# HTML és Rails
# Gyakorlat

Kovács Gábor

2019. március 12.

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="body">
    <div id="menu"></div>
    <div id="main">
      <%= yield %>
    </div>
  </div>
  <div id="footer">Copyright, RoR 2019</div>
</div>
```

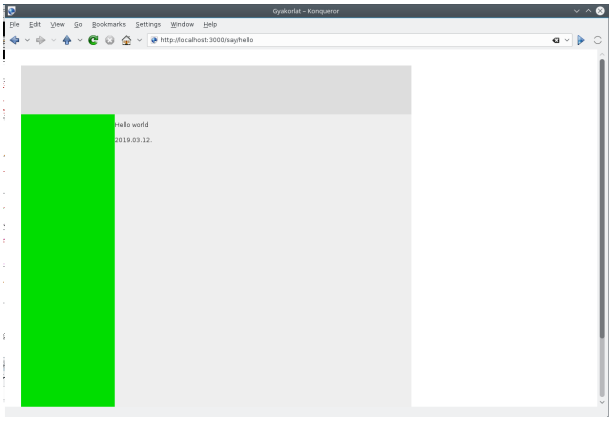
Második lépésként készítsük el az oldal stíluslapját, amivel ezek a helyükre kerülnek, és helyezzünk el benne minimális mennyiségű formázási információt. Az oldal legyen 800 pixel széles. A fejrész legyen világosszürke és 100 képpont magas. Az oldal központi része legyen 600 pixel magas. A menüsávot a központi részben, a fejléc alatt helyezzük el, és az vízszintesen a szélesség 24%-át foglalja el, magasságát a központi rész magassága definiálja. Az oldal tartalmi része világosszürke háttérrel rendelkezzen, és a menütől jobbra helyezkedjen el a vízszintesen a szélesség 76%-át elfoglalva. A lábrészben a szöveget igazítsuk középre, és legyen az is 100 pixel magas, valamint a fejlécnél világosabb szürke színű.

```
div#page {
    width: 800px;
}

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

Az így kialakított elrendezést például az 1. ábra mutatja.

Két felhasználóra készülünk fel egyelőre, egy látogatóra és egy belépett felhasználóra, aki korábban keresztülment egy regisztrációs folyamaton. A látogató csak böngészhet, bejelentkezhet és jelszóemlékeztetőt kérhet. 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 Rails konvenció szerint a részlegesen renderelt állományok neve aláhúzásjellel kezdődik. Legyen a fájlunk neve ezért `_guest_menu.html.erb`! A form tartalmazzon egy felhasználónévre utaló címkét és egy szövegbeviteli mezőt,




1. ábra. Az oldal elrendezésének kialakítása

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 18 karakter hosszúra korlátozzuk. A be nem jelentkezett felhasználónak tegyük lehetővé az elfelejtett jelszó visszaszerzését, ezt egy link hozzáadásával tesszük meg.

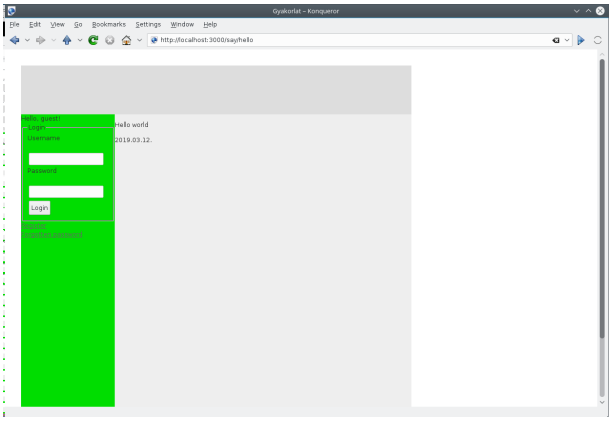
```
Hello, guest!

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

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_menu' %>
```

```
</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. 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.

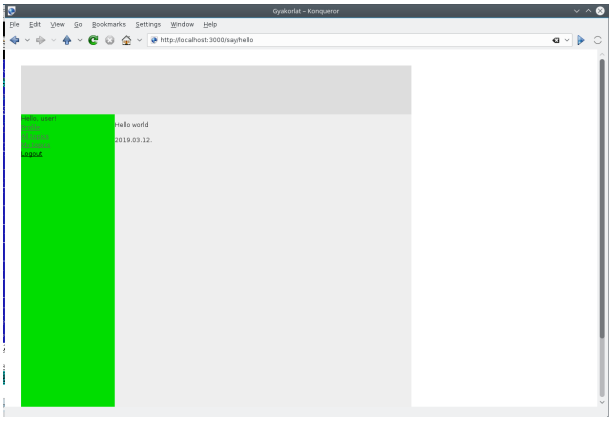
```
<% if logged_in? %>
  ...
<% else %>
  ...
<% end %>
```

A bejelentkezett felhasználó menüjét a vendéghez hasonlóan beágyazott nézettel hozzuk létre (`_user_menu.html.erb`). Egyelőre négy akciót defini-

álunk: a profiloldal megtekintését, az összes és a saját topikok listájának megtekintését, valamint a kijelentkezést.

```
Hello, user!<br>

<%= link_to "Profile", '/users/edit' %><br/>
<%= link_to "All topics", '/topics' %><br/>
<%= link_to "My topics", '/topics' %><br/>
<%= link_to "Logout", '/sessions/destroy' %>
```

A bejelentkezett felhasználó menüjének megvalósítását a 3. ábra mutatja.



3. ábra. A bejelentkezett felhasználó menüje

Kiegészítve a menü `div`-et a töredékekkel ezt kapjuk:

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

Nézzük meg a be nem lépett felhasználó létrehozásának folyamatát! Az előző gyakorlat alkalmával már létrehoztuk a felhasználó modellünk kezdetleges változtatás, így arra már tudhatunk hivatkozni egy Rails formban, amely az MVC tervezési minta szerint szorosan kapcsolódik a nézethez. Hozzuk létre a felhasználó nézetét és fontosabb akcióit a következő paranccsal:

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

```
      create  app/controllers/users_controller.rb
 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
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/users.coffee
      invoke    scss
      create      app/assets/stylesheets/users.scss
```

A parancs futtatásával létrejött az `users` kontroller és a hozzá kapcsolódó nézetek köztük az új felhasználó létrehozását lehetővé tevő `new`, a felhasználói profil szerkesztését megvalósító `edit`, és az elfelejtett jelszó esetén az email címet elkérő `forgotten` nézet. Az tervezői kérdés, hogy az elfelejtett jelszó kezelését a felhasználók kontrollere részének tekintjük, vagy önálló kontrollert hozunk létre számára. A gyakorlaton amellett döntöttünk, hogy az elfelejtett jelszó kerüljön a felhasználók kontrollerébe.

Hozzunk mindjárt létre a regisztrációs nézetet! Legyen egy címsorunk, amely elmondja a felhasználónak, hogy melyik oldalon van. Az esetleges hibaüzeneteknek tartsunk fenn helyet. Ezután egy `fieldset`-ben definiáljuk egy formot, amely ez esetben egy konkrét, létező modellhez van kötve. Ezt a `form_for` Rails helperrel tehetjük meg. Ennek első paramétere egy modell objektum vagy annak neve szimbólum formájában, második paramétere a formhoz kötött akció, amely legyen a `users` kontroller (ezt nem kell leírnunk, mert az új felhasználó létrehozása akció kontrollere ugyanaz) `create` akciója, a harmadik paramétere a HTTP metódus, ami POST. A metódus blokkjának van egy paramétere a `form`, amin kereszül definiáljuk fogjuk az űrlap elemeit. Legyen a négy elem rendre a következő: egy 20 karakter széles, a felhasználó nevére vonatkozó szövegbeviteli mező a hozzá kapcsolódó címkével, egy 30 karakter széles, a felhasználó email címére vonatkozó szövegbeviteli mező a hozzá kapcsolódó címkével, két darab 20 karakter széles jelszóbeviteli mező a hozzájuk kapcsolódó címkével, a két jelszómező eltérő

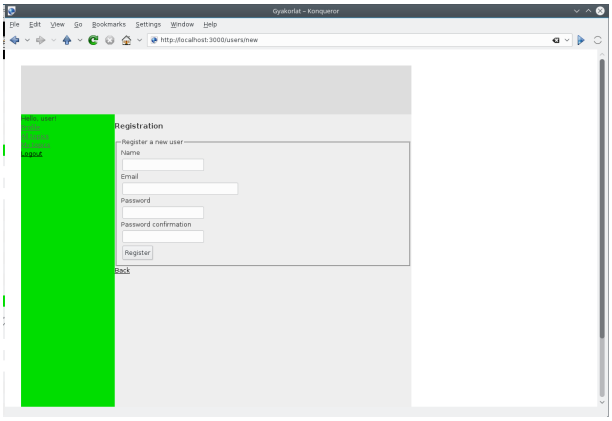
azonosítóval rendelkezzék, az egyik prefixe `_confirmation`-re végződjék.

```
<h3>Registration</h3>
<fieldset>
  <legend>Register a new user</legend>
  <%= form_for @user, url: { action: :create }, method: :
      post do |form| %>
    <div>
      <%= form.label :name %>
      <%= form.text_field :name, size: 20 %>
    </div>
    <div>
      <%= form.label :email %>
      <%= form.text_field :email, size: 30 %>
    </div>
    <div>
      <%= form.label :password %>
      <%= form.password_field :password, size: 20 %>
    </div>
    <div>
      <%= form.label :password_confirmation %>
      <%= form.password_field :password_confirmation, size:
          20 %>
    </div>
    <%= form.submit "Register" %>
  <% end %>
</fieldset>
<%= link_to "Back", :back %>
```

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 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




4. ábra. A regisztráció nézete

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'
  get 'users/edit'
  put 'users/update'
  get 'users/forgotten'
  post 'users/send_forgotten'
```

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. Az email cím 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. Ez még nem létezik, ezért definiáljuk ezt is egyelőre szintén üres törzzsel. Ezen kívül a nézetben a feliratokat kell még átírnunk regisztrációról profil szerkesztésére.

```
<h3>Profile</h3>
<fieldset>
  <legend>Edit user profile</legend>
  <%= form_for @user, url: { action: :update }, method: :
      put do |form| %>
    <div>
```

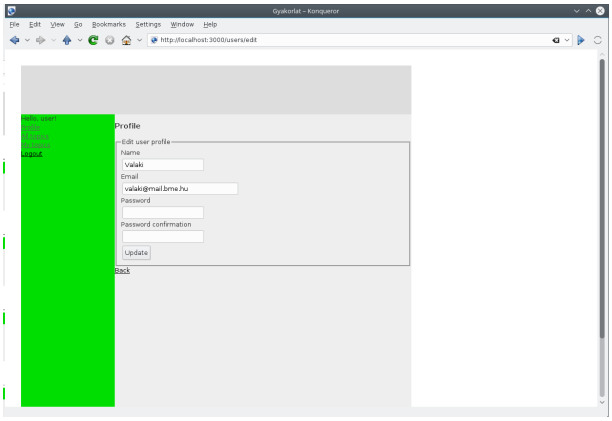
```
      <%= form.label :name %>
      <%= form.text_field :name, size: 20 %>
    </div>
    <div>
      <%= form.label :email %>
      <%= form.text_field :email, size: 30 %>
    </div>
    <div>
      <%= form.label :password %>
      <%= form.password_field :password, size: 20 %>
    </div>
    <div>
      <%= form.label :password_confirmation %>
      <%= form.password_field :password_confirmation, size:
          20 %>
    </div>
    <%= form.submit "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 create
  end

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



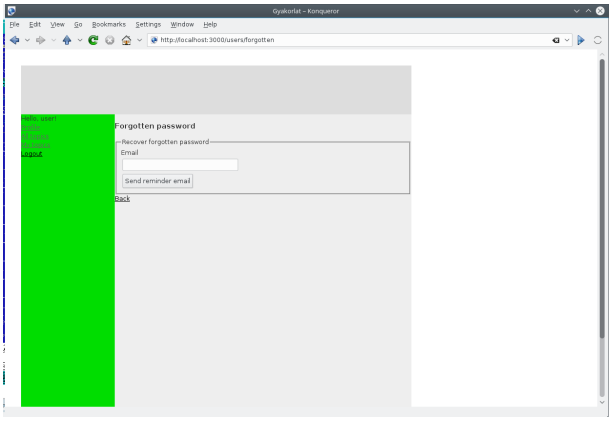

5. ábra. A profiloldal nézete

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

```
<h3>Forgotten password</h3>
<fieldset>
  <legend>Recover forgotten password</legend>
  <%= form_tag url: { controller: :users, action: :
      send_forgotten }, method: :post do %>
    <div>
      <%= label_tag :email %>
      <%= text_field_tag :email, '', size: 30 %>
    </div>
    <%= submit_tag "Send reminder email" %>
  <% end %>
</fieldset>
<%= link_to "Back", :back %>
```

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

Az elfelejtett jelszó nézetét a 6. ábra mutatja.




6. ábra. Az elfelejtett jelszó nézete

Az e-learning portálunk egy elemét, a topikokat az előző gyakorlaton `scaffold`-dal már létrehoztunk. Ezt most kiegészítjük a három további modellel, a feladatokhoz fűzött szöveges kommentek, a feladatokhoz csatolt fájlok és a feladatokhoz linket URL-ek modelljeivel, majd végrehajtuk a migrációt. A komment egy topikhoz tartozik, egy felhasználó hozza létre és egy szövegből áll, valamint létrehozásának időpecsétje automatikusan generálódik. A csatolmány szintén egy topikhoz tartozik és egy felhasználó hozza létre, erről eltároljuk még az feltöltött állomány elérési útját, MIME típusát, méretét és eredeti nevét. A link szintén egy topikhoz tartozik és egy felhasználó hozza létre, itt csak az URL-t kell megjegyeznünk.

```
rails g model Comment topic:references user:references
    comment:text
rails g model Attachment topic:references user:references
    path:string mime:string size:integer name:string
rails g model Link topic:references user:references url:
    string
rails db:migrate
```

Mivel a Rails adatbáziskezelését még nem ismerjük, cseréljük le a topikok kontrollerében az automatikusan generált adatbázisműveleteket tartalmazó kódrészleteket statikus adatokra. Ezt két helyen kell megtennünk, az

`index` akcióban, ahol a `@topics` példányváltozó topik példányokat tartalmazó tömbjét hozzuk létre, illetve a `before_action` helper függvény által kijelölt `set_topic` azonosítójú metódusban, amely lefut mind a `show`, mind az `edit` nézet betöltésekor, itt csak a `@topic` példányváltozóhoz kell egy topik példányt rendelnünk.

```
class TopicsController < ApplicationController
  before_action :set_topic, only: [:show, :edit, :update, :
     destroy]
  def index
    #@topics = Topic.all
    @topics = []
    @topics << Topic.new(title: "Hello", user_id: 1,
       contents: "Hello bello", id: 2)
    @topics << Topic.new(title: "RoR", user_id: 1, contents
       : "Milyen szep a vilag!", id:2)
  end
  private
    def set_topic
      #@topic = Topic.find(params[:id])
      @topic = Topic.new id: 2, title: "Masordszor",
         contents: "Ma kedd van", user_id: 1
    end
end
```

A topikok automatikusan generált nézetei közül az `index` nézetben és a `form` töredékben javítsuk ki a felhasználóról megjelenő adatokat, hogy az ne a felhasználó azonosítóját, hanem annak nevét írja ki. Az `index` nézetben ehhez a táblázat felhasználókra vonatkozó oszlopának értékét módosítjuk:

```
<td><%= topic.user %></td>
```

A `form` töredékben pedig a felhasználóra vonatozó `div`-et>

```
  <div class="field">
    <%= form.label :user %>
    <%= form.text_field :user %>
  </div>
```

Ezek így a felhasználó azonosítója helyett a felhasználó objektum referenciáját jelenítik meg, ami a string konverzió alapértelmezett kimenete. Ehelyett definiáljuk mi magunk a string konverziót a felhasználó modell osztályában.

```
class User < ApplicationRecord
  def to_s
```

```
    self.name
  end
end
```

Az egyes topikok adatlapja nézethez inicializáljuk a megjeneníteni kívánt topikhoz kapcsolódó kommentek, csatolmányok és linkek tömbjeit egy-egy megfelelő típusú modell példánnyal.

```
class TopicsController < ApplicationController
  def show
    @comments = []
    @comments << Comment.new(topic_id:2,user_id:1,comment:"
        Hello", created_at: Time.now)

    @attachments = []
    @attachments << Attachment.new(topic_id:2, user_id: 1,
        path: '/tmp/1.txt', mime: 'text/plain', size: 1,
        name: 'a.txt')

    @links = []
    @links << Link.new(topic_id:2, user_id:1, url: 'http://
        gyakorlat.com')
  end
end
```

A topik adatait megjelenítő `show` nézetet úgy módosítjuk, hogy hozzáadjuk a kommentek listáját és új komment felvételének lehetőségét, a csatolmányok listáját és új csatolmány hozzáadásának lehetőségét, illetve a linkek listáját és új link hozzáadásának lehetőségét. Ezeket egy-egy töredékkel valósítjuk meg, a töredékeknek pedig átadjuk a topikok kontroller `show` akciójában kigyűjtött megfelelő példányváltozó értékét, amelyek a töredékekben így lokális változóként lesznek felhasználhatóak.

```
<h3>Comments</h3>
<%= render 'comments', comments: @comments %>

<h3>Attachments</h3>
<%= render 'attachments', attachments: @attachments %>

<h3>Link</h3>
<%= render 'links', links: @links %>
```

A kommentek töredékben egy `for` ciklussal végigmegyünk az összes, a megjeleníteni kívánt topikhoz tartozó kommenten, és kiírjuk azok adatait. Továbbá felveszünk egy egyszerű formot, ahol új komment adható a topik

kommentlistájához. A form egy szövegdobozt és egy nyomógombot tartalmaz.

```
<% for comment in comments do %>
  <div>
    <%= comment.user.name %> left a comment at <%= l
        comment.created_at %>: <br/>
    <%= comment.comment %>
  </div>
<% end %>

<fieldset>
  <legend>Leave a comment</legend>
  <%= form_tag '/comments/create', method: :post do %>
    <%= text_area_tag 'comment', '', cols: 30, rows: 6 %><
        br/>
    <%= submit_tag "Comment" %>
  <% end %>
</fieldset>
```

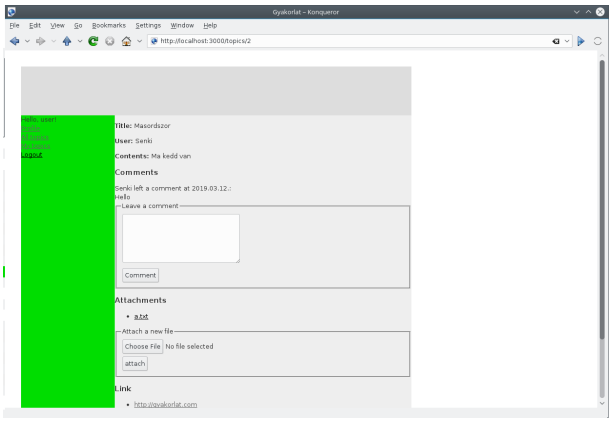
A csatolmányok töredékben egy `for` ciklussal végigmegyünk az összes, a megjeleníteni kívánt topikhoz tartozó csatolmányon, és kiírjuk az egyes csatolmányokra mutató linkeket egy lista elemeiként. Továbbá felveszünk egy egyszerű formot, ahol új csatolmány adható a topik kommentlistájához. A form egy fájlválasztó mezőt és egy nyomógombot tartalmaz.

```
<ul>
  <% for a in attachments do %>
    <li><%= link_to a.name, a.name %></li>
  <% end %>
</ul>

<fieldset>
  <legend>Attach a new file</legend>
  <%= form_tag '/attachments/create', method: :post do %>
    <%= file_field_tag 'upload' %><br/>
    <%= submit_tag "attach" %>
    <% end %>
</fieldset>
```

A linkek töredékben egy `for` ciklussal végigmegyünk az összes, a megjeleníteni kívánt topikhoz tartozó linken, és kiírjuk a linkek URL-jét egy lista elemeiben linkként. Továbbá felveszünk egy egyszerű formot, ahol új link adható a topik kommentlistájához. A form egy szövegbeviteli mezőt és egy nyomógombot tartalmaz.

```
<ul>
  <% for l in links do %>
    <li><%= link_to l.url, l.url %></li>
  <% end %>
</ul>

<fieldset>
  <legend>Add a new link</legend>
  <%= form_tag '/links/create', method: :post do %>
    <%= text_field_tag 'url' %><br/>
    <%= submit_tag "link" %>
    <% end %>
</fieldset>
```

A topik átalakított `show` nézetét a 7. ábra mutatja.




7. ábra. A topik nézet