
    <%= form_tag '/field/reservation/create', method: :post
        do %>
     <%= hidden_field_tag :field, @field.id %>
     <%= hidden_field_tag :user, User.first.id %>
     <div>
        <%= label_tag :day %>
        <%= @day %>
        <%= hidden_field_tag :day, @day %>
     </div>
     <div>
        <%= label_tag :hour %>
        <%= @hour %>
        <%= hidden_field_tag :hour, @hour %>
     </div>
     <div>
        <%= label_tag :match %>
        <%= select_tag :match, options_for_select(@matches)
            %>
     </div>
     <%= submit_tag 'Reserve' %>
```

```
    <% end %>
</fieldset>
```

Az pályafoglalás képernyő ezután így néz ki:

8. ábra. Az elfelejtett jelszó nézete