HTML és Rails
Gyakorlat

Kovács Gábor

2016. március 22.

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="menu">
  <% if logged_in? %>
    <%= render 'layouts/user_menu' %>
  <% else %>
    <%= render 'layouts/guest_menu' %>
  <% end %>
</div>

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

<div id="footer">
  Copyright, RoR, 2016
</div>
```

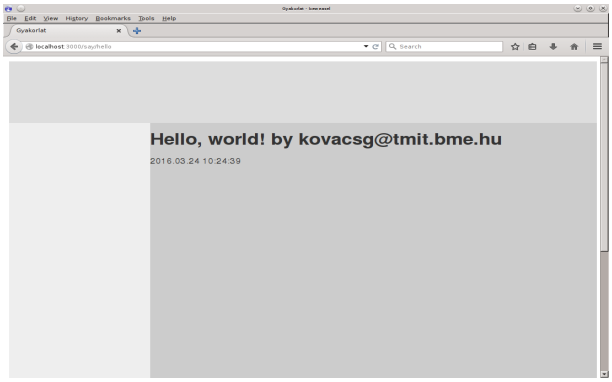
```
</div>
```

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. 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: 800px;
}

#header {
  background-color: #dddddd;
  height: 100px;
}

#menu {
  background-color: #eeeeee;
  height: 600px;
  width: 24%;
  float: left;
}

#main {
  background-color: #cccccc;
  height: 600px;
  width: 76%;
  float: left;
}

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

Az így kialakított elrendezést az 1. ábra mutatja.

Háromféle felhasználóra készülünk fel egyelőre, egy látogatóra, és egy belépett felhasználóra, aki korábban keresztülment egy regisztrációs folya-



1. ábra. Az oldal elrendezésének kialakítása

maton, és egy adminisztrátorra. A látogató csak böngészhet, ugyanakkor regisztrálhat. A bejelentkezett felhasználó vagy az adminisztrátor számára több funkciót is elérhetővé teszünk. Az adminisztrátor funkcióinak megvalósítását egyelőre talonba tesszük.

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.

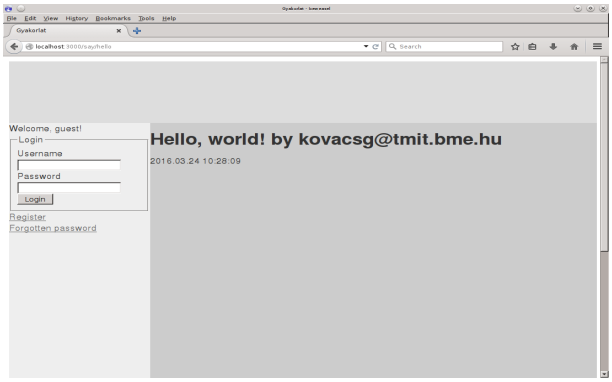
```
Welcome, guest!

<fieldset>
  <legend>Login</legend>
  <%= form_tag 'sessions/create', method: :post do %>
    <%= label_tag :username, "Username" %><br/>
    <%= text_field_tag :username, '', size: 18 %><br/>
    <%= label_tag :password, "Password" %><br/>
    <%= password_field_tag :password, '', size: 18 %><br/>
```

```
    <%= submit_tag "Login" %>
  <% end %>
</fieldset>
<%= link_to "Register", '/users/new' %><br/>
<%= link_to "Forgotten password", '/users/forgotten' %><br/
    >
```

Ezután a `menu` azonosítójú `div`-ben meghivatkozhatjuk ezt az oldalt. A Rails konvenció szerint az aláhúzásjelet el kell hagynunk.

```
<%= render "layouts/guest_menu" %>
```

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. Itt egyelőre manuálisan állítjuk, hogy be van-e jelentkezve a felhasználó. A metódus értelemszerűen boolean visszatérési értékű. A belejentkezett felhasználóról pedig egy-egy függvénnyel eldöntjük, hogy felhasználó vagy adminisztrátor-e.

```
module ApplicationHelper
  def logged_in?
    true
  end
end
```

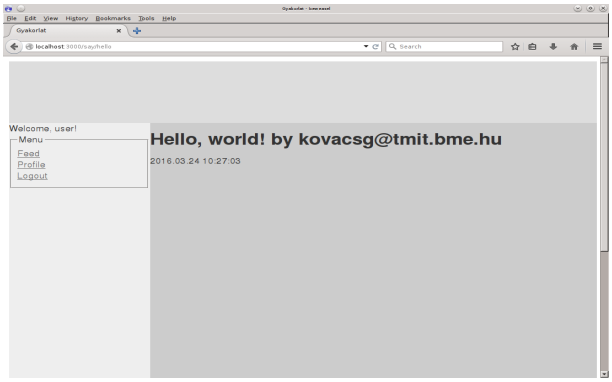
Ezt visszavezetve a keretbe a menüt megvalósító `menu`-ban a következő módosítást végezzük el. Így a helper módosításával be, illetve ki tudunk lépni az oldalról. „Jelentkeztessük” be és ki a felhasználót, hogy ellenőrizhessük, hogy a megfelelő menü jelenik-e meg a vendég és a felhasználó számára.

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

A bejelentkezett felhasználó menüjét a vendéghez hasonlóan beágyazott nézettel hozzuk létre. Egyelőre három akciót definiálunk: az üzenőfal megtekintése, a profiloldal szerkesztése és a kijelentkezés.

```
Welcome, user!

<fieldset>
  <legend>Menu</legend>
  <%= link_to "Feed", '/users/show' %><br/>
  <%= link_to "Profile", '/users/edit' %><br/>
  <%= link_to "Logout", '/sessions/destroy' %>
</fieldset>
```

A bejelentkezett felhasználó menüjének megvalósítását a 3. ábra mutatja.



3. ábra. A bejelentkezett felhasználó menüje

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
```

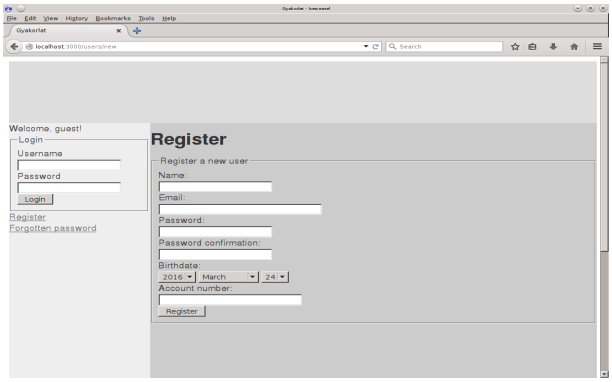
A parancs futtatásával létrejött az `users` kontroller és a hozzá kapcsolódó nézetek köztük az új felhasználó létrehozását lehetővé tevő `new`, a felhasználói profil szerkesztését megvalósító `edit`, és a felhasználói adatait megjelenítő `show`. Tervezői kérdés, hogy az elfelejtett jelszó kezelését a felhasználók kontrollere részének tekintjük, vagy önálló kontrollert hozunk létre számára. A gyakorlaton később 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 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 hat elem rendre a következő: egy húsz karakter széles felhasználó nevére vonatkozó szövegbeviteli mező a hozzá kapcsolódó címkével, egy harminc karakter széles email címre vonatkozó szövegbeviteli mező a hozzá kapcsolódó címkével, két darab 20 karakter széles jelszóbeviteli mező a hozzájuk kapcsolódó címkével, a két jelszómező eltérő azonosítóval rendelkezzék, az egyik prefixe `_confirmation`-re végződjék, egy dátumválasztó és a kapcsolódó címke a születésnap megadására, egy 26 karakter széles szövegbeviteli mező a bankszámlaszám részére a kapcsolódó címkével, és végül egy nyomógomb. 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>Register</h1>
<%= flash[:notice] %>

<div>
  <fieldset>
    <legend>Register a new user</legend>
    <%= form_for :user, url: { action: :create }, method: :
        post do |f| %>
      <div>
        <%= f.label :name %>:<br/>
        <%= f.text_field :name, :size => 20 %>
      </div>
      <div>
```

```
        <%= f.label :email %>:<br/>
        <%= f.text_field :email, :size => 30 %>
      </div>
      <div>
        <%= f.label :password %>:<br/>
        <%= f.password_field :password, :size => 20 %>
      </div>
      <div>
        <%= f.label :password_confirmation %>:<br/>
        <%= f.password_field :passwor_confirmation, :size =
            > 20 %>
      </div>
      <div>
        <%= f.label :birthdate %>:<br/>
        <%= f.date_select :birthdate %>
      </div>
      <div>
        <%= f.label :account_number %>:<br/>
        <%= f.text_field :account_number, :size => 26 %>
      </div>
      <%= f.submit "Register" %>
    <% end %>
  </fieldset>
</div>

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

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 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/show', as: 'feed'
  get 'users/new'
  post 'users/create', as: 'create_user'
  get 'users/edit'
  put 'users/update'
  get 'users/forgotten'
  post '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 `create_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>
```

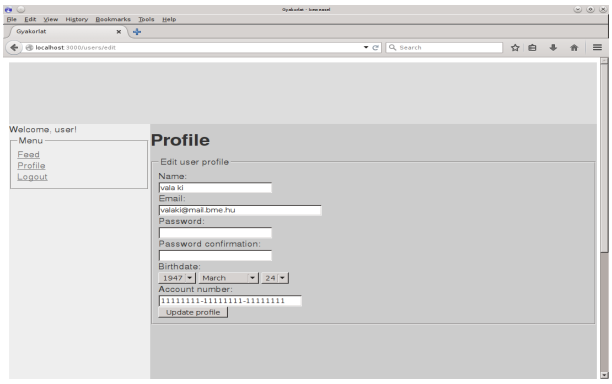
```
<%= flash[:notice] %>

<div>
  <fieldset>
    <legend>Edit user profile</legend>
    <%= form_for :user, url: { action: :update }, method: :
        put do |f| %>
      ...
      <div>
        <%= f.submit "Update profile" %>
      </div>
    <% end %>
  </fieldset>
</div>

<%= 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
   @post = Post.new
  end

  def new
    @user = User.new
  end

  def create
  end

  def edit
    @user = User.new
    @user.id = 1
    @user.name = 'vala_ki'
    @user.email = 'valaki@mail.bme.hu'
    @user.account_number = '11111111-11111111-11111111'
```

```
    @user.birthdate = Time.now - 69.years
  end

  def update
  end

  def forgotten
  end

  def send_forgotten
  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 profiloldal nézete

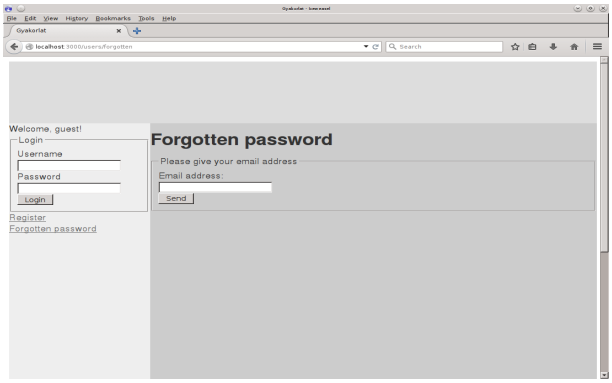
Ezután alakítsuk ki az elfejetett jelszó oldalt is. Itt egyszerűbb a formunk a beléptetésnél, csak az email címet tartalmazza.

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

<div>
  <fieldset>
    <legend>Please give your email address</legend>
    <%= form_tag '/users/send_forgotten', method: :post do
        %>
      <div>
        <%= label_tag :email, "Email address" %>:<br/>
```

```
        <%= text_field_tag :email %>
      </div>
      <%= submit_tag "Send" %>
    <% end %>
  </fieldset>
</div>

<%= 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 6. ábra mutatja.



6. ábra. Az elfelejtett jelszó nézete

A felhasználói üzenőfalon legyenek megtekinthető a bejegyzések, és lehessen új bejegyzést tenni. Először definiáljuk a bejegyzés modelljét, és létrehozzuk hozzá a nézeteket.

```
rails generate scaffold posts text:string author_id:integer
    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 :posts` bejegyzés definiál. Arról, hogy ez konkrétan mit, milyen útvonalakat jelent, később esik majd szó.

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

```
rake db:migrate
```

A bejegyzések modellje mellett létrejött egy kontroller és jó néhány előre 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 `Post` 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 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 `Post` adatstruktúrája mezői típusainak megfelelő adatbeviteli mezők jelennek meg egy a mező nevével megegyező címke után. Ezt a későbbiekben módosítjuk, ugyanis sem a bejegyzést tevő felhasználó, sem a bejegyzés üzenőfalát birtokló 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 `edit` és a `show` nézeteket adatokkal, vagyis a `@post` kontroller példányvátozóit is inicializáljuk a `set_post` nevű privát metódusban, majd ellenőrzizhetjük, hogy megjelennek-e a nézeten ezek az adatok. Az `edit` nézet esetén azt látjuk, hogy az adatstruktúra mezői értéke alapján inicializálódtak a form adatbeviteli mezői..

```
def set_post
  #@post = Post.find(params[:id])
  @post = Post.new text: "Az elet csodaszep", user_id: 1,
      id: 1
end
```

A `new` és `show` nézetek beágyazásához létre kell hoznunk azok `_` jellel prefixált változatát, ahogy azt a menük beágyazása esetében láttuk. Így a nézet beágyazható, és közvetlenül is elérhető marad. A `new` esetén a tartalalom maradjon, azonban a `show` nézetet módosítsuk:

```
<p>
  <strong>Valaki left a message at <%= @post.updated_at %><
      /strong>
  <%= @post.text %>
</p>
```

A felhasználó üzenőfala bejegyzéseket fog tartalmazni, ide ágyazunk be a bejegyzés nézetét és a bejegyzést létrehozó formot.

```
<%= render '/posts/1' %>

<%= render '/posts/new' %>
```

A `show` nézet számára inicializálnunk kell egy megjelenítendő bejegyzést példányváltozót, amit megjeleníthetünk. A következő gyakorlaton innen folytatjuk.

```
def show
    @post = Post.new
end
```