HTML és Rails
Gyakorlat

Kovács Gábor

2016. október 11.

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="page">
    <div id="menu">
      <% if logged_in? %>
        <%= render 'layouts/usermenu' %>
      <% else %>
        <%= render 'layouts/guestmenu' %>
      <% end %>
    </div>
    <div id="main">
        <%= yield %>
    </div>
</div>
<div id="footer">Copyright ROR, 2016</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. A menüsávot a fejléc alatt helyezzük el, és az vízszintesen a szélesség 24%-át foglalja el, és legyen 600 pixel magas. Az oldal tartalmi része legyen 600 képpont magas, világosszürke háttérrel rendelkezzen, és a

menütől jobbra helyezkedjen el a vízszintesen a szélesség 76%-át elfoglalva. A lábrészben a szöveget igazítsuk középre.

```
#page {
    width: 100%;
}
#header {
    height: 100px;
    background-color: #dddddd;
}
#footer {
    height: 100px;
    background-color: #00dd00;
    text-align: center;
    clear: both;
}
#menu {
    background-color: #cccccc;
    height: 600px;
    width: 24%;
    float: left;
}
#main {
    background-color: #eeeeee;
    height: 600px;
    width: 76%;
    float: left;
}
```

Az így kialakított elrendezést például az 1. ábra mutatja.

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 teszünk.

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 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

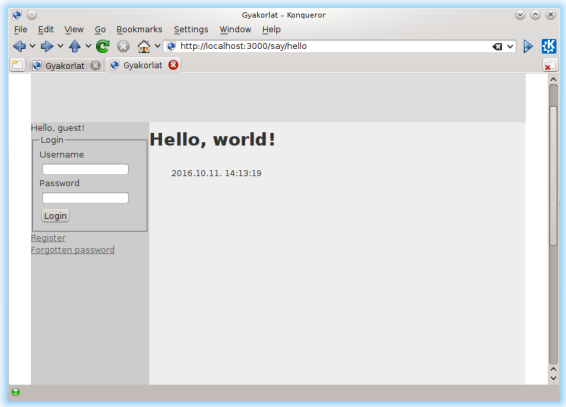
kívánjuk elküldeni. A form mezőit rendre a `label_tag`, `text_field_tag`, `password_field_tag` és `submit_tag` helperekkel hozzuk létre, és a beviteli mezőket 16 karakter hosszúra korlátozzuk. A be nem jelentkezett felhasználónak tegyük lehetővé az elfelejtett jelszó visszaszerzését, ezt egy link hozzáadásával tesszük meg.

```
Hello, guest!
<fieldset>
  <legend>Login</legend>
  <%= form_tag '/sessions/create', :method => :post do %>
    <%= label_tag :username, "Username" %>
    <%= text_field_tag :username, '', size: 16 %>
    <%= label_tag :password, "Password" %>
    <%= password_field_tag :password, '', size: 16 %>
    <%= 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/guestmenu" %>
```

A vendégfelhasználó menüjének megvalósítását a 1. ábra mutatja.



1. 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. 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, hogy ellenőrizhessük, hogy a megfelelő menü jelenik-e meg a vendég és a felhasználó számára.

```
<% if logged_in? %>
  ...
<% else %>
  ...
<% end %>
```

A bejelentkezett felhasználó menüjét a vendéghez hasonlóan beágyazott nézettel hozzuk létre. Egyelőre két akciót definiálunk: a profiloldal szerkesztését és a kijelentkezést.

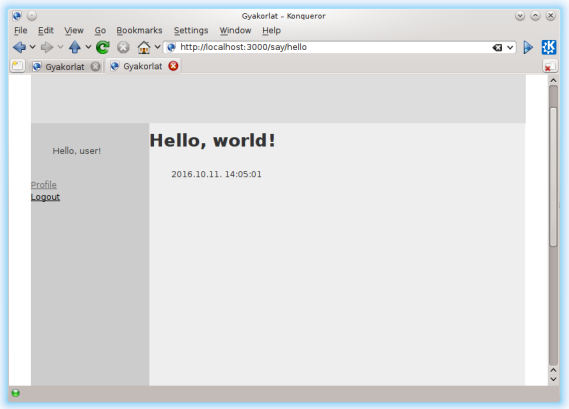
```
<p>Hello, user!</p>
<%= link_to "Profile", '/users/edit' %><br/>
<%= link_to "Logout", '/sessions/destroy' %>
```

A bejelentkezett felhasználó menüjének megvalósítását a 2. ábra mutatja.

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 show 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 `edit`, a felhasználói adatait megjelenítő `show`, 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.



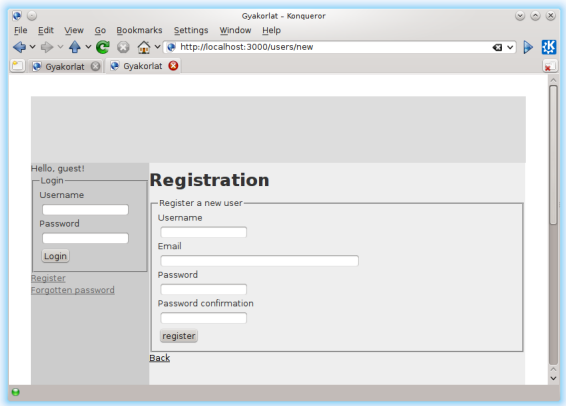
2. ábra. A bejelentkezett felhasználó menüje

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 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 16 karakter széles felhasználó nevére vonatkozó szövegbeviteli mező a hozzá kapcsolódó címkével, egy 40 karakter széles email címre vonatkozó szövegbeviteli mező a hozzá kapcsolódó címkével, két darab 16 karakter széles jelszóbeviteli mező a hozzájuk kapcsolódó címkével, a két jelszómező eltérő azonosítóval rendelkezzék, az egyik prefixe `_confirmation`-re végződjék. Az oldal alján helyezzünk el egy visszalépés gombot azon felhasználóknak, akik véletletül tévedtek ide. A `:back` szimbólummal hivatkozott értékben a Rails a `HTTP_REFERER` HTTP fejléc elem vagy a Javascript `history` attribútuma alapján tárolja a megelőző HTTP kérésben használt URI-t.

```
<h1>Registration</h1>
<%= flash[:notice] %>

<fieldset>
  <legend>Register a new user</legend>
  <%= form_for :user, url: {:action => :create}, method: :
```

```
    post do |f| %>
  <div>
    <%= f.label :username %>
    <%= f.text_field :username, size: 16 %>
  </div>
  <div>
    <%= f.label :email %>
    <%= f.text_field :email, size: 40 %>
  </div>
  <div>
    <%= f.label :password %>
    <%= f.password_field :password, size: 16 %>
  </div>
  <div>
    <%= f.label :password_confirmation %>
    <%= f.password_field :password_confirmation, size: 16
        %>
  </div>
  <%= f.submit :register %>
<% end %>
</fieldset>

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

A felhasználói regisztráció nézetét a 3. ábra mutatja.



3. 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 hashére 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 formok eseménykezelőihez a generate controller parancsunk nem generált útvonalakat, ezért azokat felvesszük a `config/routes.rb`-be. Jelenleg a nézetek kialakításához szingleton példányokkal dolgozunk, vagyis egyetlen felhasználó adatait vagyunk képesek megjeleníteni. Később szükségünk lesz egy `:id` szimbólumra, ami képessé tesz minket a felhasználók megkülönböztetésére, és azonos nevű paraméterként jelentkezik majd a kontrollerben.

```
get 'users/new'
post 'users/create', to: 'users#create'

get 'users/edit'
put 'users/update', to: 'users#update'

get 'users/show'

get 'users/forgotten'
post 'users/send_forgotten', to: 'users#send_forgotten'
```

Az `as` opció használatával az útvonalakat reprezentáló stringek karbantarthatóságán javíthatunk, így az csakis a `config/routes.rb` fájlban kell módosítanunk ezután, és nem az összes nézetben, ahol előfordul. A gyakorlatban minden útvonalra érdemes ilyen helper függvényt definiálnunk. A `'users/create'` stringet a `create_user_path` metódus állítja elő, ami az `as` opció és a `_path` vagy `_url` posztfix összefűzéséből adódik, és URL típusú függvényparaméterek helyett használható.

A form eseményét a Rails konvenció szerint a `create` kontroller metódus fogja kezelni. Ez még nem létezik, ezért definiáljuk azt egyelőre üres törzzsel.

A felhasználói profil szerkesztésének nézetében (`edit.html.erb`) található form szinte teljesen megegyezik az új felhasználót létrehozó formmal. A felhasználónév módosítását inaktívvá tehetjük, illetve az eseménykezelő kontroller akciót kell módosítanunk. A form eseményét a Rails konvenció szerint a `update` kontroller metódus fogja kezelni, erre létrehozzuk az útvonalat, és a `update_user_path` helpert. Ez még nem létezik, ezért definiáljuk ezt is egyelőre szintén üres törzzsel. Ezen kívül a nézetben a feliratokat kell még

átírnunk regisztrációról profil szerkesztésére.

```
<h1>Profile</h1>
<%= flash[:notice] %>

<fieldset>
  <legend>Edit user profile</legend>
  <%= form_for :user, url: {:action => :update}, method: :
      put do |f| %>
    <div>
      <%= f.label :username %>
      <%= f.text_field :username, size: 16 %>
    </div>
    <div>
      <%= f.label :email %>
      <%= f.text_field :email, size: 40 %>
    </div>
    <div>
      <%= f.label :password %>
      <%= f.password_field :password, size: 16 %>
    </div>
    <div>
      <%= f.label :password_confirmation %>
      <%= f.password_field :password_confirmation, size: 16
          %>
    </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.

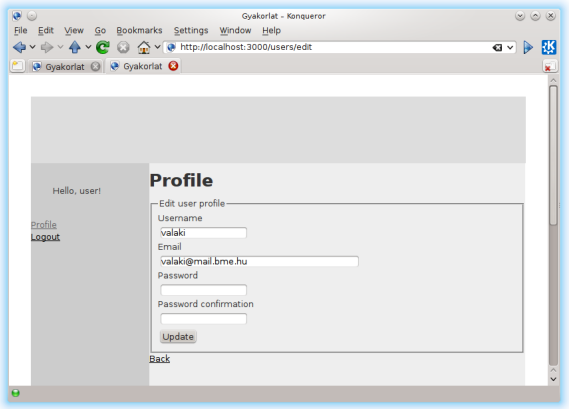
```
class UsersController < ApplicationController
  def show
    @user = User.new
```

```
    @user.username = 'valaki'
    @user.email = 'valaki@mail.bme.hu'
    @user.id = 1
  end

  def new
    @user = User.new
  end

  def create
  end

  def edit
    @user = User.new
    @user.username = 'valaki'
    @user.email = 'valaki@mail.bme.hu'
    @user.id = 1
  end

  def update
  end

  def forgotten
  end

  def send_forgotten
  end
end
```

A felhasználói profiloldal szerkesztésének nézetét az 4. 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.

```
<h1>Forgotten password</h1>
<%= flash[:notice] %>

<fieldset>
  <legend>Please give your email address</legend>
  <%= form_for :user, url: {:action => :send_forgotten},
      method: :post do |f| %>
    <div>
      <%= f.label :email %>
```



4. ábra. A profiloldal nézete

```
      <%= f.text_field :email, size: 40 %>
    </div>
    <%= f.submit 'Send' %>
  <% 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. Ha a felhasználó meggondolná magát, és megsem kívánná elküldetni magának a jelszavát, egy `Back` feliratú linkkel biztosítjuk számára a lehetőséget az előző oldalra való visszatérésre.

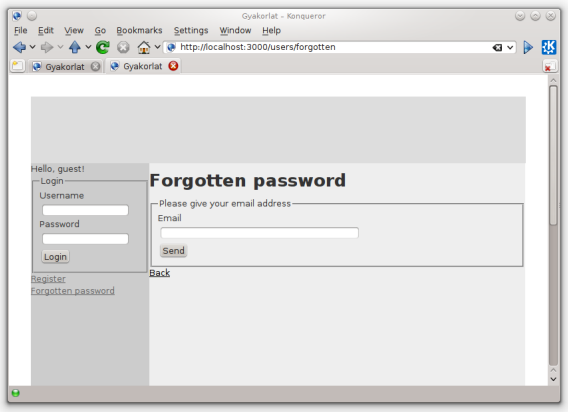
Az elfelejtett jelszó nézetét a 5. ábra mutatja.

A kérdőívszerkesztő portálunk központi elemei maguk a kérdőívek, amelyeket a létrehozó felhasználójuk szerkeszthet. Egy kérdőívnek van neve, leírása, létrehozási ideje, szerzője és sok kérdésből áll. Először definiáljuk a kérdőív modelljét, és létrehozzuk hozzá a nézeteket.

```
rails generate scaffold questionnaires title:string
   description:text user:references
```

A `scaffold` művelet egyszerre hozza létre a modellt és a hozzá kapcsolódó, a REST konvenciónak megfelelő események halmazát, amit a `routes.rb`-ben a `resources :questionnaires` bejegyzés definiál. Arról, hogy ez konkrétan mit, milyen útvonalakat jelent, később esik majd szó.

A kérdőívek modellje mellett létrejött egy kontroller és jó néhány előre



5. ábra. Az elfelejtett jelszó nézete

definiált nézet. Nézzük meg a létrejött nézeteket, és, hogy hogyan néznek ki a megjelenítendő adatok! 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 `Questionnaire` 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 bejegyzés azonosítójával, így az mindig a táblázat megfelelő sorában megjelenő bejegyzésre vonatkozik. Az oldal alján egy új bejegyzés létrehozását lehetővé tevő nézetre mutató link található. E linkeket használjuk majd fel a felhasználók profil oldalán.

A `new` és az `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 `Questionnaire` adatstruktúrája mezői típusainak megfelelő adatbeviteli mezők jelennek meg egy a mező nevével megegyező címke után. Ezt a későbbiekben módosítjuk, ugyanis a kérdőívet létrehozó felhasználó nem állítható. Minden típus előre definiált adatbeviteli mezőre képeződik le, csak string típusú mezőink vannak, azok szövegbeviteli mezővel szerkeszthetők. Az oldalak tetején megjelenhetnek az esetleges hibaüzenetek. A `show` nézet a szó adatstruktúra mezőinek értékeit írja ki.

Töltsük fel az `index`, `edit` és a `show` nézeteket adatokkal! A kontroller osztály második sorában lévő `before_action` függvényhívás azt mondja a kontroller számára, hogy mielőtt végrehajtaná az `only` kulcs utáni listában szereplő metódusokat, futtassa el a `set_questionnaire` nevű privát metódust. Így az inicializációt, amely alapesetben egy rekord előkeresését jelenti az adatbázisból, csak egy helyen kell leírnunk, karbantartanunk, és nem az összes metódus elején. Definiáljunk ebben a metódusban kezdeti adatokat. Az `edit` nézet esetén azt látjuk, hogy az adatstruktúra mezői értéke alapján inicializálódtak a form adatbeviteli mezői. Az `index` akció nem szerepel a `only` utáni felsorolásban, ezért ott külön kell elvégeznunk a példányváltozó inicializációját.

```
class QuestionnairesController < ApplicationController
  before_action :set_questionnaire, only: [:show, :edit, :
     update, :destroy]
  def index
    #@questionnaires = Questionnaire.all
    q1 = Questionnaire.new
    q1.id = 1
    q1.title = "Ime, az elso"
    q1.description = "Hosszu szoveg ide"
    q1.user_id = 1
    q2 = Questionnaire.new
    q2.id = 2
    q2.title = "Ime, a masodik"
    q2.description = "Hosszu szoveg ide"
    q2.user_id = 2
    @questionnaires = [q1, q2]
  end
  ...
  private
    def set_questionnaire
      @questionnaire = Questionnaire.new
      @questionnaire.id = 1
      @questionnaire.title = "Ime, az elso"
      @questionnaire.description = "Hosszu szoveg ide"
      @questionnaire.user_id = 1
      @questionnaire
      #@questionnaire = Questionnaire.find(params[:id])
    end
end
```

Egy kérdőív sok kérdésből áll, ezért azok számára is létre kell hoznunk

egy modellt. Egy kérdés pontosan egy kérdőívhez tartozik, fel kell vennünk egy idegen kulcsot. A kérdésnek van egy típusa, ami lehet szabad szöveg, listából választás, vagy pontozás, ennek megadására felveszünk egy típus mezőt. Végül a kérdésnek van egy sorszáma is a kérdőíven belül.

Egy kérdésre több válasz is adható például választás típusú kérdés esetén, ezért szükségünk vagy egy további típusra is, a válaszra. A válasz tehát idegen kulccsal hivatkozni fog a kérdésre, a válasz rendelkezik egy szöveggel, és a válasznak is lehet sorszáma.

Hozzuk létre ezt a két modellt a kérdőív mintájára:

```
rails generate scaffold questions questionnaire:references
   question:string type:integer{1} index:integer{2}
rails generate scaffold answers question:references answer:
   string index:integer{2}
```

Mivel modelleket is létrehoztunk, létrejött egy migráció is. Hajtsuk végre az adatbázissémánk módosítását!

```
rake db:migrate
```

Ezután a felületen ellenőrizhetjük e három modell nézeteit. A kérdőív esetén az `index` nézet a `/questionnaires`, a `show` nézet a `/questionnaires/1`, a `edit` nézet pedig a `/questionnaires/1/edit` útvonalon lesz elérhető a portálon. A másik két modell, vagyis a kérdések és a válaszok esetén hasonló módon csalogathatók elő a megtervezett nézetek.