Aplikacija za dijeljenje informacija o kolegijima

Šajina, Romeo

Sveučilište Jurja Dobrile u Puli
Odjel za informacijsko-komunikacijske tehnologije

Romeo Šajina

APLIKACIJA ZA DIJELJENJE INFORMACIJA O KOLEGIJIMA

Završni rad

Pula, rujan 2017. godine

Sveučilište Jurja Dobrile u Puli
Odjel za informacijsko-komunikacijske tehnologije

Romeo Šajina

APLIKACIJA ZA DIJELJENJE INFORMACIJA O KOLEGIJIMA

Završni rad

JMBAG: 0303054696, redoviti student
Studijski smjer: Sveučilišni preddiplomski studij informatika
Kolegij: Napredne tehnike programiranja
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: doc. dr. sc. Tihomir Orehovački

Pula, rujan 2017. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani, Romeo Šajina, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Romeo Šajina

U Puli, rujan 2017. godine

IZJAVA

o korištenju autorskog djela

Ja, Romeo Šajina, dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom „Aplikacija za dijeljenje informacija o kolegijima“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, rujan 2017. godine

Potpis

Romeo Šajina

Sažetak

U ovom su radu opisane tehnologije i koncepti koji su korišteni pri izradi aplikacije za dijeljenje informacija o kolegijima. Aplikacija je izrađena kako bi studentima omogućila brže i lakše dijeljenje informacija za pojedini kolegij, uz mnogo drugih funkcionalnosti. Koncept dizajna strukture, koji aplikacija slijedi, jest realizacija aplikacije korištenjem RESTful servisa i klijentskog sloja aplikacije. Klijentski sloj aplikacije sa servisom komunicira kod dohvaćanja i ažuriranja podataka. Takav dizajn ubrzava rad aplikacije i poboljšava cjelokupni korisnički doživljaj. Bitno je napomenuti da korisnici aplikacije imaju potpunu kontrolu nad sadržajem. Dakle, korisnik može dodavati, uređivati ili izbrisati bilo koji podatak na aplikaciji. Kako bi se korisnika što više zainteresiralo za korištenje aplikacije, u nju su implementirani elementi gejmifikacije, odnosno ukomponirani su neki koncepti koji se uobičajeno koriste u video igrama.

***Ključne riječi**: web aplikacija, adik, dijeljenje informacija o kolegijima, AngularJS, PHP, RESTful servisi, HTML, CSS, Bootstrap, jQuery, JSON, MySQL, SQL, gejmifikacija.*

Abstract

This paper describes technologies and concepts used at developing application for sharing information of courses. Application is developed with intention of helping students in sharing information for each course quicker and easier, and is filled with different functionalities. Concept of designing structure that application follows is realization of application using RESTful service and client application layer. Client application layer communicates with service when fetching or updating data. Such a design accelerates application performance and improves the overall user experience. It is important to note that application users have complete control over content,

meaning that a user can add, edit, or delete any data in the application. In order to make the user more interested in using the application, gamification elements are implemented, respectively incorporated some commonly used concepts in video games.

Keywords: *web application, adik, sharing information, AngularJS, PHP, RESTful services, HTML, CSS, Bootstrap, jQuery, JSON, MySQL, SQL, gamification.*

Sadržaj

1. Uvod

Broj web aplikacija se je u posljednjem desetljeću jako povećao. Mogućnosti korištenja aplikacije neovisno o platformi na kojoj radi računalo kojim pristupamo aplikaciji, mogućnosti korištenja aplikacije bez instaliranja iste, pružanje bogatog korisničkog iskustva samo su neke od prednosti kod razvijanja web aplikacije. Kako se povećava broj web aplikacija, povećava se i broj web tehnologija koje se razvijaju u svrhu bržeg razvoja web aplikacija, pružanja boljeg korisničkog iskustva, povećanja brzine i sl. Između mnoštva tehnologija koje omogućavaju brz i jednostavan razvoj web aplikacije, za izradu aplikacije *adik* odabrane su tehnologije koje se najviše spominju u posljednjih nekoliko godina.

Aplikacija *adik* je izrađena u svrhu lakšeg dijeljenja informacija o kolegijima između studenata. Motivacija za izradu aplikacije je većim dijelom došla zbog dosadašnjeg loše strukturiranog načina dijeljenja informacija, datoteka i poveznica vezano uz pojedine kolegije. Dosadašnji načini dijeljenja takvih informacija provodili su se kroz razne grupe na društvenoj mreži Facebook. Tamo bi se objavljivale razne informacije, datoteke, poveznice i pitanja uz pokoju ponudu za sezonski posao. Taj princip djeluje nedovoljno funkcionalnim, prije svega, zbog loše strukture objavljenog sadržaja, tj. na jednom su mjestu sve objavljene informacije bez mogućnosti filtriranja sadržaja za određeni kolegij.

Kod izrade aplikacije svi su dosadašnji nedostaci uzeti u obzir kako bi aplikacija omogućila lakše, brže i razumljivije dijeljenje informacija za pojedine kolegije. Uz razne mogućnosti koje aplikacija nudi, bitno je napomenuti da pomoću iste studenti imaju potpunu kontrolu nad svim sadržajem, odnosno u mogućnosti su dodavati, uređivati ili izbrisati bilo koji podatak u aplikaciji bez ograničenja.

Kao koncept za izradu aplikacije odabran je moderniji pristup koji je složeniji od tradicionalnog koncepta, ali pruža mnogo prednosti kao što su: bolje korisničko iskustvo, smanjenje opterećenja na poslužitelju, ubrzavanje rada cijele aplikacije i sl. Glavna razlika između dvaju pristupa jest u mjestu izvođenja pojedinih operacija. Navedenu razliku je najlakše objasniti kroz primjer prezentacije podataka. Tradicionalni koncept podrazumijeva prezentaciju podataka na poslužitelju i slanje

kreiranog sadržaja klijentu. Moderniji koncept, koji je implementiran u aplikaciji, podrazumijeva aplikaciju na klijentu koja će od poslužitelja zatražiti samo podatke, dok će prezentaciju podataka izvršiti na samom klijentu.

U sljedećem će poglavlju biti prikazani osnovni zahtjevi na sustav na kojima se je temeljila izrada aplikacije.

U trećem će se poglavlju spomenuti i pobliže objasniti sve tehnologije koje su korištene u svrhu razvoja web aplikacije. One će biti podijeljene u dvije podskupine kako bi se jasno vidjelo koje su se tehnologije koristile na poslužiteljskoj strani, a koje na klijentskoj.

U četvrtom će poglavlju biti prikazane slične postojeće aplikacije na kojima se je temeljilo prikupljanje osnovnih zahtjeva na sustav i gdje su se potencijalno uvidjeli njihovi nedostaci, koji su se u izradi aplikacije ispravili.

U petom će poglavlju biti prikazano programsko rješenje odnosno kako su sve navedene tehnologije ukomponirane. Objašnjenje komuniciranja tehnologija je ponovno podijeljeno na tehnologije na poslužiteljskoj i klijentskoj strani. Kod kompliciranijih slučajeva komuniciranja tih tehnologija bit će grafički prikazan način izvođenja pojedinog segmenta.

U šestom će poglavlju biti pojašnjene korisničke mogućnosti u korištenju aplikacije, te potencijalne administrativne mogućnosti koje još nisu implementirane.

Naposljetku će biti objašnjen i socijalni aspekt koji je implementiran u aplikaciju.

2. Osnovni zahtjevi na sustav

Prije samog početka opisivanja strukture i korištenih tehnologija, potrebno je navesti osnovne zahtjeve na sustav na kojima se je izrada aplikacije bazirala. Bit će navedeni samo funkcionalni zahtjevi dok će nefunkcionalni zahtjevi biti objašnjeni kroz rad u obliku mjera koje su poduzete kako bi se poboljšalo cjelokupno korisničko iskustvo.

Osnovni funkcionalni zahtjevi na sustav su:

1. uređivanje smjerova,
2. uređivanje kolegija za pojedini smjer,
3. pretraživanje kolegija i nastavnika,
4. uređivanje nastavnika,
5. povezivanje nastavnika s njegovom službenom stranicom na FET-u,
6. komentiranje, postavljanje pitanja, uređivanje datoteka i poveznica,
7. mogućnost označavanja pojedinog kolegija kao *omiljeni*.

3. Korištene tehnologije za razvoj aplikacije

Za realiziranje aplikacije korištene su razne tehnologije koje omogućavaju brzu, interaktivnu i responzivnu okolinu, te tehnologije koje se brinu o pohranjivanju podataka. U sljedećih nekoliko poglavlja će one biti prikazane i pojašnjene.

3.1. Poslužiteljska strana

Na strani poslužitelja (eng. *Server side* ili *Backend*) korištene su dvije najpoznatije tehnologije kod programiranja dinamičkih web aplikacija, a to su: skriptni jezik PHP i MySQL baza podataka, koji će biti pojašnjeni kroz sljedeća dva poglavlja.

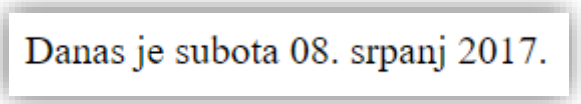
3.1.1. PHP

PHP (rekurzivni akronim za engl. *Hypertext Preprocessor*) je programski jezik koji služi za generiranje dinamičkih web stranica i dizajniran je u svrhu jednostavnog ugrađivanja u HTML stranice, iako se ne treba tako koristiti. Većina PHP stranica ima izmiješan PHP i HTML kod. PHP je najpoznatiji i najkorišteniji programski jezik kod programiranja dinamičkih web stranica zbog svoje sintakse, koja je slična C-u, brzine izvođenja, raznih mogućnosti i cijene (PHP je otvorenog koda). Izvođenje PHP stranice radi tako da kada poslužitelj pročita PHP stranicu, on traži dvije oznake, preko kojih će znati čita li stranicu kao PHP ili HTML, a to su početni i završni znakovi PHP-a: *<?php* i *?>* (PHP, n.d.).

Prethodno prikazani znakovi su već zadani, međutim oni se mogu konfigurirati na poslužitelju tako da neki drugačiji znakovi označavaju PHP kod (npr. <% %>). Osim brojnih mogućnosti koje dolaze s verzijom PHP-a kao što su: mogućnost komuniciranja s različitim bazama podataka, rad s datotekama i sl., postoji mnogo knjižnica koje su razvili programeri za različite svrhe, čime ga čine još moćnijim programskim jezikom. Glavna specifičnost PHP-a je u definiranju varijabli koje moraju imati prefiks *$* čime će PHP *interpreter* znati da se radi o varijabli, dok će se imena bez prefiksa *$* smatrati kao funkcije čime će se ubrzati izvođenje skripte.

```php
<?php

  setlocale(LC_ALL, 'Croatian');

  $time = mktime(0, 0, 0, date("m"), date("d"), date("y"));

  echo "Danas je " . strftime("%A %d. %B %Y.", $time);

?>
```

Slika 1. Prikaz ispisa datuma prema postavljenoj lokali u PHP-u



Slika 2. Rezultat ispisa datuma

Slika 1. prikazuje kod u PHP-u koji ispisuje datum oblikovan prema postavljenoj lokali. Prvom naredbom *setLocale* postavlja se lokala na *Croatian,* odnosno nakon postavljanja lokale, sve funkcije koje koriste lokalu za oblikovanje podatka će podatak oblikovati prema standardu definiranom za tu lokalu. U drugoj funkciji *mktime* se kreira trenutni datum u obliku broja (npr. za datum 8.7.2017. je on 1499464800). Naposljetku, kod ispisa datum se formatira prema postavljenoj lokali koristeći funkciju *strftime* koja prema proslijeđenom formatu oblikuje proslijeđeni datum.

PHP je kod izrade aplikacije korišten za realizaciju RESTful (kratica za engl. *representional state transfer*) servisa s kojim će aplikacija na klijentu komunicirati. RESTful servisi omogućavaju interoperabilnost između različitih programskih jezika i aplikacija, a na određeni zahtjev odgovaraju u standardiziranom formatu kao što su: XML, JSON i sl.

REST je stil softverske arhitekture koji se može pratiti tijekom dizajniranja softvera. REST je idealan stil dizajniranja za aplikacije koje se baziraju na web servisima. Principe vezane uz REST je prvotno opisao Roy Fielding u svojoj disertaciji.

Slijede ukratko ključni principi REST-a:

- pružiti svakom resursu jedinstveni ID, npr. URI,
- međusobno povezati resurse, ostvarujući vezu između resursa,
- koristiti standardne metode (HTTP, medijske tipove, XML),
- resurs može imati više reprezentacija koji odražavaju drugačija stanja aplikacije,
- komunikacija bi trebala biti bez stanja koristeći HTTP.

Često se REST stil dizajniranja odnosi kao stil dizajniranja web-a. Većina današnjih web aplikacija demonstriraju karakteristike REST stila dizajniranja prilikom izgradnje web servisa na web-u (Abeysinghe, 2008).

3.1.2. MySQL

MySQL je najpoznatiji besplatan sustav otvorenog koda za upravljanje bazom podataka. Razlog popularnosti MySQL-a je: cijena, brzina obrade naredbi i lakoća izvođenja u zahtijevanju poslužiteljskih resursa. MySQL je točnije sustav za upravljanje relacijskim bazama podataka koji radi preko SQL-a (MySQL, 2017).

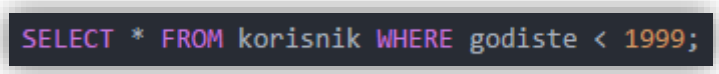
SQL (kratica za engl. *Structured Query Language*) je jezik pomoću kojeg se izvršavaju naredbe nad podacima u bazama podataka. On omogućuje operacije kao što su spremanje, ažuriranje i dohvaćanje podataka iz baze podataka.

Primjer SQL upita može biti dohvaćanje svih korisnika koji su stariji od 18 godina, no prije prikaza upita potrebno je definirati tablicu nad kojom će se upit izvršiti. Tablica *korisnik* će sadržavati informacije o imenu, prezimenu i godištu korisnika.

```
CREATE TABLE korisnik(
    id INTEGER NOT NULL AUTO_INCREMENT,
    ime VARCHAR(20) NOT NULL,
    prezime VARCHAR(20) NOT NULL,
    godiste INTEGER NOT NULL,
    CONSTRAINT korisnik_id_pk PRIMARY KEY(id)
);
```

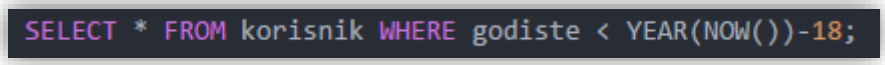
Slika 3. Prikaz kreiranja tablice

Tablica korisnik sadrži i kolonu *id* koja će jednoznačno identificirati korisnika. Ona je ujedno i primarni ključ koji se automatski popunjava i povećava.



Slika 4. Prikaz SQL upita koji dohvaća punoljetne korisnike za 2017. godinu

SQL upit na slici 4. dohvaća sve punoljetne korisnike za 2017. godinu. Upit se izvršava i rezultati odgovaraju zahtjevu, međutim problem se javlja kad se godina promijeni u 2018. Dakle, programer bi onda trebao promijeniti uvjet u upitu, što nije najpraktičnije, ali i za to postoji rješenje.



Slika 5. Prikaz SQL upita koji dohvaća punoljetne korisnike

Slika 5. prikazuje SQL upit koji također dohvaća sve punoljetne korisnike, ali ovom slučaju godina nije kodirana u upitu već se dinamički izračunava pomoću funkcije *YEAR,* koja iz datuma izvlači godinu, i funkcije *NOW* koja vraća trenutni datum i vrijeme. Primjer je samo uvod u mogućnosti koje nudi MySQL baza podataka.

3.2. Klijentska strana

Postoje razne tehnologije koje se mogu koristiti u izvođenju aplikacije na klijentskoj strani (eng. *Client side* ili *Frontend*). Kod web aplikacije standardno se koriste: HTML za prikaz sadržaja i CSS za oblikovanje prikazanog sadržaja na klijentu, te su one korištene i u ovoj aplikaciji. Međutim kako bi se postigla responzivnost, interaktivnost i veća brzina aplikacije korištene su i druge (novije) tehnologije kao što su Bootstrap, jQuery i AngularJS. Klijentske tehnologije koje se koriste u aplikaciji biti će pojašnjene u nekoliko sljedećih poglavlja.

3.2.1. HTML

HTML (kratica za engl. *HyperText Markup Language*) je prezentacijski jezik koji omogućuje izradu prezentacijskog sloja aplikacije. S obzirom na to da je HTML samo prezentacijski jezik, on ne može raditi nikakve operacije s podacima što ga čini vrlo jednostavnim, razumljivim i brzo savladivim. Dakle, on služi samo za opis dokumenta, a to se vrši kroz predefinirane elemente koji se označavaju sintaksom: <element atribut="vrijednost"> Sadržaj </element>. Trenutna verzija HTML-a je 5 (tzv. HTML5) koja donosi nove elemente kao što su <nav> (element za oblikovanje navigacije), <footer> (element koji označava podnožje stranice), <canvas> (element koji omogućuje crtanje) i mnogi drugi. HTML5 već podržava većina web preglednika (HTML.com, n.d.).

3.2.2. CSS

CSS (kratica za engl. *Cascading Style Sheets*) je stilistički jezik koji se koristi kod oblikovanja HTML elemenata. CSS se s HTML elementom kojeg oblikuje najčešće povezuje preko HTML atributa *class,* u kojeg se kao vrijednost stavlja ime definirane klase u CSS-u. CSS stilovi se mogu definirati u zasebnoj datoteci, što daje mnogo prednosti, a neke od njih su: promjena stila na jednom mjestu odrazit će se na sve elemente koji koriste taj stil, lakše održavanje aplikacije, konzistentniji izgled aplikacije, web preglednik učita jednom CSS datoteku i kasnije je koristi za ostale stranice čime se smanjuje opterećenje na poslužitelja itd.

Sintaksa CSS-a jest:

selektor{ svojstvo : vrijednost; } .

Npr. div.group button{ width : 50%; } što znači da će svaki gumb, koji ima nadređeni element *div* s klasom *group* kao atribut, biti širok 50% dostupne širine.

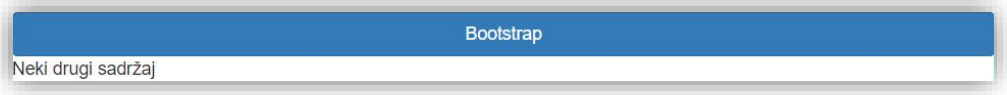
3.2.3. Bootstrap

Bootstrap je besplatni web okvir otvorenog koda za dizajniranje web stranica i web aplikacija. Sastoji se od HTML i CSS baziranih predložaka za različite komponente poput formi, gumbova, navigacije i ostalih komponenti na korisničkom sučelju. Bootstrap su razvili Mark Otto i Jacob Thornton u tvrtki Twitter kao okvir koji će osiguravati konzistentnost kroz web aplikaciju (Bootstrap, n.d.).

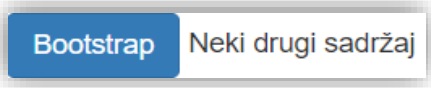
Bootstrap, dakle, dolazi s unaprijed definiranim načinom ponašanja web aplikacije, koje radi jednako na svim web preglednicima i automatski se prilagođava uređaju koji koristi aplikaciju (računalo, mobitel ili tablet). Prednost korištenja Bootstrap-a je očita, a to je brzi razvoj responzivne aplikacije, dok je nedostatak sličnost drugim aplikacijama koje također koriste Bootstrap.

```
<button class="btn btn-primary col-sm-12">Bootstrap</button>
Neki drugi sadržaj
```

Slika 6. Prikaz definiranja izgleda i ponašanja gumba koristeći Bootstrap



Slika 7. Prikaz ponašanja gumba pri širini ekrana većoj od 768px



Slika 8. Prikaz ponašanja gumba pri širini ekrana manjoj od 768px

Prethodni primjer prikazuje kako se gumb ponaša na različitim širinama ekrana, a to ponašanje, kao i sam izgled gumba je definirano koristeći Bootstrap CSS klase.

3.2.4. jQuery

Iako je sam JavaScript vrlo moćan s mnogo uključenih funkcija, postići interoperabilnost određenog koda između različitih web preglednika, zbog različitog načina rada pojedinih web preglednika, je gotovo nemoguće.

Štoviše, kao posljedica različitih ratova između web preglednika tijekom godina, frustracijske i dosadne nekompatibilnosti web preglednika dođu i otiđu u različito vrijeme na različitim platformama i programima. Kao rezultat, osiguravajući da će web stranice izgledati isto na svim uređajima može ponekad biti postignuto jedino kroz naporni JavaScript koji računa na sve razlike između raspona web preglednika i verzija objavljenih tijekom posljednjih godina. Ukratko – noćna mora (Nixon, 2015, p. 499).

Dakle, prije jQuery knjižnice programer je morao znati koje će web preglednike koristiti korisnik aplikacije i testirati aplikaciju na svim tim web preglednicima. To je jedan od glavnih razloga zašto je osmišljen jQuery. jQuery rješava problem različitog ponašanja određenog koda u različitim web preglednicima, skraćuje sintaksu koda, daje mogućnost ulančavanja funkcija, omogućuje odjeljivanje JavaScript-a od HTML-a itd. Zbog svojih prednosti jQuery je najkorištenija JavaScript knjižnica. Prema istraživanjima W3Tech-a (2017) jQuery koristi preko 70% web stranica. Razlika između običnog JavaScript-a i jQuery-a prikazana je i sljedećim primjerom.

```
<button type="button" id="javascript-button"
        onclick="javascriptButtonClicked()">JavaScript</button>

<script type="text/javascript">

  function javascriptButtonClicked() {
    var button = document.getElementById('javascript-button');
    button.innerHTML = "Kliknuto";
  }

</script>
```

Slika 9. Prikaz definiranja obrade klika pomoću JavaScript-a

```
<button type="button" id="jquery-button">jQuery</button>

<script type="text/javascript">

  $('#jquery-button').on('click', function() {
    $(this).html("Kliknuto");
  });

</script>
```

Slika 10. Prikaz definiranja obrade klika pomoću jQuery-a

3.2.5. AngularJS

AngularJS je aplikacijski okvir baziran na JavaScript-u koji se izvodi na klijentskoj strani. Aplikacijski okvir omogućuje jednostavnu implementaciju MVC[1] arhitekture aplikacije na klijentu. AngularJS radi tako da prvo pročita HTML stranicu, pronalazi sve direktive koje su definirane bilo kao atribut HTML elementa ili kao prošireni HTML element. AngularJS zatim interpretira prepoznate direktive te povezuje podatke s *model* dijelom koji je zapravo realiziran kroz standardne JavaScript varijable.

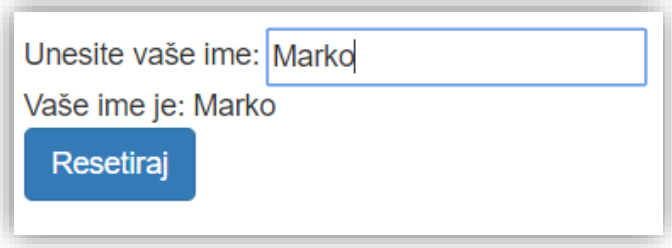
Knjižnice poput jQuery-a omogućavaju djelomično osvježavanje stranice s novim sadržajem mijenjajući unutrašnji sadržaj elementa. To sve rado dosta dobro, ali kada želite dodati svježe podatke u UI ili promijeniti podatke ovisno o korisničkom unosu morate napraviti podosta ne trivijalnog posla da bi osigurali pravilno stanje podataka, oboje u UI i JavaScript svojstvima. Ali što ako bismo mogli napraviti sav taj posao bez pisanja koda? Što ako bi mogli deklarirati koji dio UI-a se mapira na koje JavaScript svojstvo te da se oni sinkroniziraju automatski? Takav stil programiranja se zove povezivanje podataka (engl. Data Binding). Uključili smo ga u Angular jer radi dobro s MVC arhitekturom kako bi se eliminiralo dodatno kodiranje kod pisanja *view* i *model* dijela. Većina posla kod premještanja podataka iz jednog drugom se događa automatski (Green & Seshadri, 2013, p. 16).

[1] MVC : Model - View - Controller

Kao što je prethodno spomenuto, glavna značajka AngularJS-a jest tako zvano dvostrano povezivanje podataka (engl. two-way data binding) koje nakon promjene podataka ažurira *view i model* dio bez potrebe pisanja funkcija koje će se brinuti o takvom ažuriranju. Dvostrano povezivanje podataka najlakše je objasniti na primjeru pa će ono biti prikazano kroz sljedeći primjer.

```
<div ng-controller="TwoWayBindingCtrl">

  Unesite vaše ime: <input type="text" ng-model="user.name"> <br>
  Vaše ime je: {{user.name}} <br>

  <button ng-click="reset()" class="btn btn-primary">Resetiraj</button>

</div>

<script type="text/javascript">

angular.module('adik', [])
.controller('TwoWayBindingCtrl', ['$scope', function($scope) {

    $scope.user = {'name' : ''};

    $scope.reset = function(){
      $scope.user.name = '';
    }
}]);

</script>
```

Slika 11. Prikaz dvostranog povezivanja podatka u AngularJS-u



Slika 12. Prikaz automatskog ažuriranja *model* dijela nakon što korisnik mijenja *view* dio

Slika 13. Prikaz automatskog ažuriranja *view* dijela nakon što se ažurira *model* dio

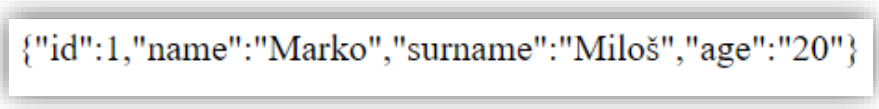
Na slici 11. prikazan je programski dio primjera u kojem možemo vidjeti da se na komponentu *input* veže atribut *name* objekta *user*, što znači da će se, nakon što korisnik promjeni tekst unutar polja, automatski promijeniti atribut *name* na svim mjestima gdje se koristi, bilo na *view* ili *model* dijelu. Kada korisnik unosi nešto u polje on time mijenja *view* dio, a AngularJS se brine da se ažurira i *model* dio. Pritiskom gumba Resetiraj javlja se obrnuta situacija pri čemu se direktno mijenja *model* dio, a AngularJS se brine da se promjena automatski primijeni i na *view* dijelu. Nakon pritiska gumba atribut se prazni, korisniku se briše uneseni sadržaj u polju, te ispis atributa ne ispisuje ništa. Kako je prethodno navedeno, automatsku sinkronizaciju *model* i *view* dijela vrši AngularJS i samim time programeru smanjuje opseg posla, ali i brige o kreiranju prisluškivača promjena na korisničkom sučelju kako bi automatski ažurirali varijablu na koju se veže to polje koje korisnik ažurira.

3.2.6. JSON

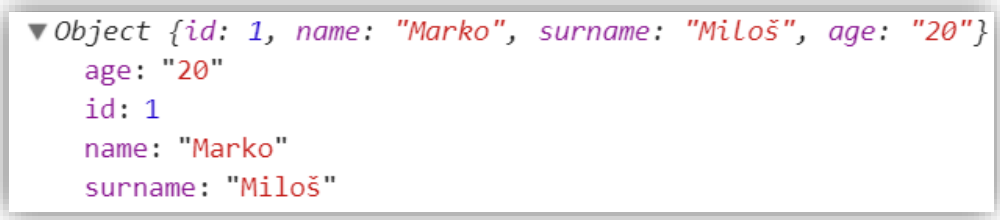
JSON (kratica za engl. *JavaScript Object Notation*) je format podataka koji potječe od programskog jezika JavaScript. To čini JSON podskup jezika JavaScript. Kao podskup, JSON ne posjeduje nikakve dodatne značajke koje jezik JavaScript već ne posjeduje. Iako je JSON podskup programskog jezika, on sam po sebi nije programski jezik već je zapravo format za razmjenu podataka, što znači da se može koristiti pri komunikaciji različitih tehnologija. Razmjena podataka se može dogoditi između preglednika i poslužitelja, ali i između poslužitelja i poslužitelja. Naravno, to nisu sve situacije gdje je moguća razmjena podatak u JSON formatu (Smith, 2015, p. 55).

```
<script type="text/javascript">

  var obj = {
    'id' : 1,
    'name' : 'Marko',
    'surname' : 'Miloš',
    'age' : '20'
  }

  var json = JSON.stringify(obj);

  document.write(json);

  var objFromJson = JSON.parse(json);

  console.log(objFromJson);

</script>
```

Slika 14. Prikaz pretvorbe objekta u JSON zapis i obrnuto



Slika 15. Prikaz objekta u JSON zapisu



Slika 16. Prikaz objekta koji je oblikovan iz JSON zapisa

Na slici 14. prikazano je kreiranje objekta u JavaScript-u, koji se zatim funkcijom *JSON.stringify* oblikuje u zapis JSON formatom. Dakle, u varijabli *json* je spremljen zapis u JSON formatu koji se može smatrati „običnim“ tekstom, što prikazuje i slika 15. Taj zapis se dalje može koristiti na različite načine: može se poslati poslužitelju na obradu (npr. spremi podatak u bazu podataka) čime poslužitelj može isto odgovoriti u obliku JSON formata (ali i ne mora), može ga se spremiti neki klijentski memorijski prostor (npr. spremanje u *Cookie*) itd.

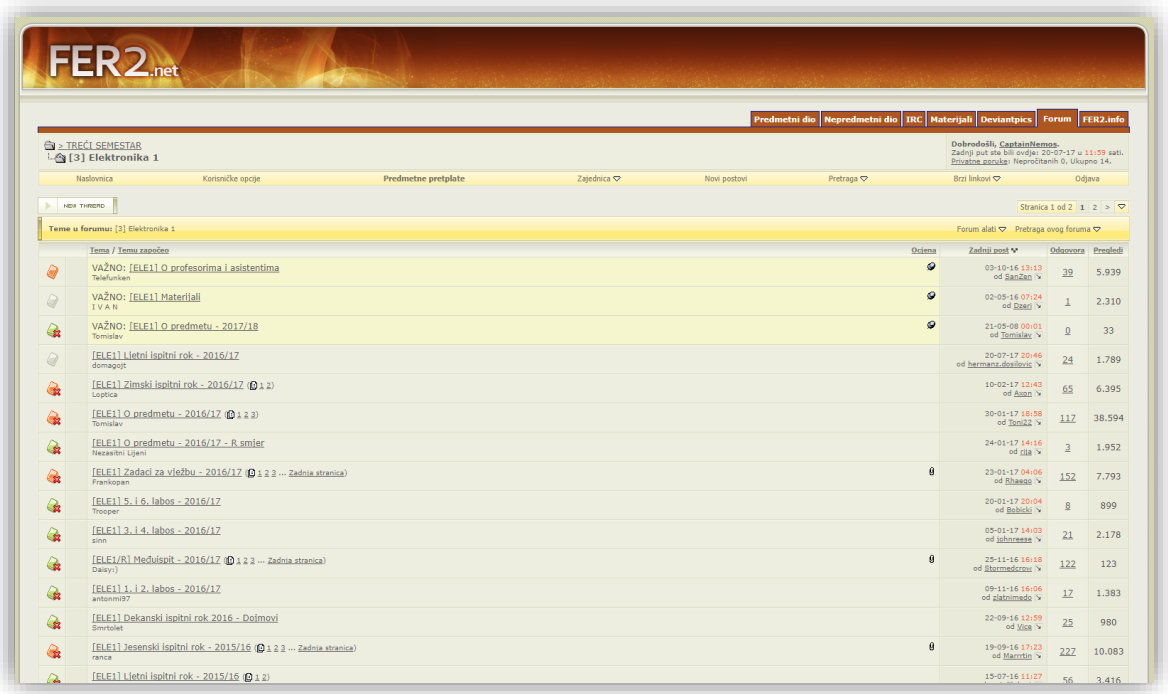
4. Postojeće aplikacije

U ovom će poglavlju biti prikazane postojeće aplikacije slične izrađenoj aplikaciji kako bi se najbolje uvidjela dosadašnja organizacijska problematika kod dijeljenja informacija o kolegijima, a koje su korištene i kao ogledni primjer kod izrade aplikacije, prije svega, kako bi se što bolje utvrdili nedostaci i eventualno uočile funkcionalnosti koje nisu bile definirane u korisničkim zahtjevima.

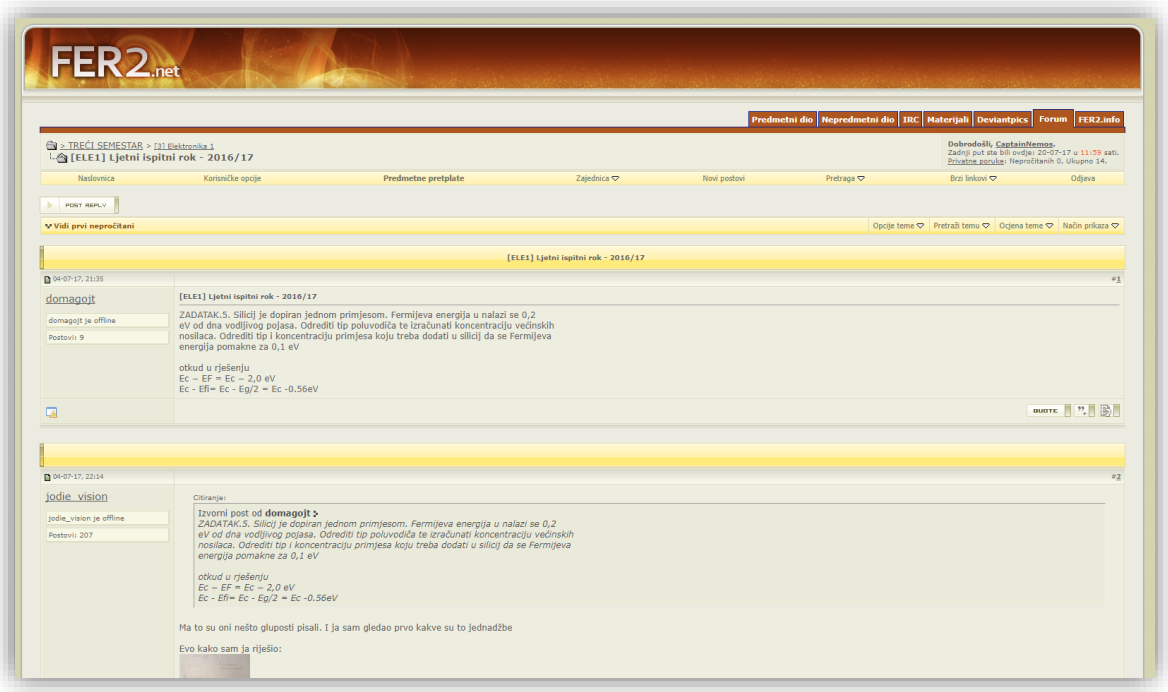
4.1. FER2.net

Aplikacija se zove FER2.net i izradili su je studenti Fakulteta elektrotehnike i računarstva po bolonjskom (FER2) programu. Na aplikaciju se mogu registrirati samo studenti koji se nalaze na popisu FER2 studenata uz godišnju novčanu pretplatu (FER2.net, 2017).

Aplikacija je izrađena u obliku foruma gdje studenti mogu postavljati nova pitanja, odgovarati na njih, postavljati razne materijale i slično. Kod postavljanja novog sadržaja odabire se predmet na koji se sadržaj odnosi.



Slika 17. Prikaz tema u aplikaciji FER2.net (FER2.net, 2017)



Slika 18. Prikaz pitanja i odgovora u aplikaciji FER2.net (FER2.net, 2017)

Aplikacija FER2.net pruža mnogo funkcionalnosti kao što su:

- objavljivanje materijala,
- objavljivanje postova,
- prilaganje slika u postu,
- mogućnost privatnog komuniciranja s ostalim studentima,
- pretraživanje prema ključnim riječima,
- prikaz najnovijih tema,
- pregled prijatelja koji su trenutno prijavljeni u aplikaciju,
- pregled administratora koji su trenutno prijavljeni u aplikaciju,
- administratorske mogućnosti i sl.

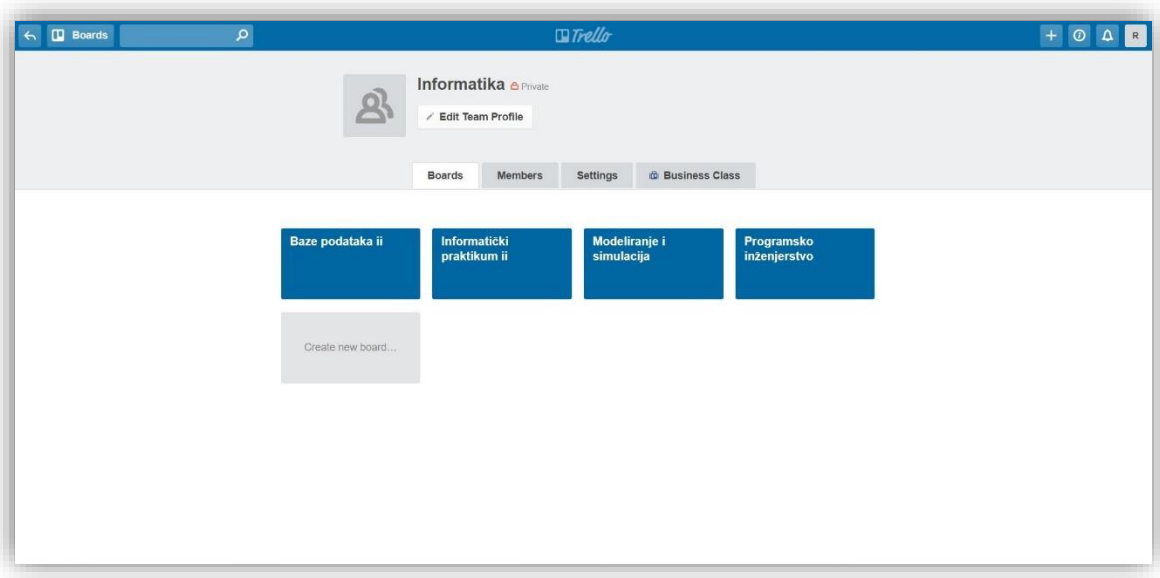
Međutim, glavni je nedostatak već prethodno spomenuta novčana pretplata bez koje student ne može pristupiti predmetima, što je razumljivo, s obzirom na to da određeni studenti u ulozi administratora moraju provoditi nekolicinu vremena prateći radnje studenata unutar aplikacije te odgovarati na različita pitanja, točnije zahtjeve. U aplikaciji *adik* je taj problem riješen tako da je studentu omogućeno mijenjanje svog sadržaja na aplikaciji, ili preciznije, sadržaj se daje studentu na povjerenje, a samim time jest potreba za administratorima manja, pri čemu za korištenje aplikacije nije potrebno plaćati pretplatu.

4.2. Trello

Trello je kolaboracijski alat koji organizira projekte u ploče. Ukratko, Trello korisniku govori na čemu se radi, tko radi na tome i gdje je nešto u procesu. Trello ploča je lista s listama, popunjenih s karticama koje koriste pojedinci i njihov tim. Trello pruža mogućnost kreiranja timova i postavljanja ploča tako da one pripadaju određenim timovima. Kao i ploče, timovi mogu imati članove (administratore i normalne članove). Nema granice u broju članova koji mogu biti u timu ili u broju timova koji se vežu na jedan korisnički račun (Trello, 2017).

Trello nije specijalizirana aplikacija za dijeljenje informacija o kolegijima. Međutim, zbog svojih brojnih mogućnosti i jednostavnosti, Trello može poslužiti kao dobro rješenje tog problema.

Prije samog početka rada s Trello-m potrebno je postaviti strukturu smjerova i kolegija. Za svaki smjer potrebno je kreirati jedan tim gdje bi se uključili svi studenti koji pohađaju taj smjer. Unutar pojedinog tima odnosno smjera, dodaju se ploče koje označavaju određeni kolegij.

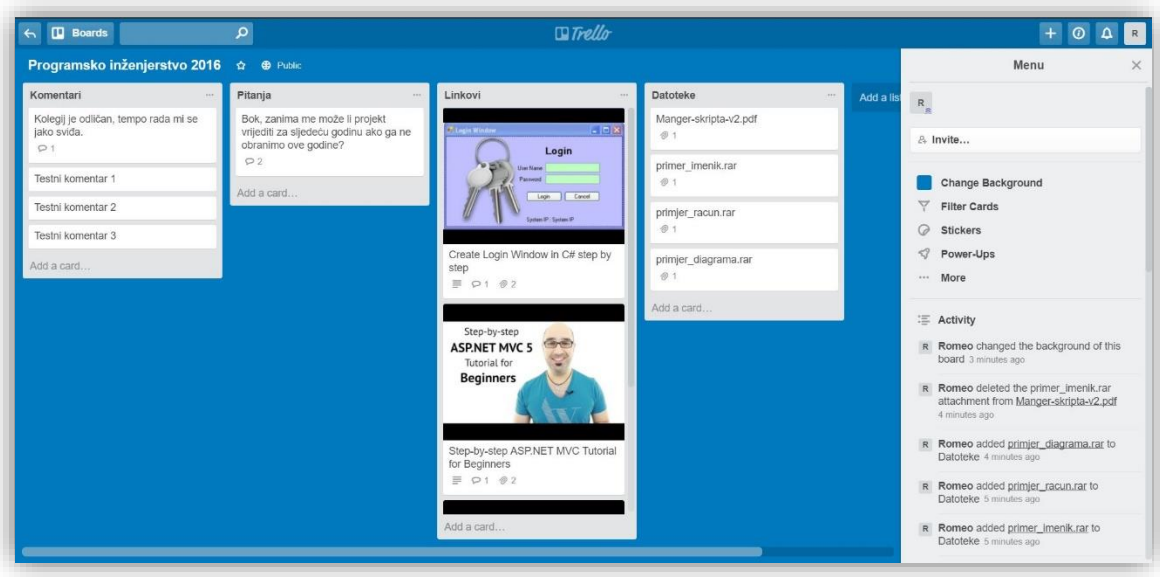


Slika 19. Prikaz ploča u timu (Trello, 2017)

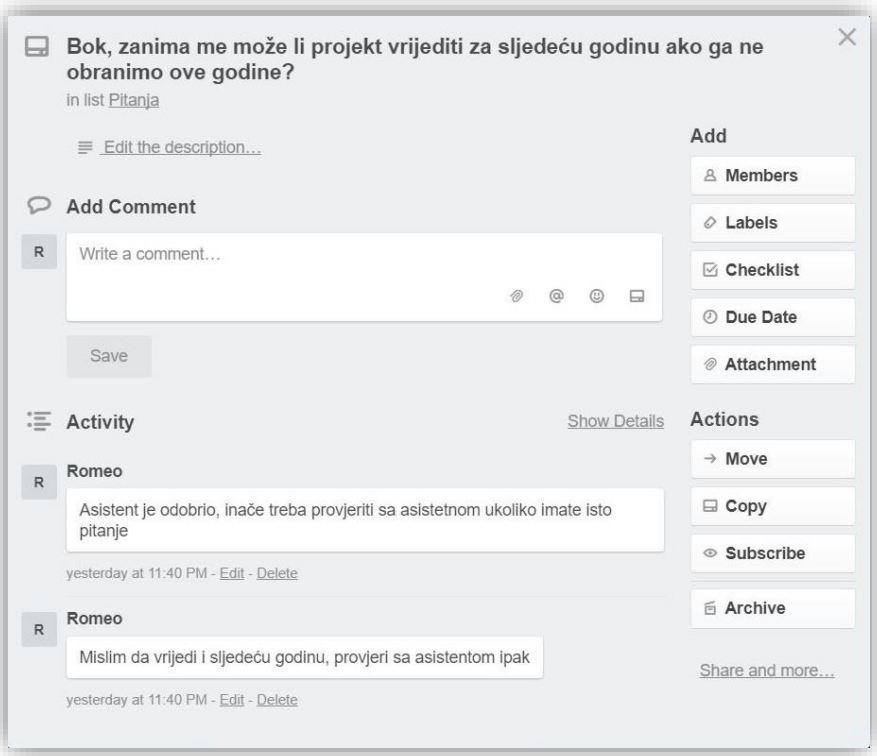
Svaki kolegij dalje ima liste za:

- komentare,
- pitanja,
- poveznice,
- datoteke i sl.

U svaku listu studenti, uključeni u tim koji posjeduje ploču, mogu dodavati kartice sa sadržajem. Svaki dodani sadržaj se može pregledavati, komentirati i uređivati, čime je većina zahtjeva aplikacije za dijeljenje informacija o kolegijima zadovoljena.



Slika 20. Prikaz ploče za kolegij (Trello, 2017)



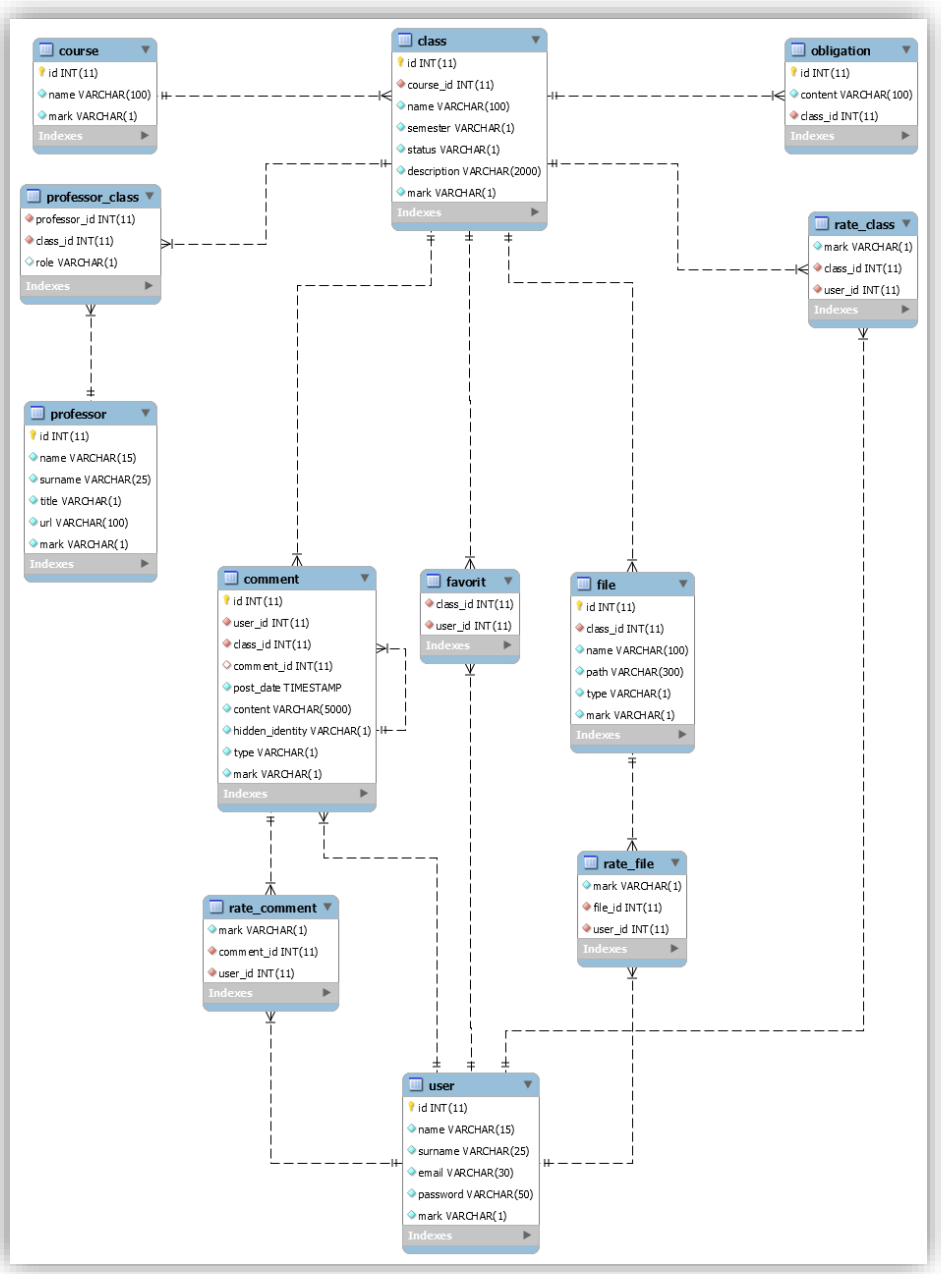
Slika 21. Prikaz komentiranja (Trello, 2017)

Iako Trello pruža mnogo mogućnosti kod realizacije takvog načina dijeljenja informacija o kolegijima, potrebno je napomenuti i njegove nedostatke u takvom kontekstu. Njegov najveći nedostatak jest prikaz sadržaja u obliku kartice. Kartica simbolizira kratkotrajnu informaciju ili zadatak, što u kontekstu aplikacije za dijeljenje informacija o kolegijima nije poželjno jer su informacije puno dužeg vijeka. Sljedeći nedostatak bio bi veličina kartica zbog koje bi, uslijed povećanja sadržaja, bilo otežano snalaženje kroz objavljen sadržaj. Posljednji nedostatak jest ne mogućnost ocjenjivanja pojedinog sadržaja. To znači da se sadržaj može samo komentirati, tako da bi jedini način utvrđivanja korisnosti pojedine informacije bilo čitanje njenih komentara.

5. Programsko rješenje

5.1. Uvod u programsko rješenje

U sljedećih nekoliko poglavlja bit će prikazana struktura programskog rješenja. Objašnjenje započinje sa strukturom baze podataka koja je prikazana na sljedećoj slici:



Slika 22. ERD Model

Prema modelu sa slike može se vidjeti da se baza podataka sastoji od dvanaest tablica koje su međusobno povezane. U modelu sa slike su, zbog jednostavnijeg pregleda, izostavljene veze između nekih tablica koje prate informaciju o korisniku koji je posljednji mijenjao određeni redak. Većina tablica ima primarni ključ *id* koji se automatski generira na bazi. Tablice: *favorit, rate_comment, rate_file, rate_class* i *professor_class* nemaju kolonu *id* jer nije potrebna kada već postoje dva vanjska ključa koja će biti u jedinstvenoj kombinaciji, što će jednoznačno identificirati redak. Također, u većini se tablica kontrolira valjanost podataka, odnosno, je li podatak prikazan korisniku ili nije, a to se kontrolira kroz polje *mark* u kojem se bilježe oznake N (nije označen kao nevidljiv odnosno izbrisan) ili D (označen je kao nevidljiv odnosno izbrisan). U sljedećih nekoliko odlomaka bit će pojašnjen sadržaj pojedine tablice.

U tablici *user* nalaze se informacije o korisniku, tj. njegovo ime, prezime, e-pošta, lozinka i oznaka. Prve četiri kolone opisuju svoje značenje dok polje *oznaka* treba biti dodatno pojašnjeno. Polje *oznaka* može poprimiti četiri vrijednosti:

- vrijednost *E*, što je ujedno i zadana vrijednost, koja označava da korisnik mora potvrditi svoj email kako bi mogao nastaviti s radom,
- vrijednost *N* koja označava da je korisnik potvrdio svoj email i može normalno raditi na aplikaciji,
- vrijednost *D* koja označava da je korisniku zabranjen pristup na aplikaciju,
- vrijednost *Y* koja označava da je korisnički račun obrisan.

U tablici *course* nalaze se informacije o smjerovima, točnije u njoj se nalazi podatak o imenu smjera i njegovom valjanosti koja se kontrolira kroz polje *mark.*

U tablici *class* nalaze se informacije o kolegijima, dakle, prati se ime kolegija, smjer kojem kolegij pripada, semestar izvođenja, status predmeta (redovni ili izborni), opis te polje *mark* kojim se kontrolira valjanost kolegija.

U tablici *obligation* nalaze se informacije o obavezama, tj. sadržaj obaveze i kolegij za kojega se obaveza veže.

U tablici *professor* nalaze se informacije o nastavnicima, točnije o svakom nastavniku bilježi se ime, prezime, titula, link na službenu stranicu FET-a te oznaka valjanosti podatka. Polje koje bilježi titulu može poprimiti pet vrijednosti:

- vrijednost *A* koja označava da je titula nastavnika: asistent,
- vrijednost *D* koja označava da je titula nastavnika: docent,
- vrijednost *S* koja označava da je titula nastavnika: suradnik,
- vrijednost *I* koja označava da je titula nastavnika: izvanredni nastavnik,
- vrijednost *R* koja označava da je titula nastavnika: redoviti nastavnik u trajnom zvanju.

U tablici *professor*_class nalaze se informacije o ulozi pojedinog nastavnika na pojedinom kolegiju. Polje koje bilježi ulogu nastavnika na kolegiju može poprimiti dvije vrijednosti:

- vrijednost *H* (engl. holder) koja označava da je nastavnik u ulozi nositelja kolegija,
- vrijednost *A* (engl. associate) koja označava da je nastavnik u ulozi suradnika na kolegiju.

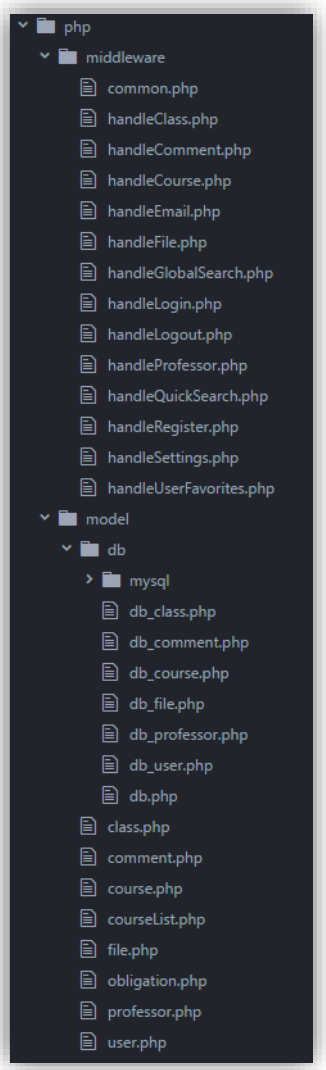
U tablici *comment* nalaze se komentari i pitanja. Dakle, za njih se prati korisnik koji ga je objavio, kolegij na kojem je objavljen komentar/pitanje, komentar ili pitanje koji mu je nadređen (ako se radi o pod komentaru/pitanju), datum i vrijeme objavljivanja, sadržaj, oznaku skrivenog identiteta koja može poprimiti vrijednosti *Y* (skriven identitet korisnika) i *N* (nije skriven identitet korisnika), te polje koje će označavati radi li se o komentaru ili pitanju što znači da će poprimiti vrijednost *C* (engl. *Comment*) ili *Q* (engl. *Question*).

U tablici *favorit* nalaze se korisnik i kolegij, odnosno prate se omiljeni kolegiji za pojedinog korisnika.

U tablicama *rate_comment, rate_file, rate_class* prate se ocjene koje korisnik daje pojedinom podatku. Svaka tablica ima polje *mark* koje će u ovom slučaju sadržavati ocjenu koju je korisnik dodijelio, što znači da polje može poprimiti dvije vrijednosti: *L* (engl. *Like* – sviđa mi se) i *D* (engl. *Dislike* – ne sviđa mi se).

5.2. Poslužiteljska strana

Na strani poslužitelja koriste se tehnologije MySQL i PHP. Tablice koje su kreirane na bazi su već prethodno prikazane pa prema tome, ovo poglavlje se ne osvrće na iste. Što se tiče organizacije PHP dijela aplikacije odlučeno je objektno se orijentirati, što ima smisla zbog cijelog objektnog koncepta aplikacije. U aplikaciji je definiran *Model* dio u kojem se nalaze sve potrebne klase za objektnu realizaciju podataka, a u pod direktoriju *db* definirane su statičke klase koje će služiti za komunikaciju s bazom. U *Middleware* dijelu se viši komunikacija s vanjskim sustavima odnosno odgovara na određene zahtjeve koristeći prethodno definirani *Model*.



Slika 23. Prikaz strukture organizacije PHP-a

Zbog lakšeg pronalaženja pogrešaka i smanjenja istih u budućnosti, svaka klasa ima određenu provjeru na atributima. Također, svaka klasa ima metode koje omogućuju pretvorbu objekta u JSON oblik, kako bi se podatak mogao poslati na klijenta. Sukladno tome većina klasa ima i metodu koja iz JSON zapisa gradi objekt. Pored prethodno navedenih komunikacijskih metoda, u klasama su definirane i metode koje omogućuju CRUD (kratica za engl. *Create, Read, Update, Delete*) operacije uz još mnoštvo drugih metoda specifičnih za pojedinu klasu. U svaku klasu je uključena odgovarajuća klasa za njezino komuniciranje s bazom.

```php
<?php

require_once 'db/db_course.php';
require_once 'class.php';

class Course{

  private $id;
  private $name;
  private $classes;

  function __construct($id, $name, $classes){
    $this->SetId($id);
    $this->SetName($name);
    $this->SetClasses($classes);
  }


  public function GetId(){return $this->id;}
  public function SetId($value){
    if($value == "") throw new InvalidArgumentException('ID smjera mora biti postavljen!');
    $this->id = $value;
  }

  public function GetName(){return $this->name;}
  public function SetName($value){
    if($value == "") throw new InvalidArgumentException('Ime smjera mora biti uneseno!');
    $this->name = $value;
  }

  public function GetClasses(){return $this->classes;}
  public function SetClasses($value){$this->classes = $value;}
  public function AddClass($value){array_push($this->classes, $value);}

  public function PrepareJSON(){
    return array(
          'id' => $this->GetId(),
          'name' => $this->GetName(),
          'classes' => array_map(function ($class) { return $class->PrepareJSON();}, $this->GetClasses())
      );
  }

  public function PrepareProfessorsJSON(){
    return array(
          'id' => $this->GetId(),
          'name' => $this->GetName(),
          'professors' => array_map(function ($professor) { return $professor->PrepareJSON();}, $this->GetClasses())
      );
  }
```

Slika 24. Primjer klase 1. dio (course.php)

Slika 25. Primjer klase 2. dio (course.php)

Klase koje komuniciraju s bazom se neće posebno opisivati. Bitno je samo naglasiti da su klase statičke, imaju metode koje omogućavaju CRUD operacije, a koriste klase koje su definirane u modelu.

Dakle, može se vidjeti da klase imaju metode koje omogućavaju komunikaciju s klijentom preko JSON zapisa, međutim, to još nije viđeno u upotrebi. Tu dolazi *Middleware* dio koji služi kao spojnica između model dijela i klijenta. U pojedinim datotekama u *Middleware* dijelu će biti smještena komunikacija između klijenta i poslužitelja. Standardni sadržaj takvih datoteka je kreiranje objekata, pozivanje metoda koje će taj podatak spremiti, ažurirati ili brisati uz druge metode specifične za pojedine klase. Podaci s klijenta će se uvijek dobivati kroz metodu POST, dok će odgovor, ako je zahtjev uspješno izvršen, biti u obliku JSON zapisa odnosno ako dođe do greške (npr. nisu poslana sva polja, korisnik nije prijavljen itd.) odgovor će biti u obliku zastavice (npr. 0 znači da korisnik nije prijavljen).

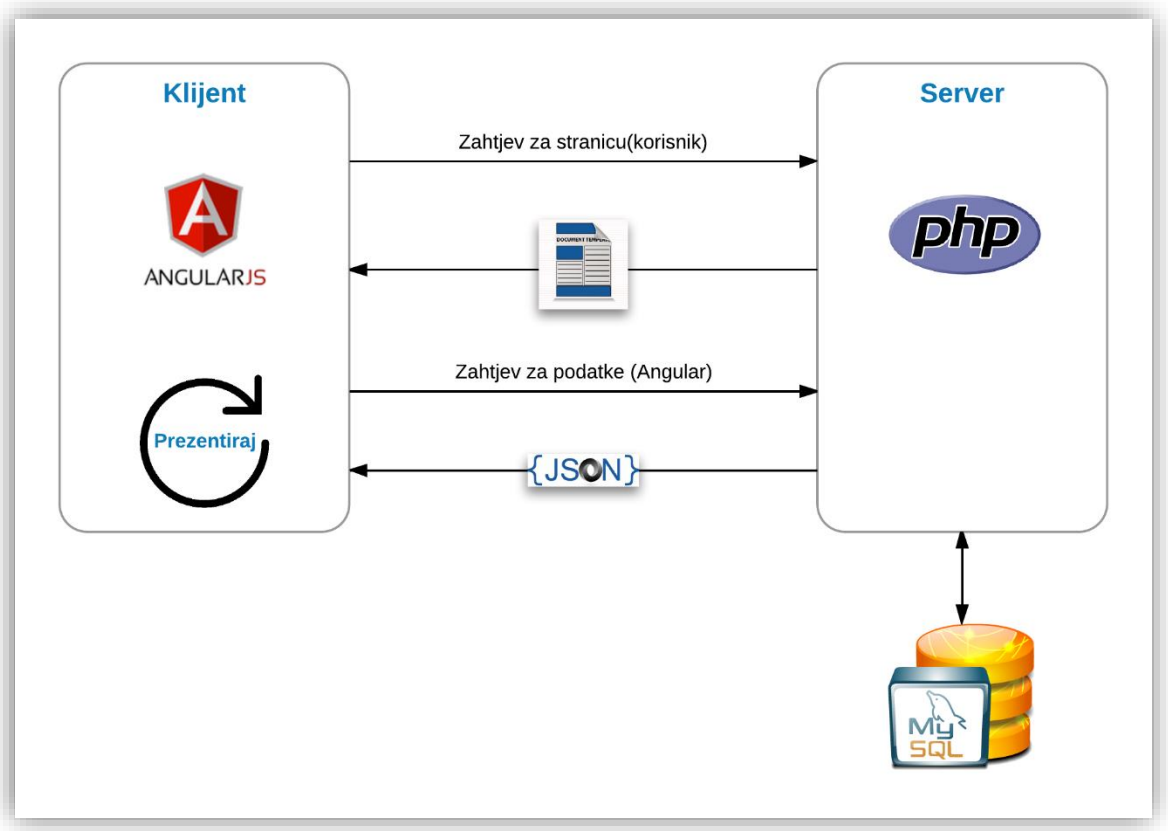
```php
<?php

require_once '../model/courseList.php';
require_once 'common.php';

if(CheckLogedIn()){

  if(CheckSetPost(array("id", "name")) ){

    if($_POST["id"] == "-1"){

      $course = new Course(-1, Capitalize($_POST["name"]),
                           isset($_POST["classes"])? _Class::BuildAll($_POST["classes"]): array() );

      $course->Save(GetSessionUser());
      echo $course->ToJSON();

    }else{
      $course = new Course( $_POST["id"], Capitalize($_POST["name"]),
                          isset($_POST["classes"])? _Class::BuildAll($_POST["classes"]): array() );
      $course->Update(GetSessionUser());
      echo $course->ToJSON();
    }

  }else if(isset($_POST["reqid"])){
    echo Course::GetCourseById($_POST["reqid"], GetSessionUser())->ToJSON();

  }else if(isset($_POST["deleteId"])){
    Course::MarkWithId($_POST["deleteId"], GetSessionUser());

  }else if(!isset($_POST["type"])){
    echo "0";
  }

}

if( isset($_POST["type"]) && $_POST["type"] == "short"){
   echo CourseList::GetCourseClassAssistantsToJSON();
}

?>
```

Slika 26. Primjer PHP-datoteke middleware dijela

5.3. Klijentska strana

Do sada je bio prikazan poslužiteljski dio aplikacije koji je prilično jednostavan u odnosu na klijentsku stranu. Klijentski sloj ima mnogo funkcionalnosti, kao što su pretraživanje bez osvježavanja stranice, tj. bez ponovnog dohvaćanja filtriranog sadržaja s poslužitelja, dodavanje novog podatka bez osvježavanja stranice, mijenjanje prikaza podataka na sljedeći set podataka bez osvježavanja stranice itd. Sve to dodatno komplicira „priču“ klijentskog sloja, međutim to je bio i cilj kako bi se

poboljšalo korisničko iskustvo u korištenju aplikacije, iako prvenstveno zbog smanjenog broja zahtjeva prema poslužitelju (tradicionalni način je da se nakon unosa osvježi stranica što bespotrebno ponovno šalje zahtjev prema poslužitelju za podatke koje je klijent već imao). Dodatna prednost kreiranja prezentacije na klijentu jest u količini podataka koja treba biti poslana do klijenta. Tradicionalni način jest da se prezentacija napravi na poslužiteljskoj strani pa se cijeli sadržaj šalje na klijenta, dok u ovom slučaju klijent dobije posebno predložak i posebno podatke (u JSON zapisu), te se onda prezentacija napravi na samom klijentu. Ovaj način je puno brži, pogotovo kod velike količine podataka, a korisniku se početni prikaz jako brzo učita jer dobije samo predložak bez podataka koji se pročita kao obična statična stranica.



Slika 27. Prikaz principa dohvaćanja sadržaja s poslužitelja klijentu

Na slici 25. je prikazan princip dohvaćanja sadržaja s poslužitelja klijentu koji je korišten u aplikaciji. Na slici možemo vidjeti da korisnik prvo zatraži stranicu na što mu poslužitelj odgovori sa statičnim predloškom koji je definiran za tu stranicu. Kod dohvaćanja predloška dohvaća se i Controller koji nakon učitavanja šalje zahtjev prema poslužitelju za određeni set podataka. Poslužitelj na zahtjev odgovara u JSON zapisu da ga AngularJS na strani klijenta može pretvoriti u objekt. Nakon što AngularJS pretvori odgovor u objekt potvrđuje ga na modelu tako da se dohvaćeni sadržaj prikaže korisniku. Na slici je prezentiranje prikazano kao „Prezentiraj“, a ono simbolizira iteriranje kroz dohvaćeni sadržaj, te kreiranja HTML elemenata (što bi se tradicionalno radilo na poslužitelju).

```
<table style="margin-top: 40px;" class="table table-hover table-responsive">

  <tbody ng-repeat="course in data.courses track by $index">

      <tr class="pointer" ng-click="course.expand = !course.expand"
          ng-show="(filtered.length > 0) || (course.classes.length == 0 && search == '' && selectRI =='')"
          ng-init="course.expand = false;">

        <th class="col-lg-8" scope="row">
            <span class="glyphicon {{course.expand?'glyphicon-collapse-down' : 'glyphicon-expand pointer'}}"></span>
            <a href="course.php?id={{course.id}}" ng-click="course.expand = !course.expand">{{course.name}}</a>
        </th>
        <td class="col-lg-1"></td>
        <td class="col-lg-3 text-right"><span class="badge">{{filtered.length}}</span></td>

      </tr>

      <tr ng-repeat="class in course.classes | filter:{status:selectRI} | filter:search as filtered" class="sub-row" ng-show="course.expand">

          <th class="col-lg-8" scope="row"><span class="glyphicon glyphicon-chevron-right"></span> <a href="class.php?id={{class.id}}">{{class.name}}</a></th>

          <td class="col-lg-1">
            <a class="btn" uib-popover-template="'PopoverTemplate.html'" popover-trigger="'outsideClick'" popover-title="Nastavnici" popover-placement="bottom" >
              <span style="color:#808080;" class="glyphicon glyphicon-user"></span>
            </a>
          </td>

          <td class="col-lg-3 text-right">
            <span class="label label-{{class.status == 'R'? 'primary' : 'success'}}">{{class.status == "R"? "Redovni" : "Izborni"}}</span>
            <span class="label label-warning">{{convertSemester(class.semester)}} semestar</span>
          </td>

      </tr>

  </tbody>

</table>
```

Slika 28. Većinski dio predloška indeksne stranice

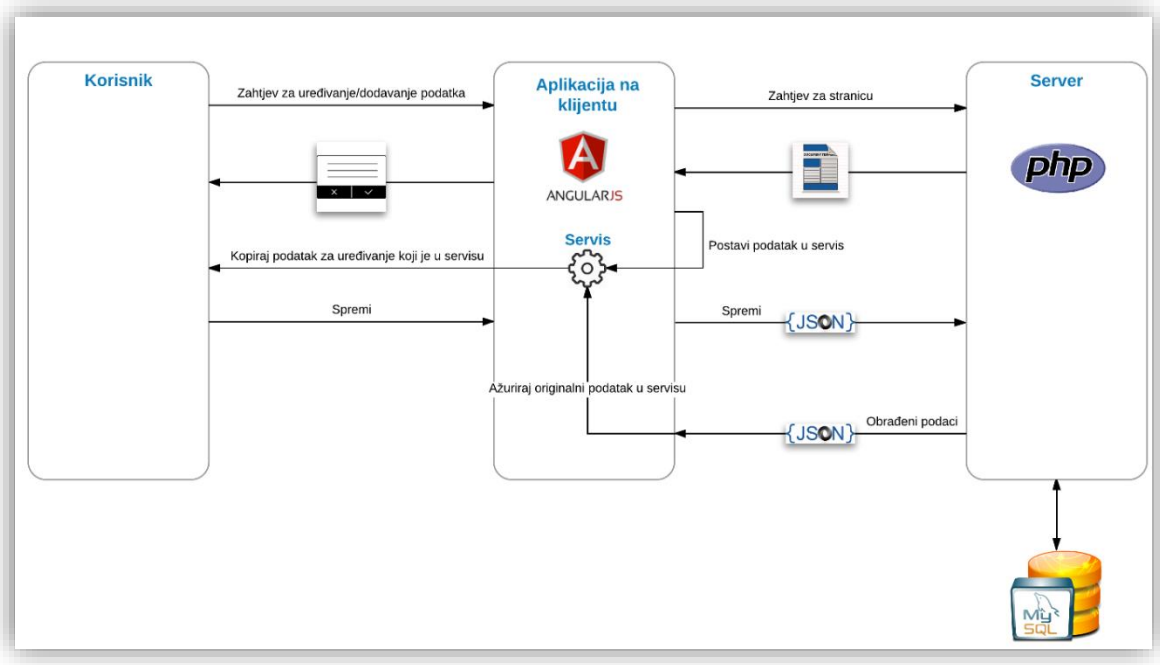
Na slici 26. je primjer predloška koji se inicijalno dohvaća na korisnikov zahtjev. Na slici nije prikazan cijeli predložak već samo dio koji se brine za kreiranje tablice. Predložak sam po sebi neće korisniku prikazivati nikakav podatak, već se podaci moraju dodatno dohvatiti s poslužitelja kako je prethodno opisano, što je prikazano na sljedećoj slici.

```
RequestData('php/middleware/handleCourse.php', function(data) {
    $scope.data = JSON.parse(data);
    $scope.$apply();
    hideLoading();
},{'type':'short'});
```

Slika 29. Dohvaćanje podataka s poslužitelja

Funkcija *RequestData* radi na principu asinkronog zahtjeva što znači da se stranica na klijentu neće blokirati dok se podaci dohvaćaju već se na klijentu može nešto izvoditi dok ne dođe odgovor s poslužitelja, kao što je npr. prikaz učitavanja. Funkcija prima dva ili tri argumenta od kojih je prvi argument put do stranice s koje se zahtijevaju podaci, kao drugi argument se prosljeđuje funkcija koja će se izvesti nakon što dobije odgovor s poslužitelja, i treći argument, koji nije obavezan, jest podatak koji funkcija šalje prilikom zahtjeva. Kada funkcija dobije podatke ona ih prevodi iz JSON zapisa u objekt i potvrdi nove podatke da se odraze na model. Nakon potvrde AngularJS na prethodno prikazanom predlošku izvodi prezentaciju čime korisnik u ovom slučaju dobije tablični prikaz svih smjerova i kolegija koji su vezani na smjer.

Kod ažuriranja ili unosa novog podatka je komunikacijski proces između klijenta i poslužitelja sličan. Bitno je napomenuti da niti jedan ekran za unos ili izmjenu, nije inicijalno dohvaćen na klijenta kako bi se što više smanjilo učitavanje nepotrebnog sadržaja na klijenta, već se HTML ekran za uređivanje mora dohvatiti s poslužitelja kada ga korisnik zatraži. Komunikacijski proces je vrlo brz i lagan jer su podaci koji se izmjenjuju između klijenta i poslužitelja samo oni podaci koji se uređuju. Dakle, ne šalje se cijela stranica poslužitelju što je tradicionalni način rada web aplikacija.



Slika 30. Prikaz komuniciranja korisnika, aplikacije na klijentu i poslužitelja kod uređivanja podatka

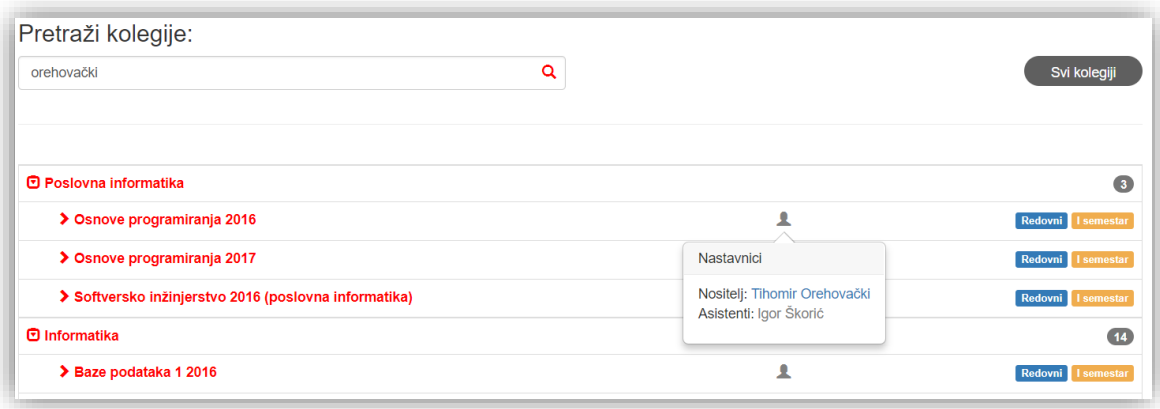
Na slici 28. je prikazan komunikacijski proces između korisnika, klijenta i poslužitelja prilikom uređivanja nekog podatka. Inicijalno, korisnik zatraži ekran za uređivanje nekog podatka čime aplikacija šalje zahtjev prema poslužitelju za tu datoteku kroz koju će se podatak uređivati. Nakon toga se taj dohvaćeni sadržaj dodaje na stranicu i otvara u obliku prozorčića. Podatak koji korisnik uređuje se dohvaća s aplikacije na klijentu, odnosno, ne dohvaća se njegov sadržaj s poslužitelja jer nije potrebno. Ekran koji služi za uređivanje podatka ima svoj *Controller* pa se u svrhu razmjene podataka između dva *Controller*-a koriste servisi. Tako da *Controller* koji kontrolira uređivanje napravi kopiju podatka čime korisnik uređuje kopiju podatka. Ako korisnik odustane, prozorčić se zatvara i u tom slučaju nema nikakve komunikacije s poslužiteljem. Međutim, ako korisnik spremi podatak, poziva se funkcija *SubmitData* koja nije asinkrona, što znači da blokira stranicu dok ne dobije odgovor od poslužitelja. Poslužitelj obrađuje podatak, odnosno, sprema ga ili ažurira ovisno o njegovom statusu (dodan novi ili uređen postojeći podatak). Nakon obrade podatka poslužitelj vraća obrađeni podatak natrag klijentu. Na klijentu se zatim sinkronizira podatak koji je unesen ili izmijenjen s onim kojeg je poslužitelj vratio nakon obrade, tako da se

ažurira originalni podatak koji je postavljen u servisu. Prozorčić se nakon uspješnog spremanja zatvara i korisnik može nastaviti s radom.

6. Mogućnosti aplikacije

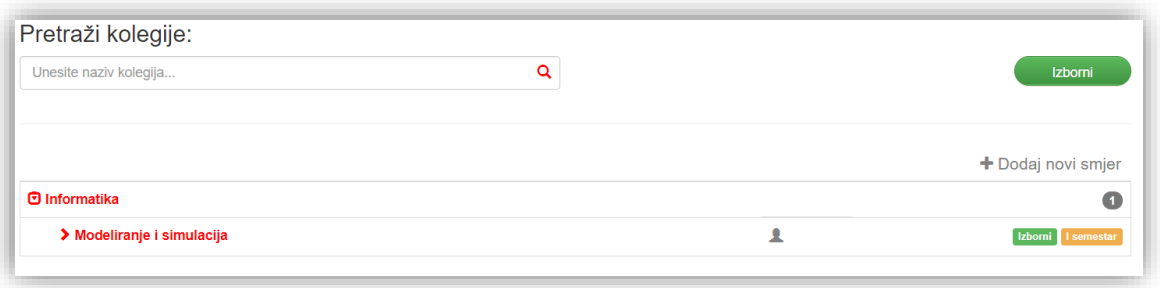
6.1. Korisničke mogućnosti

U ovom će poglavlju biti prikazan rad web aplikacije s njenim glavnim mogućnostima. Aplikacija je napravljena tako da korisnik može pregledavati samo naslovnu stranicu dok nije prijavljen, točnije, tek kad se prijavi može pregledavati i uređivati sav sadržaj na aplikaciji. Mogućnosti koje su dostupne svim korisnicima (prijavljenim i ne prijavljenim) su filtriranje svih kolegija prema unesenom pojmu ili prema njegovom statusu (redovan ili izborni). Kada se filtriraju kolegiji prema unesenom pojmu, onda filter rekurzivno pretražuje sve atribute objekata tako da se mogu dobiti npr. svi kolegiji u kojima se pojavljuje određeni nastavnik, bez obzira na status nastavnika (nositelj ili suradnik).



Slika 31. Prikaz filtriranja kolegija prema nastavniku

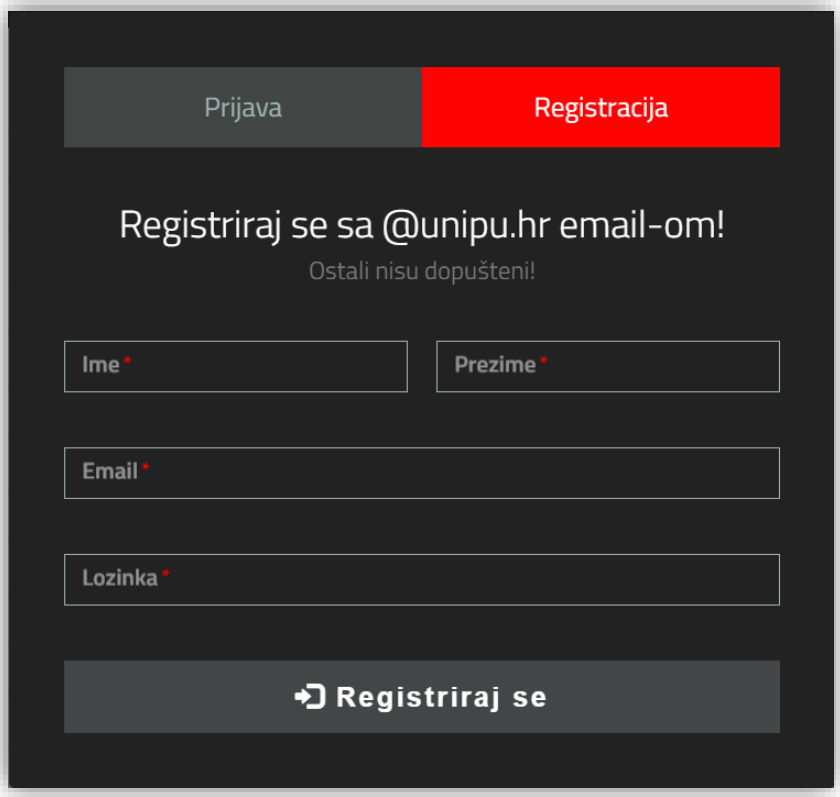
Kao što je prethodno spomenuto, na aplikaciji se dodatno može filtrirati prema statusu kolegija (redovan ili izborni) klikom na gumb koji mijenja filter.



Slika 32. Prikaz filtriranja kolegija prema statusu

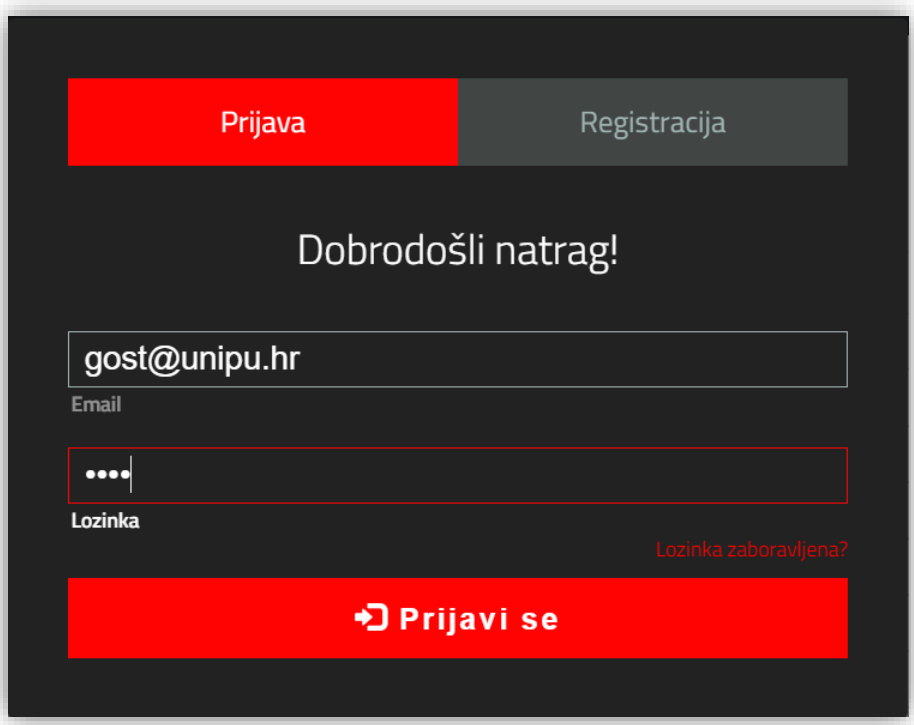
6.1.1. Prijava odnosno registracija

Kako bi korisnik mogao u potpunosti koristiti aplikaciju mora se prijaviti, a u slučaju da to već nije napravio mora se registrirati. Sljedeća slika prikazuje registraciju.



Slika 33. Prikaz ekrana za registraciju

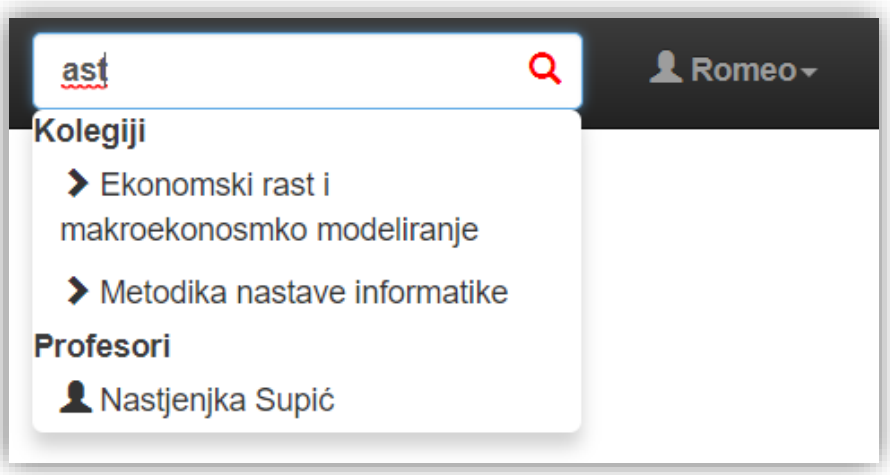
Kod registracije korisnik unosi svoje ime, prezime, e-poštu i lozinku. E-pošta mora biti iz domene @unipu.hr što je mala sigurnosna mjera kako bi pristup aplikaciji imali samo studenti. Ako je e-pošta već korištena kod registracije na aplikaciji, korisniku se javlja poruka o grešci. Nakon uspješne registracije korisnika se usmjerava na stranicu računa e-pošte kako bi ga potvrdio. U e-pošti je poveznica kojom se e-pošta korisnika potvrđuje, nakon čega se može prijaviti i započeti koristiti aplikaciju.



Slika 34. Prikaz ekrana za prijavu

Kod prijave korisnik unosi svoju e-poštu i lozinku koju je unio pri registraciji. Ako su podaci netočni, korisniku se javlja poruka o grešci. U slučaju da je korisnik zaboravio svoju lozinku ima mogućnost ponovnog postavljanja lozinke tako da mu aplikacija na unesenu adresu e-pošte šalje privremenu lozinku koju može već kod sljedeće prijave promijeniti.

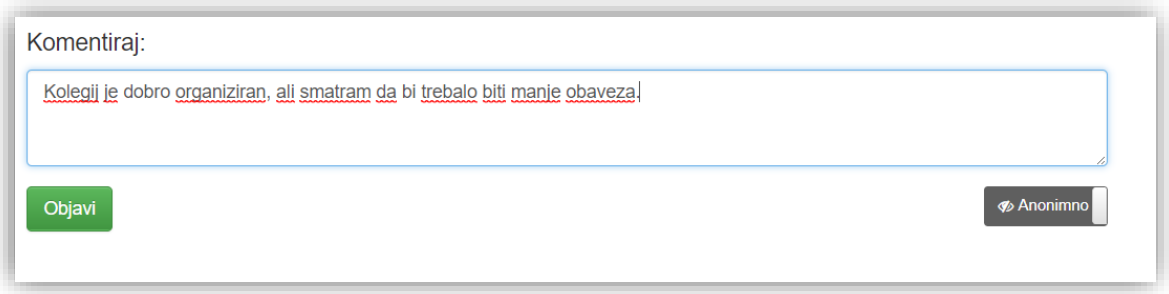
Nakon registracije i uspješne prijave korisnik može pregledavati i uređivati sav sadržaj na aplikaciji. Sljedeća mogućnost omogućuje prijavljenom korisniku brzo pretraživanje pojma. Korisnik mora samo napisati željeni pojam i aplikacija mu vrati rezultate za traženi pojam.



Slika 35. Prikaz brzog pretraživanja

6.1.2. Objavljivanje komentara ili pitanja

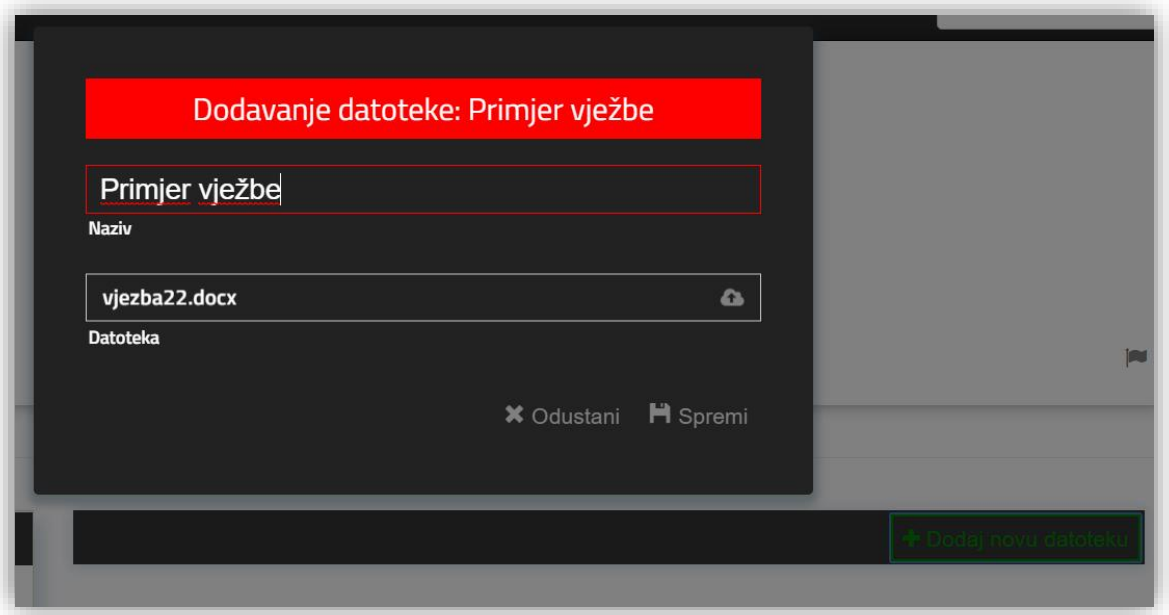
Korisnik ima mnogo mogućnosti kao što su: uređivanje smjera, uređivanje kolegija, uređivanje nastavnika. Njima se pak ne bavi u ovome radu, nego će se fokus usmjeriti prema mogućnostima koje će se puno češće koristiti kao što su uređivanje komentara, uređivanje datoteka itd. Na sljedećoj slici je prikaz forme za objavljivanje komentara gdje korisnik upisuje svoj komentar i gdje mu se nudi mogućnost da objavi komentar anonimno ili javno, ovisno o tome želi li sakriti svoj identitet prilikom prikaza komentara. Na sličan način korisnik može komentirati i neki komentar.



Slika 36. Prikaz objavljivanja komentara

6.1.3. Postavljanje materijala ili poveznica

Kada korisnik želi dodati datoteku kliknut će na gumb *Dodaj novu datoteku* i otvara mu se prozorčić za učitavanje datoteke na kojoj mora dodijeliti ime datoteci što prikazuje sljedeća slika.



Slika 37. Prikaz ekrana za unos nove datoteke

Nakon što korisnik odluči spremiti datoteku počinje učitavanje datoteke na poslužitelja. Nakon uspješnog spremanja prozorčić se zatvara i nova datoteka se prikazuje u popisu. Unos poveznica je vrlo sličan prikazanom unosu datoteka. Ako korisnik želi ažurirati postojeću datoteku ili poveznicu otvara mu se isti prozorčić gdje ima mogućnost izmjene i brisanja datoteke ili poveznice.

6.2. Administrativne mogućnosti

U aplikaciji još nisu realizirane uloge običnog korisnika i administratora. Međutim zbog najveće funkcionalnosti, a to je da svi korisnici mogu uređivati sav sadržaj, trebao bi postojati administrator s dodatnim ovlastima koji će pratiti izmjenu i brisanje podataka. Administrator bi imao mogućnosti dodjeljivanja administratorskih ovlasti drugim korisnicima, onemogućavanje korištenja aplikacije korisniku zbog nepravilnog korištenja aplikacije itd. Ova je funkcionalnost ostavljena kao prostor unaprjeđenja aplikacije za sljedeću verziju.

7. Socijalni aspekt aplikacije – gejmifikacija

U ovom će poglavlju biti pojašnjen socijalni aspekt aplikacije, tj. kakav bi utjecaj aplikacija potencijalno mogla ostvariti na studente kroz elemente gejmifikacije implementirane u nju. Prije svega potrebno je pojasniti pojam gejmifikacija jer nije tako učestao izraz.

> Gejmifikacija je vještina poticanja svih zabavnih i ovisnih elemenata koji se mogu pronaći u igrama primjenjujući ih u stvarnom svijetu ili produktivnim aktivnostima. Ja to nazivam „Ljudski-fokusiran“ dizajn koji je suprotnost „Funkcionalno-fokusiranom“ dizajnu. To je proces dizajniranja koji je optimiziran za čovjeka u sustavu, za razliku od čiste učinkovitosti sustava.
>
> Većina sustava su „funkcionalno-fokusirano“ dizajnirani kako bi odradili posao brzo. To je kao tvornica koja pretpostavlja da će njeni radnici odraditi svoj posao. Međutim, ljudski-fokusiran dizajn se prisjeća da ljudi unutar sustava imaju osjećaje, nesigurnosti i razloge zašto žele ili ne žele nešto napraviti, pa je time dizajn optimiziran prema njihovim osjećajima, motivacijama i angažmanu (Chou, 2017).

Nakon upoznavanja s gejmifikacijom vrijedi prikazati njene elemente koji su implementirani u aplikaciji. Sama aplikacija je socijalnog karaktera, što znači da se studenti dijeljenjem informacija međusobno povezuju gdje im se otvara mogućnost zajedničkog rada i napretka. Time im se povećava kako volja, tako i želja za radom i napretkom. Drugi element gejmifikacije jest ocjenjivanje pojedine informacije, tj. ocjenjivati se mogu kolegiji, komentari, pitanja, datoteke i poveznice. Za datoteke i poveznice se ne može vidjeti koja je osoba objavila tu informaciju dok se za pitanja i komentare može vidjeti ime i prezime osobe koja je informaciju objavila (osim ako pitanje ili komentar nije objavljen u anonimnom načinu). Elementi gejmifikacije su upravo u tim situacijama najviše zastupljeni. Student koji objavi informaciju dobije povratnu informaciju od ostalih studenata o korisnosti objavljene informacije u postotnom obliku. Student dobije osjećaj postignuća, slično kao što je u igrici prolaz na

sljedeću razinu, što ga dalje motivira da napreduje odnosno da prijeđe na sljedeću razinu. Za razliku od elemenata gejmifikacije u igricama koji korisnika „navuku“ da troši svoje vrijeme bez ikakvog stvarnog postignuća i napretka u stvarnom svijetu, elementi gejmifikacije u studentskom ili poslovnom svijetu su poželjni jer stimuliraju korisnika da širi svoje znanje i samopouzdanje na zabavan način.

Slika 38. Prikaz elementa gejmifikacije (ocjenjivanje datoteke)

8. Zaključak

U radu su prikazani osnovni koncepti koji se trenutno koriste kod izrade web aplikacija, te njihove prednosti nad tradicionalnim konceptima. Tradicionalni koncept izrade aplikacija je prilično jednostavan - na poslužiteljskoj strani se koristi sustav za upravljanje bazom podataka (npr. Oracle, MySQL i sl.) i programski jezik (npr. Java, PHP i sl.) koji komunicira s bazom podataka kako bi generirao sadržaj ili ažurirao podatke. Glavni nedostaci takvog koncepta su: ne mogućnost ponovnog korištenja dijela aplikacije u druge svrhe (npr. izrada druge aplikacije koja će te iste podatke grafički prikazivati), brzina rada (cijeli sadržaj se generira na poslužitelju pa se šalje klijentu), osvježavanje stranice kod svake operacije i nepotrebno ponovno dohvaćanje podataka.

Današnji najčešći pristup kod izrade web aplikacija jest kreiranje web servisa i aplikacije koja komunicira s tim servisom. Tehnologije i koncepti koji su prikazani u radu objašnjavaju jedan od mogućih načina realizacije takve strukture. Za kreiranje web servisa korišten je programski jezik PHP u kombinaciji s MySQL bazom podataka. Web servis s korisnikom servisa komunicira kroz JSON format koji je, u odnosu na XML, jako lagan format zapisa. Korisnik servisa je u ovom slučaju aplikacija na klijentu koja je najvećim dijelom realizirana kroz aplikacijski okvir AngularJS. Aplikacija na klijentu radi tako da podatke čita i ažurira s prethodno spomenutog web servisa. Aplikacija drži podatke na klijentu sve dok korisnik ne osvježi, zatvori ili izađe sa stranice, tj. prilikom mijenjanja podatka aplikacija na klijentu mijenja samo taj podatak ne osvježavajući ostale podatke, čime se opterećenje na poslužitelj smanjuje i poboljšava cjelokupni korisnički doživljaj.

Koncepti izrade aplikacija nisu napredovali samo u tehničkom smislu već i u psihološkom smislu. Dosadašnji pristup gdje je aplikacija napravljena da zadovolji funkcionalne zahtjeve više nije dovoljan, korisnika treba zainteresirati za korištenje aplikacije kako bi bio produktivniji i željniji rada. To se može postići implementiranjem gejmifikacijskih elemenata u aplikaciju. Ukratko, gejmifikacija znači uzimanje nekih koncepata koji se koriste kod izrade video igara i implementiranje u druge svrhe (u ovom slučaju aplikaciju) u svrhu povećanja zainteresiranosti korisnika u korištenju usluge ili proizvoda (u ovom slučaju aplikacije).

Ako se osvrnemo na današnje web aplikacije možemo zaključiti da će one u budućnosti zamijeniti *desktop* aplikacije, pogotovu o kombinaciji s *Cloud*-om. Područje web aplikacija i tehnologija koje se razvijaju da bi web aplikacije bile brže, bolje, interaktivnije i sl. je svakim danom sve šire, samim time i zanimljivije.

Izrađena aplikacija je podignuta na hosting i može se posjetiti na adresi http://adik.newapp.ml/. Prijava se može izvršiti s e-poštom: gost@unipu.hr i lozinkom: gost, dok se testni podaci nalaze na adresi http://adik.newapp.ml/class.php?id=232.

9. Literatura

1) Abeysinghe, S., (2008), RESTful PHP Web Services, Birmingham: Olton, dostupno na: https://books.google.hr/books?id=jklKNnLO104C&printsec=frontcover&dq=restful&hl=hr&sa=X&ved=0ahUKEwji_Or6pLnVAhVIOxoKHQitA5MQ6AEITjAF#v=onepage&q&f=true [31.7.2017.].
2) Bootstrap, (n.d.), dostupno na: http://getbootstrap.com/about [1.8.2017.].
3) Chou, Y., (2017), dostupno na: http://yukaichou.com/gamification-examples/what-is-gamification/ [6.8.2017.].
4) FER2.net, (2017), dostupno na: http://www.fer2.net/ [3.8.2017.].
5) Green, B. & Seshadri, S., (2013), AngularJS, 1st ed., Sebastopol: Gravenstein Highway North, dostupno na: ftp://94.244.139.11/lit/1.%20Manuals/AngularJS.pdf [5.3.2017.].
6) HTML.com, (n.d.), dostupno na: http://html.com/html5/ [1.8.2017.].
7) MySQL, (2017), dostupno na: https://www.mysql.com/why-mysql/ [2.8.2017.].
8) Nixon, R., (2015), Learning PHP, MySQL & JavaScript with jQuery, CSS & HTML, 4th ed., Sebastopol: Gravenstein Highway North, dostupno na: https://doc.lagout.org/programmation/Learning%20PHP,%20MySQL%20%26%20JavaScript_%20with%20jQuery,%20CSS%20%26%20HTML5%20(4th%20ed.)%20%5BNixon%202014-12-14%5D.pdf [12.3.2017.].
9) PHP, (n.d.), dostupno na: http://php.net/manual/en/intro-whatis.php [8.3.2017.].
10) Smith, B., (2015), Beginning JSON, New York: Spring Street, dostupno na: http://choonsiong.com/public/books/JavaScript/Beginning%20JSON.pdf [14.3.2017.].
11) Trello, (2017), dostupno na: http://help.trello.com/ [3.8.2017.].
12) W3Tech, (2017), dostupno na: https://w3techs.com/technologies/details/js-jquery/all/all [21.3.2017.].

10. Popis slika