SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET
Preddiplomski studij informacijskih i komunikacijskih znanosti

POTPUNA IZRADA SUSTAVA ZA EVALUACIJU KOLEGIJA

Završni rad

Mario Mikić

Mentor: dr.sc. Vedran Juričić

Zagreb, 2016.

Sažetak

U ovom radu obrađen je potpuni proces izrade web aplikacije za evaluaciju kvalitete kolegija. Za izradu aplikacija korišten je niz tehnologija, razvojnih okvira i alata uz koje je pojašnjena njihova uporaba i generalna funkcija. Za razvoj korisničkog sučelja korišten je Javascript Vue.js razvojni okvir te Model-View-ViewModel paradigma, dok je u administratorskom sučelju i serverskoj logici aplikacije korišten PHP i Laravel razvojni okvir, te učestala Model-View-Controller paradigma. Osim osnovne logike iza aplikacije obrađen je i kratak povijesni pregled spomenutih tehnologija, a i niz dodatnih često korištenih alata u današnjem standardnom razvojnom okruženju.

Ključne riječi: Javascript, PHP, MVC, MVVM, web aplikacije

Development of a system for the evaluation of faculty courses

Abstract

This paper goes through the complete process of developing a web application for the evaluation of faculty courses. While developing the aforementioned application a wide variety of technologies, frameworks and toolsets were used. For the development of the front-end interface, Javascript and Vue.js framework were used, along with the Model-View-ViewModel paradigm implemented by the framework. On the administrative part of the application, and the server-side logic of the application, the programming language used was PHP and the Laravel framework with its more common Model-View-Controller paradigm. Additionally to the development process, the paper goes through a short history of the used technologies, and also through a list of commonly used toolsets in a standard development environment.

Keywords: Javascript, PHP, MVC, MVVM, web applications

SADRŽAJ

Popis slika:

Popis tablica:

1. UVOD

1.1. Ideja

Sustav za evaluaciju kolegija nastao je kao projekt za kolegij *Web 2.0 aplikacije* pri Filozofskom fakultetu Sveučilišta u Zagrebu. Inicijalno zamišljen kao projekt vezan uz kolegij, kasnije je obrađen u proširenom obliku za potrebe završnog rada pri Odsjeku za informacijske i komunikacijske znanosti Filozofskog fakulteta u Zagrebu.

Ideja aplikacije dolazi iz razgovora s kolegama na odsjeku o potrebi konstantne evaluacije kvalitete kolegija. Riječ je o sustavu za evaluaciju kolegija gdje studenti mogu anonimno ispunjavati ankete koje bi se kasnijom obradom prikazivale nositeljima kolegija u obliku konstruktivnih kritika za unaprijeđenje kvalitete kolegije.

1.2. Struktura

Sustav za evaluaciju kolegija dijeli se na dva osnovna segmenta koja funkcioniraju gotovo neovisno jedno o drugog. Prva dio naziva se Administrativno sučelje i pruža funkcionalnost administracije Sustava. Drugi dio naziva se Korisničko sučelje i pruža primarnu funkciju Sustava što je evaluacija kolegija.

1.2.1. Autorizacija korisnika

Sustav je izrađen na principu da se korisnik prvo mora autorizirati kako bi mogao pristupiti ispunjavanju anketa. Takav pristup na prvu ostavlja dojam da ankete nisu anonimne jer sustav prati ispunjenost anketa na temelju korisnika Međutim razlog obavezne autorizacije je drugog podrijetla. Idenficiranjem korisnika moguće je ograničiti dostupnost kolegija koji se ocjenjuju, a također i pratiti ispunjenost kolegija, odnosno prijestupe u pogledu višestrukog ispunjavanja ankete.

Dostupnost kolegija je prvi važan faktor koji pridonosi kvaliteti rezultata Sustava. Uzimanjem informacija poput godine studija moguće je učitavati samo kolegije koje je korisnik do tog

trenutka imao priliku odslušati. Time se značajno smanjuje mogućnost ispunjavanja netočnih podataka.

Dodatna razina validacije je registracija korištenjem isključivo *ffzg.hr* vršne domene čime se onemogućuje registracija korisnika koji nisu aktivni članovi Filozofskog fakulteta.

Važno je napomenuti kako unatoč navedenim preventivnim mjerama nije moguće u potpunosti točno pratiti uvjete za ispunjavanje određene ankete. Takva mogućnost mogla bi se ostvariti jedinu u trenutku kada bi se autorizacija ostvarila korištenjem *AAI@edu.hr* identiteta čime bi se dobio uvid u stvarno stanje odslušanih kolegija korisnika. Time bi efektivno nestala i generalna komponenta korisnika koja je trenutno ugrađena u Sustav, a potencijalno i komponenta kolegija, što bi u konačnici umanjilo sustav za cijeli Administrativni dio.

1.2.2. Korištenje sustava kao korisnik

Jedina funkcija koju korisnik (student) može koristiti u sklopu Sustava jest ispunjavanje anketa o kolegijima. Sva dodatna funkcionalnost nije moguća jer cilj Sustava nije prikazivanje statistike kolegija već samo njena evaluacija. Dostupnost statistike i informacija sustava seže izvan opsega temeljne zamisli i ovisi o administratorima Sustava i dogovoru o diseminaciji *sirovih* podataka i statistika.

Ankete se mogu ispunjavati samo jednom, ponovno ispunjavanje ankete nije omogućeno jer ne postoji nikakav do sad otkriven objektivni argument koji bi pridonio kvaliteti rezultata pružanjem navedene mogućnosti.

1.2.3. Korištenje sustava kao profesor

Sustav u svojoj strukturi podržava mogućnost integracije profesorskog sučelja čime bi se svaki profesor korištenjem svojih korisničkih podataka mogao autorizirati u Administrativni dio Sustava i pregledavati statistike uz vezane kolegija. Međutim kao što je navedeno, funkcionalnost obrade i objavljivanja rezultata seže izvan inicijalnog opsega sustava te s toga korištenje Sustava za profesore nije moguće. Ipak, pružanje mogućnosti pristupa Sustavu moglo bi se jednostavno izraditi jer u samom temelju su kolegiji vezani uz korisnike u sustava.

1.2.4. Korištenje sustava kao administrator

Administrativni dio Sustava izrađen je isključivo za korisnike koje imaju administratorsku razinu pristupa. U sklopu administrativnog dijela omogućena je sva funkcionalnost potrebna za kontrolu korisnika, kolegija i anketa u Sustavu.

U pogledu anketa, sustav je zamišljen tako da se ankete mogu vezati primarno uz kolegije. U sklopu Sustava anketa se naziva *Grupom pitanja* koja može sadržavati niz pitanja. Uz kolegije može se vezati neograničen broj grupa, a idejno bi svaka grupa predstavljala jednu smislenu cjelinu, primjerice "Seminari" ili "Predavanja".

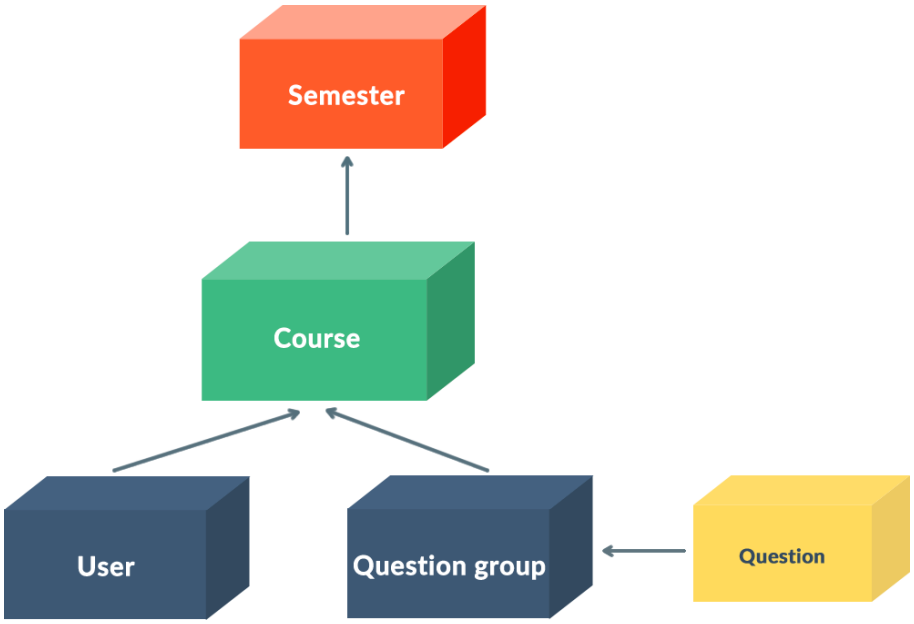
Struktura sustava postavljana je tako da je moguće razviti i ocjenjivanje samih profesora, međutim navedena funkcionalnost nije implementirana jer izlazi iz inicijalne speficikacije funkcionalnosti.

Pojedinačna pitanja, koja su grupirana u Grupe pitanja trenutno po vrsti pitanje trenutno mogu spadati u jednu od tri kategorije. Prva moguća kategorija su pitanja istinitosti, odnosno da/ne pitanja. Druga kategorija je skala za ocjenjivanje od jedan do pet, dok je treća kategorija polje za upis odgovora. Polje za upis odgovora se pak dijeli na kratke i dugačke (esejske) odgovore, ali riječ je o čisto tehničkoj razlici unutar kategorije.

1.2.5. Arhitektura

Temeljni entitet oko kojeg je koncipiran sustav jest Kolegij. Kolegij je vezan na entitet Semestra, ali na sebe veže sve ostale entitete. Tako Kolegij uz sebe veže entitet Korisnika, bilo to profesor ili student, i Grupe pitanja. Grupa pitanja veže pojedinačno pitanje.

Grafički bi se ukupna povezanost Entiteta mogla prikazati na sljedeći način:



Slika 1: Grafički prikaz entiteta u sustavu.

1.2.6. Zaključak

Sustav za evaluaciju kolegija izrađen je na način da omogućuju dodatno proširivanje funkcionalnosti. Određene nove funkcionalnosti već su na početku uočene, ali zbog jednostavnosti razvoja nisu implementirane.

Detaljnija razrada modela kao i tehnički pristup stvaranju relacija pojašnjen je u sljedećem poglavlju koje se bavi tehničkim aspektom izrade Administrativnog dijela Sustava.

2. ADMINISTRATORSKO SUČELJE I SERVERSKA STRANA

2.1. Uvod

U prethodnom poglavlju ukratko je pojašnjena temeljna specifikacija *Sustava za evaluaciju kolegija.* Sustav je izrađen u obliku web aplikacije zbog jednostavnosti pristupa od strane korisnika, čime je omogućen pristup s bilo koje lokacije i bilo kojeg uređaja koji ima mogućnost spajanja na Internet.

Dodatna odluka koja je donesena jest korištenje PHP programskog jezika za realizaciju serverske strane rješenja aplikacije. PHP je odabran zbog svoje velike rasprostranjenosti i dostupnosti, kao i dostupnosti velikog niza razvojnih okvira koji omogućuju produkciju Sustava u kratkom vremenskom roku.

2.1.1. PHP programski jezik

PHP, skraćeno od *Hypertext Preprocessor* je (primarno) serverski programski jezik namijenjen za izradu web aplikacija. Izrađen 1994. godine, a originalno je skraćenica značila *Personal Home Tools*.[1] Izrađen je kao kolekcija CGI skripti pisanih u *Pearlu* kako bi ubrzao izradu određenih funkcionalnosti - u početku brojača posjeta web stranice. Kroz godina PHP doživljava niz inačica koje iz samih temelja izmjenjuju mogućnosti PHP-a, konstantno nadograđujući, ali i mijenjajući sintaksu i mogućnosti čime dolazi na loš glas u programerskoj zajednici. Kao što je navedeno, početno je PHP razvijen kao kolekcija *Pearl* skripti, u svojoj idućoj iteraciji je nanovo napisan u C-u, a objektno-orijentirani pristup sintaktički preuzima iz C++-a.[2]

Međutim PHP nosi i svoje pozitivne strane. Primarno, dostupan je gotovo svugdje. Omogućuje brzo i efikasno razvijanje aplikacija, odličan je programski jezik za početnike jer su rezultati odmah vidljivi, a presudan faktor koji bi mogao ostaviti i iskusnije programere jest značajan napredak u posljednjih godina sa izlaskom novih verzija (5.6. ili najavljeni PHP 7).

[1] PHP manual: History of PHP. URL: <http://php.net/manual/en/history.php.php> (12.09.2016.)
[2] PHP: A fractal of bad design. URL: <http://eev.ee/blog/2012/04/09/php-a-fractal-of-bad-design/> (12.09.2016.)

2.1.2. Composer dependency manager

Svojevrsnom revolucijom u svijetu PHP-a smatra se pojava *Composer* menadžera ovisnosti (*eng. dependency managera*) za PHP kojim je omogućen jednostavan i brz način ugradnje PHP programskih paketa za vrijeme izrade projekta. Izvršavanjem naredbe putem naredbene linije (*eng. command line*) programeri mogu dohvatiti PHP programski paket napisan od treće strane koji vrši određenu funkciju. Upravo mogućnošću korištenja tuđih paketa otvorio se prostor za stvaranje novih razvojnih okvira koji trenutno vladaju PHP zajednicom.

2.1.3. Razvojni okviri

Zajednica koja okružuje PHP veoma je dinamična te se pojave novih razvojnih okvira, kao i generalno aplikacija otvorenog kôda (*eng. open source*) koje pružaju različite funkcionalnosti, događaju na dnevnoj razini. Glavni pokretač tolike količine razvojnih okvira i rješenja jest dostupnost PHP razvojnog okruženja.

Ipak određeni razvojni okviri su se isprofilirali te zahvatili dobar dio razvojne zajednice, svaki nudivši određenu distinkciju koja je presudan faktor u odabiru razvojnog okvira. Sredinom 2000.-ih tržištem su dominirali *CodeIgniter*, *CakePHP* i *ZendFramework*. U navedenom periodu PHP je svojim opsegom i kvalitetom i dalje bio okarakteriziran kao primitivni jezik, te su navedeni razvojni okviri bili među začetnicima paradigme objektno-orijentiranog razvoja u PHP-u, najčešće implementirajući *Model-View-Controller* arhitekturu.

Napretkom PHP-a, dolaskom verzije 5.3.0. koja uvodi niz novosti među kojima su i imenski prostori (eng. *namespace*) u PHP zajednicu, te dolaskom spomenutog *Composera* - na tržištu se počinju pojavljivati i alternativni razvojni okviri među kojima su i neki koji danas dominiraju razvojnom zajednicom. Tako se pojavljuje *Yii* koji uvodi konzolarne aplikacije koje omogućuju generiranje raznih segmenata PHP kôda putem naredbene linije[3], te primjerice *Phalcon* koji kao svoju primarnu prednost navodi da je pisan u programskom jeziku C, isporučen kao PHP

[3] Yii framework documentation: automatic code generation. URL: <http://www.yiiframework.com/doc/guide/1.1/en/topics.gii> (12.09.2016.)

ekstenzija, te time iznimno brz[4]. Ali najveći zamah uzimaju *Symfony* i *Laravel* koje bismo mogli nazvati "razvojnim okvirima nove generacije PHP-a".

Symfony se smatra velikim bratom *Laravela* što potvrđuje i sama činjenica da *Laravel* koristi određene komponente *Symfonyja*[5]. Korištenje *Symfonyja* nedvojbeno više promovira se pisanje kvalitetnog kôda i potpuno razumijevanje programskih paradigmi, a i sam razvojni okvir razlikuje se u određenim pristupima koje neke skupine programera preferiraju. Poput *Doctrine Object-relational-mapping* (ORM) koji implementira *Data mapper* za razliku od *Laravelovog Eloquent* ORM-a i *Active mapper* paradigme.

Zato i ne čudi činjenica da u popularnosti dominira mlađi brat *Laravel* koji promovira brzu i efikasnu izradu web aplikacija. Osobine kojima često karakteriziraju programere PHP aplikacija su površno razumijevanje i *get things done* pristup radu što Laravelu donosi simpatije zajednice.

U posljednje vrijeme pojavljuju se i mikro razvojni okviri kao podrška sve popularnijem *single page applications* (*SPA*) konceptu. Njihov fokus je mala temelja baza kôda i pružanje funkcionalnosti za razvoj *RESTful* aplikacija, skraćeno od *Representational State Transfer*. Među popularnijima su *Lumen* od autora okriva *Laravel, Silex* od autora *Symfonyja,* te *Slim*.

2.2. Proces izrade

Administrativno sučelje izrađeno je kao podrška korisničkom sučelju i uvelike se razlikuje u pristupu izrade. Detaljnije o konceptu iza korisničkog sučelja može se pročitati u poglavlju *3. Front-end sučelje*, ali u samom temelju korisničko sučelje izrađeno je kao *aplikacija jednog dokumenta* (en. *Single Page Application*, skraćeno SPA) koja intenzivno koristi *XMLHttpRequest* API. Administrativno sučelje temelji se na suprotnom principu gdje svaka radnja uzima jedan zahtjev za obradu. Takav pristup rezultira neznatno sporijim i tradicionalnijim radom aplikacije, ali za potrebe administracije dovoljnim, te bi svaki napredniji oblik izrade bio samo nepotrebno korištenje resursa.

[4] Phalcon project. URL: <https://phalconphp.com/en/> (12.09.2016.)
[5] Laravel: projects using Symfony. URL: <http://symfony.com/projects/laravel> (12.09.2016.)

2.2.1. Baza podataka

Za potrebe Sustava korištena je *MySQL* baza podataka koja je ujedino i najpopularnija baza podataka u kombinaciji s PHP-om, te sastavni dio LAMP (Linux, Apache, MySQL, PHP) razvojnog okruženja[6].

Koristeći *Artisan*, *Laravelov* API za konzolarne naredbe, moguće je generirati entitet (Model) i odgovarajuću migraciju za bazu podataka. Migracije su nadogradnja na *Laravel Schema builder* apstrakciju podatkovnog sloja aplikacije koje omogućuju jednostavno postavljanje i ažuriranje strukture baze podataka. Konkretno migracije su PHP klase u kojima se unutar definiranih metoda opisuje struktura baze podataka bez potrebe za pisanjem SQL upita na bazi.

Artisan pruža mogućnost izrade Modela i odgovarajuće migracije u jednoj konzolarnoj naredbi, čime posao programera preostaje samo opisati strukturu migracije i relacije modela. Na primjeru izrade modela Kolegija, potrebno bi bilo izvršiti sljedeću naredbu:

```
> php artisan make:model Course --migration
```

Što bi rezultiralo automatskim generiranjem dvaju datoteka: *app/Course.php* i *database/migrations/2016_07_12_0643101_create_courses_table.php.* Prva datoteka predstavlja stvoreni Model u aplikaciji, te iza nje ne postoje nikakva skrivena pravila, dok druga datoteka sa sobom nalaže nekoliko konvencija.

Imenovanje migracija je automatizirani proces, a samo migriranje vrši se prema vremenskom žigu datoteke (timestamp) koji se nalazi kao prefiks imena datoteke. Vremenski žig prvi je detalj koji je važan kod *Laravela* jer definira poredak izvršavanja migracija. Ako migracija sadrži stupac sa stranim ključem koji se referira na tablicu u bazi za koju još nije izvršena migracija, doći će do greške prilikom izvršavanja upita na bazu. Zato je prilikom korištenja migracija uvijek važno pratiti poredak samih datoteka[7]. Nadalje, *Artisan* generira i PHP kôd unutar same datoteke kako bi omogućio brži razvoj, pa time i pretpostavlja naziv same tablice na temelju naziva modela, prateći dogovoreni *Laravel* standard. Time će se za model *Course* generirati

[6] What is LAMP. URL <http://searchenterpriselinux.techtarget.com/definition/LAMP> (12.09.2016.)
[7] Laravel documentation: migrations. URL: <https://laravel.com/docs/5.3/migrations> (12.09.2016.)

migracija koja će pretpostaviti da se odgovarajuća tablica u bazi naziva kao i istoimeni model, u množini, pisan svim malim slovima, što će rezultirati nazivom tablice *courses.* Mehanizam odlučivanja o nazivu tablice je ipak malo kompleksniji, a unutar *Artisan* naredbe moguće je i definirati naziv tablice. Međutim ako se programer odluči za poštovanje *Laravel* standarda, mnoge *skrivene* funkcionalnosti *Laravela* će raditi bez ikakvog dodatnog namještanja (eng. *out of the box*)[8].

Još jedna važna napomena prilikom izrade migracija odnosi se na pivot tablice gdje je standard imenovanja obavezan u pogledu da tablica mora nositi nazive oba Modela koje povezuje, u jednini, odvojenih povlakom _ i napisanih abecednim redom, što bi se na primjeru modela *User* i *Course* glasilo *timestamp_create_course_user_table.php*[9].

Koristeći *Artisan* za potrebe *Sustava za evaluaciju kolegija* izrađeni su sljedeći Modeli i odgovarajuće tablice u bazi, te pivot tablice koje ih povezuju.

`Course`	`Page`	`Qset`
`Question`	`User`	`Semester`

2.2.2 Kontroler

Uz modele koji su sastavni dio *Model-View-Controller* arhitekture aplikacije o kojima je pisano u prethodnom poglavlju, važan element čine i kontroleri. Zadaća kontrolera je kontrola tijeka informacija, te zaprimanje i proslijeđivanje podataka. U izradi *Sustava za evaluaciju kolegija* izrađena je jednak broj kontrolera koliko je i korištenih Modela unutar aplikacije, s iznimkom jednog dodatnog kontrolera za upravljanje podacima Korisničkog dijela Sustava.

Kontrolere je moguće izraditi jednako kao i Modele, korištenjem *Laravel Artisana* i konzolarnih naredbi. Sintaksta za izradu osnovnog kontrolera na primjeru Course modela glasi:

```
> php artisan make:controller CourseController
```

[8] Ibid
[9] Ibid

Čime će se generirati datoteka *app/http/controllers/CourseController.php*. Dodatno moguće je proslijediti i modifikator naredbe (*eng. flag*) kako bi se prilikom generiranja datoteke postavile i sve potrebne funkcije za kontroler resursa (*eng. resource controller*) kakav se koristi kod *RESTful* aplikacija. Proslijeđivanjem modifikatora `--resource` na kraju naredbe, generator će stvoriti i slijedeće metode unutar `CourseController` klase[10]:

Tablica 1: Prikaz generiranih metoda korištenjem `--resource` modifikatora naredbe

Naziv	**HTTP metoda**	**Opis radnje**
`index()`	GET, HEAD	Prikaz svih resursa
`create()`	GET, HEAD	Kreiranje novog resursa
`store()`	POST	Pohranjivanje novog resursa
`show()`	GET, HEAD	Prikaz pojedinačnog resursa
`edit()`	GET, HEAD	Uređivanje pojedinačnog resursa
`update()`	PUT, PATCH	Ažuriranje pojedinačnog resursa
`destroy()`	DELETE	Brisanje pojedinačnog resursa

Zadatak kontrolera je zaprimanje proslijeđenih informacija, odlučivanje o daljnjim radnjama i proslijeđivanje podataka na *View* i *Model* komponente. Dakle kontroleri su točke poveznice ukupne aplikacije te trebaju biti dostupne putem definiranih adresa.

Mapa adresa i odgovarajućih kontrolera nalazi se u datoteci *app/http/routes.php* gdje je moguće korištenjem *Router* klase i njegovih metoda jasno mapirati određene HTTP metode, obrasce adresa i još niz mogućnosti, prema odgovarajućim kontrolerima i metodama unutar kontrolera. Tako primjerice za usmjeravanje korisnika s adrese /korisnici/ prema `index()` metodi `CourseControllera` potrebno je zapisati sljedeći kôd u navedenu datoteku.

```
Route::get('/korisnici', 'CourseController@index');
```

[10] Laravel documentation: Console commands. URL: <https://laravel.com/docs/5.3/artisan> (12.09.2016.)

2.2.3. View

Na kraju, potrebno je prikazati određenu datoteku korisniku. Odabir datoteke prepušten je kontroleru, dok sam prikaz informacije, bilo varijabli ili statičnog sadržaja preostaje na datoteci unutar */resource/views/* mape. Korištenjem PHP-a moguće je na lagan način unutar samog HTML-a učitati sadržaj PHP varijable otvaranjem linije PHP-a, primjerice:

```
<h1>Pozdrav, <?php echo $ime; ?></h1>
```

Naravno, moguće je i veće segmente programskog kôda uključiti unutar HTML sadržaja, ali u takvim situacijama kôd postaje nečitak, a često se gubi i načelo podjele odgovornosti (eng. *Separation of Concerns*, skraćeno SoC). Međutim, iteriranje kroz listu korisnika i slične funkcionalnosti su česta potreba pri samom prikazu sadržaja, a kako bi HTML kôd ostao što čišći, Laravel koristi Blade[11] alat za obrađivanje predložaka (eng. *templating engine*). Korištenjem *Bladea* moguće je gore navedeni primjer prikazati na sljedeći način:

```
<h1>Pozdrav, {{ $ime }}</h1>
```

[11] Laravel documentation: Blade templates. URL: <https://laravel.com/docs/5.3/blade> (12.09.2016.)

3. FRONT-END SUČELJE

3.1. Uvod

3.1.1. HTML 5 i promjena paradigme

Osim znamenitih promjena u pristupu radu i razvoju serverskih aplikacija i rješenja, razvoj web aplikacija i web stranica doživio je značajne promjene i na klijentskoj strani. Prve promjene u paradigmi razvoja događaju se dolaskom 5. verzije HTML-a kojom se uvode značajne promjene u percepciji toga što jest HTML i klijentski razvoj.

Osim određenih estetskih izmjena u fundamentu samog HTML-a, poput odmicanja od sintaktičke striktnosti XML-a, te uvođenja novih semantičkiih elemenata <*section*>, <*header*>, <*footer*> i ostalih, u sklopu HTML5 dolaze i razni drugi noviteti razređeni u odvojenim specifikacijama, na primjer *Web forms 2.0 i Web applications 1.0* od kojih će posljednja postati ono što se danas naziva HTML5.[12]

Navedene specifikacije omogućile su ugradnju novih aplikacijskih programskih sučelja, skraćeno API (*eng. Application Programming Interface*). Time se efektivno pojam HTML-a uzdiže od same paradigme označavanja teksta u svrhu prikazivanja u preglednicima, te se usko povezuje s Javascriptom. Mnogi od navedenih noviteta upravo omogućuju funkcionalnosti kontrolirane pomoću Javascripta, a zajedno uklopljene pod nazivom HTML5.

3.1.2. Kratka digresija u svijet Javascripta

Dok se razvijala *Web applications 1.0* specifikacija i ostali noviteti koji će kasnije biti prozvani HTML5, Javascript je kao primarni klijentski programski jezik također doživljavao određenu renesansu. U posljednjih pet godina rapidan razvoj i sve učestalija ažuriranja Javascript specifikacije omogućile su da se spomenuti jezik odmakne od teške stigme koju je doživio početkom 2000-ih godian. U početcima razvoja web stranica, Javascript je služio kao šegrt HTML, namijenjen isključivo za uljepšavanje korisničkog iskustva. Nije nosio nikakvu značajnu dodatnu funkcionalnost, niti je bio presudan faktor u razvoju web stranica i aplikacija. Prva

[12] Pilgrim, Mark: *HTML5 Up and running*. "The past". O'Reilly media: 2010.

pozitivna promjena koja je nagovjestila rast Javascripta bila je sve učestalija uporaba *XMLHttpRequest* API-ja, koji je u kombinaciji s Javascriptom dobio poznatiji naziv *AJAX* (*eng. Asynchronous Javascript and XML)*. Njime se omogućila pozadinska komunikacija između klijentskog i serverskog dijela aplikacije, što je rezultiralo raznim uporabama, od popularnih *chatova* do ranih oblika *SPA* (*eng. Single Page Application*) koje nisu zahtjevale bilo kakva naknadna učitavanja.[13]

Cjelokupni trend razvoja klijentski-orijentiranih aplikacija počeo je doživljavati sve veći zamah, pa se i organizacija zadužena za izradu Javascript specifikacija, ECMA International i njena radna skupina TC39 aktivirala, što je rezultiralo češćim ažuriranjima specifikacije, odnosno standarda zvanog ECMAScripta, u implementaciji poznatijeg kao Javascript. Primjera radi, ECMAScript 3 standard objavljen je 1999. godine, a nasljednik ECMAScript 5 tek deset godina kasnije - 2009. godine. Međutim od početka 2015. godine i odluke radne skupine TC39, novi ECMAScript standard objavljuje se na godišnjoj razini pa tako aktualna verzija ECMAScripta jest broj 6, dok se trenutno izrađuje nacrt za ECMAScript 7[14].

Osim pukih brojki, značajne izmjene nad standardima omogućile su i značajan rast Javascripta kao programskog jezika. U skladu s time, Javascript se sve intenzivnije počeo koristiti za ozbiljnije zadatke. Tako recimo korištenjem *Node.js* izvršnog okruženja, programeri mogu koristiti Javascript za razvoj serverskih aplikacija, dok je NPM (*eng. Node Package Manager*) jedan od najvećih repozitorija programskih paketa (*eng. library*) za Javascript programski jezik[15]. Upravo koristeći NPM i naredbeni redak (*eng. Command line*) razvijena je i aplikacija koja prati ovaj rad. *Node package manager* korišten je za instalaciju zavisnih programskih paketa o kojima će kasnije biti riječi.

3.1.3. Raslojavanje aplikacija na komponente

Razvojem prethodno spomenutih tehnologija otvorila se mogućnost razvoja kompleksnijih aplikacija koristeći Javascript i nove značajke HTML-a. Jedan od značajnijih aktualnih trendova postalo je raslojavanje aplikacija na komponente. Komponentama se smatraju neovisni blokovi

[13] Rauschmayer, Axel. *Speaking JavaScript: An In-Depth Guide for Programmers*, poglavlje 6. O'Reilly media.
[14] Ibid, poglavlje 5.
[15] Understanding NPM. URL: <https://unpm.nodesource.com/> (12.09.2016.)

programske logike i sučelja koji se mogu primjenjivati u web aplikacijama. Trenutno se koncept komponenti razvija u sklopu *Web components* specifikacije[16], ali mnogi Javascript razvojni okviri (*eng. framework*) već implementiraju temeljnu ideju komponenti. Način implementacije komponenti pak zavisi do razvojnog okvira. Popularni razvojni okvir *React* koristi samo logiku iza web komponenti, dok *Googleov* projekt *Polymer* implementira samu *Web components* specifikaciju za korištenje komponenti[17].

Dodatna karakteristika komponenti jest i tijek podataka. Koncept komponenti temelji se na ideji da komponente kao takve djeluju neovisno jedna od druge, te raspolažu samo nužno potrebnim podacima za njen rad.

3.2. Vizualne komponente i korisničko sučelje

3.2.1. Progresivno unaprijeđenje

Progresivno unaprijeđenje (*eng. Progressive enhancement*) paradigma je prema kojoj se korisničko sučelje, bilo to web stranica ili web aplikacija, razvija tako što se primarno fokusira na mobilnog korisnika. Time se automatski postiže popularan trend *responzivnog web dizajna* (*eng. Responsive web design*, skraćeno RWD) gdje se jedno sučelje koristi za sve vrste uređaja. Rastom mogućnosti preglednika i uređaja, omogućuju se i dodatne funkcionalnosti sučelja. Suprotni pristup razvoju mogao bi se nazvati *dostojna degradacija* od engleskog pojma *graceful degradation* gdje se mogućnosti sučelja umanjuju prema mogućnostima preglednika i uređaja.
Upravo na principu progresivnog unaprijeđenja temelji se i korisničko sučelje *Sustava za evaluaciju kolegija*. Sustav koristi *Bootstrap* razvojni okvir i popis prijelomnih točaka uređaja (*eng. breakpoints*) prema kojima se sučelje prilagođava. Prijelomne točke uređaja su numeričke vrijednosti izražena u pikselima koje označavaju širinu ekrana, a koristeći CSS3 *media query* konstrukt enkapsuliraju niz CSS pravila koja vrijede samo u slučaju da uređaj svojom širinom zadovoljava prijelomnu točku.

U slučaju korištenje paradigme progresivnog unaprijeđenja, pristup razvoju je sljedeći:

[16] W3C Web components. URL: <https://www.w3.org/standards/techs/components#w3c_all> (12.09.2016.)
[17] Vue.js documentation: Comparison with Other Frameworks. URL: <http://vuejs.org/guide/comparison.html> (12.09.2016.)

```
/* Pravilo koje vrijedi za <300px uredaje */
.selektor { width: 100%; }

/* Pravilo koje vrijedi za [300, 600> uredaje */
@media (min-width: 300px) {
	.selektor { width:50%; }
}

/* Pravilo koje vrijedi za 599px> uredaje */
@media (min-width: 600px) {
	.selektor { width:33%; }
}
```

Suprotno, paradigma dostojne degradacija implementira sljedeći princip:

```
/* Pravilo koje vrijedi za >599px uredaje */
.selektor { width: 33%; }

/* Pravilo koje vrijedi za <300, 600] uredaje */
@media (max-width: 600px) {
	.selektor { width:50%; }
}

/* Pravilo koje vrijedi za <300px uredaje */
@media (max-width: 300px) {
	.selektor { width:33%; }
}
```

3.2.2. CSS predobrađivači

Pri razvoju većih web aplikacija ili stranica potrebno je razlomiti CSS u više cjelina kako bi se lakše održavao kôd. Za osnovne potrebe modularizacije CSS-a dovoljna je standardna sintaksa pomoću koje se može uključiti niz datoteka u glavnu CSS datoteku.

```
@import ‘dizajn.css’;
@import ‘stranica/kontakt.css’;
/* ostatak CSS-a */
```

Međutim pri razvoju većih projekata *import* konstrukt nije dovoljan za svladavanje aplikacije. Ti problemi posebice se iskazuju u projektima na kojima radi više osoba. Kako bi sve osobe uključene u razvoj držale isti standard, svaka osoba trebala bi pohraniti niz vrijednosti u svojoj glavi jer CSS ne podržava varijable.

U takvim situacijama koriste se CSS predobrađivači (*eng. CSS preprocessor*) od kojih su najpopularniji *Less*[18] i *Sass*[19]. Takvi predobrađivači omogućuju programerima da koriste dodatne funkcionalnosti u CSS datotekama, poput varijabli i funkcija, a koje prije ugrađivanja CSS datoteke u web stranicu prolaze kroz interpreter kako bi se generirao u potpunosti važeći CSS kôd.

Za potrebe *Sustava za evaluaciju kolegija* korišten je *Sass* interpreter, a kôd kojim su označena obilježja CSS-a pisan je pomoću SCSS sintakse.

Korištene su varijable poput *$c-primary* koja označava primarnu boju sučelja te je definirana u na samom početku u sklopu datoteke *_variables.scss* čime je omogućena izmjena ukupnog dizajna sučelja putem samo jedne varijable.

Sass osim varijabli pruža i dodatne mogućnosti, poput petlji i funkcija, čime je razvoj primarne palete boja moguć u samo tri linije kôda:

```
/* generira dvije varijacije primarne boje */
$c-primary: #FF0000;
$c-primary-light: lighten($c-primary, 10);
$c-primary-dark: darken($c-primary, 10);
```

Osim funkcija *lighten()* i *darken()* koje su ugrađene u *Sass*, korišten je *Bourbon* kao dodatni set funkcija dostupnih za *Sass* predobrađivač. Za uključivanje *Bourbona* korišten je *Node package manager* čime je ugradnja *Bourbona* u sustav odrađena jednom naredbenom linijom.

```
> npm install bourbon
```

[18] Less CSS. URL: <http://lesscss.org/> (12.09.2016.)
[19] SASS: Syntactically Awesome Style Sheets. URL: <http://sass-lang.com/> (12.09.2016.)

3.3. Aplikacijski sloj

3.3.1. Single page applications (SPA)

Single page applications (skraćeno SPA), što bi se moglo prevesti kao Aplikacije jednog (web) dokumenta, je paradigma izrade web aplikacija prema kojoj se sva funkcionalnost web aplikacije ili web stranice ugrađuje u jedinstveni web dokument. Primarni razlog takvog pristupa jest fluidnije korisničko iskustvo jer posjetioc ne mora niti jednom ponovno izvršavati potpuno učitavanje stranice. Takav pristup najočitije se manifestira u obliku bržeg otvaranja podstranica, kao i animiranog otvaranja podstranica. Web aplikacija tako ostavlja puno ozbiljniji dojam i više podsjeća na nativne računalne aplikacije.

Velika mana SPA u samim početcima njenog pojavljivanja bila je problematika URI-ja (eng. *Uniform Resource Identifiera*) i korisničkog iskustva kretanja kroz aplikaciju. Kako se funkcionalnost kretanja zasnivala na učitavanju sadržaja koristeći *XMLHttpRequest,* korisnik je često u pogledu adresne trake preglednika ostajao na istoj lokaciji, najčešće naslovnici. Tako recimo i nakon ulaska u tri razine duboku hijerarhiju web stranice, adresna traka bi i dalje kao adresu samo prikazivala *http://example.org/*. Time korisnik nije mogao poslati ili pohraniti poveznicu na određenu podstranicu već je morao pratiti put dolaska do željene informacije. Također, izvorna funkcionalnost navigacije korištenjem povijesti preglednika bila je onemogućena što je značajno štetilo korisničkom iskustvu jer pritiskom na gumb za povratak jedan korak u povijest pregledavanja bi rezultiralo potpunim odlaskom iz željene aplikacije.

Današnje *aplikacije jednog dokumenta* koriste napredne tehnike kako bi u potpunosti isključile navedenu manu. Korištenjem *History API-ja* moguće je kontrolirati korake korisnika kroz aplikaciju, a izmjenom sidrene vrijednosti u adresnoj traci moguće je replicirati kretanja i pohranjivanje specifičnih informacija. Velik dio novih razvojnih okvira (*eng. framework*) koriste gotove paketa za usmjeravanje korisnika u SPA okruženju. Jedan od poznatijih takvih rješenja jest *Director*[20].

[20] Github: Director. URL: <https://github.com/flatiron/director> (12.09.2016.)

3.3.2. Javascript razvojni okviri za aplikacije jednog dokumenta

Osnovnu funkcionalnost učitavanja podstranica korištenjem *XMLHttpRequest* API-ja moguće je postići i bez korištenja dodatnih razvojnih okvira. Moguće je također razviti i vlastitu komponentu za usmjeravanje korisnika unutar aplikacije. Međutim u jednom trenutku dosegne se razina na kojoj vlastita rješenja počinju rezultirati lošim aplikacijama, te se valja prikloniti nekom od gotovih razvojnih okvira koje održavaju i desetci programera.

Postoji veliki izbor razvojnih okvira koji su u čestoj uporabi prilikom izrade web stranica i web aplikacija. Javascript razvojni okviri nisu nužno vezani uz pojam *aplikacija jednog dokumenta* već općenito pružaju dodatnu i lakšu funkcionalnost prilikom razvoja funkcionalnosti na klijentskoj strani. Vjerojatno najpoznatiji razvojni okvir današnjice jest *jQuery*. Korištenjem navedenog okvira programeri je dugi niz godina bila olakšana osnovna funkcionalnost Javascripta kroz metode koje pruža. To je na kraju čak rezultiralo programerima koji nisu niti poznavali osnovnu sintaksu Javascripta, odnosno kako postići istu funkcionalnost bez korištenja *jQueryja*.

Kroz godine *jQuery* i dalje ostaje glavni razvojni okvir za osnovne funkcionalnosti koje pruža Javascript programski jezik. Drugi razvojni okviri gube takvu utrku, poput *MooToolsa*[21]. Na tržište izlaze razvojni okviri (*eng. framework*) koji počinju pružati mnogo više od osnovih funkcionalnosti dohvaćanja elemenata iz Document object modela (DOM) ili kraće sintakse pisanja *XMLHttpRequest* zahtjeva.

Okviri poput *Angular.js, Ember.js, React.js, Backbone.js* i *Vue.js* počinju pružati dodatne funkcionalnosti koje omogućuju razvoj kompleksnijih sustava i SPA korištenjem raznih koncepata, pretežito *Model-View-Controller* i *Model-View-Viewmodel* paradigmi. Takvi okviri pretežito se baziraju na prije spomenutom konceptu komponenti, te korištenjem HTML atributa vežu funkcionalnosti uz sâm HTML kôd čime se efektivno isprepliće HTML i Javascript odnosno stvara čvrst odnos između dvoje.

[21] Usage statistics and market share of MooTools for websites. URL: <https://w3techs.com/technologies/details/js-mootools/all/all> (12.09.2016.)

Za razvoj *Sustava za evaluaciju kolegija* od navedenih izabran je *Vue.js*. *Vue.js* prihvaćen je kao Javascript razvojni okvir koji ide "ruku-pod-ruku" s *Laravelom*, PHP razvojnim okvirom koji je također korišten za izradu sustava. Iako su navedeni okviri u potpunosti neovisni jedan o drugome, njihova kombinacija je toliko popularna da je nastao i *fork Laravela* i *Vuea* pod nazivom *Laravue*[22].

3.3.3. Vue.js

Rani razvojni okviri, poput *jQueryja* i *MooToolsa* temelje se na imperativnom pristupu razvoja aplikacija. Ukratko, imperativni pristup nalaže da se sva funkcionalnost aplikacije, koja uključuje i vizualne promjene unutar DOM-a (*eng. Document Object Model*) mora specificirat unutar same aplikacije. Konkretno u pogledu *jQueryja*, svaka izmjena vrijednosti nekog HTML elementa mora se "ručno" izmijeniti. Takav postupak, pogotovo u pogledu većih aplikacija i SPA često postaje zamoran i repetitivan, a također rezultira i velikom količinom programskog kôda čija je isključiva funkcija ažuriranje DOM-a.

Vue.js implementira *Model-View-ViewModel* paradigmu koja omogućuje reaktivno vezanje podataka i *Document Object Modela.* Temeljna zamisao jest da podaci reaktivno diktiraju izgledom DOM-a, dakle izgledom web aplikacije ili web stranice. Svaka izmjena podataka automatski će se rezultirati izmjenom sadržaja DOM-a bez dodatne potrebe za manipulacijom HTML-a. *Vue.js* postiže navedenu funkcionalost vezanjem dodatnih atributa na HTML elemente čime se stvara poveznica između podataka i DOM-a. Najosnovniji oblik vezanja podataka i DOM-a je sljedeći[23]:

```
<!-- HTML -->
<div id=”primjer”>
      <p>Pozdrav {{ ime }}</p>
</div>
```

```
/* Javascript */
var Model = {
      ime: ‘Marko’
};
var ViewModel = new Vue({
      el: ‘#primjer’,
```

[22] Github: Laravue. URL: <https://github.com/laravue/laravue> (12.09.2016.)
[23] Vue.js documentation: Getting started. URL: <http://vuejs.org/guide/#Hello-World> (12.09.2016.)

```
        Data: Model
});
```

Stvaranjem objekta koji predstavlja Model, te instance *Vuea* koja predstavlja ViewModel i prima element vezivanja te Model, sadržaj HTML-a koji predstavlja View automatski će se ažurirati ovisno o vrijednosti u Modelu. Prilikom bilo kakve promjene vrijednosti imena u Modelu, izmjene će se uvijek odraziti i u samom HTML-u bez potrebe za dodatnom manipulacijom DOM-a.

Vue.js omogućuje i dvosmjerno vezivanje čime izmjene u HTML-u (View) automatski ažuriraju vrijednost u Modelu[24]. Navedena funkcionalnost postiže se posebnom `v-model` direktivom. Izmijenjena vrijednost odabranog polja u HTML-u rezultirat će ažuriranjem samog Modela, te reaktivnim ažuriranjem HTML-a na svim potrebnim lokacijama. Primjer direktive i dvosmjernog vezanja (*eng. two-way data binding*) podataka može biti sljedeći:

```
<!-- HTML -->
<div id=”primjer”>
      <p>Pozdrav {{ ime }}</p>
      <input
            type=”text”
            v-model=”ime”
            value=”{{ ime }}”
      >
</div>
```

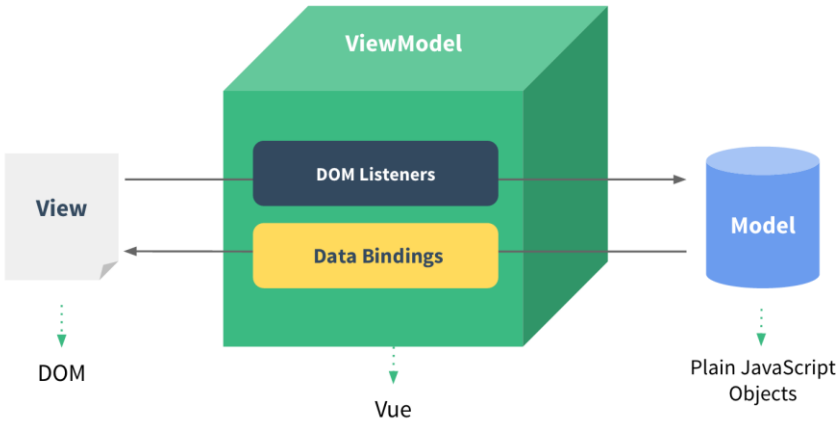
```
/* Javascript */
var Model = {
      ime: ‘Marko’
};
var ViewModel = new Vue({
      el: ‘#primjer’,
      Data: Model
});
```

Navedeni primjer u pregledniku prikazat će se kao paragraf teksta s umetnutom vrijednosti “Pozdrav Marko”, te tekstualnim poljem vrijednosti “Marko”. Izmjenom vrijednosti polja u “Mario” automatski će se ažurirati vrijednost imena u Modelu, te shodno tome i paragraf teksta koji će ispisati “Pozdrav Mario”.

Struktura koja odražava ovakav pristup naziva se Model-View-ViewModel, pri čemu je Model *Javascript* objekt koji sadrži potrebne informacije, View je HTML odnosno *Document Object*

[24] Vue.js documentation: Two-way binding. URL: <http://vuejs.org/guide/#Two-way-Binding> (12.09.2016.)

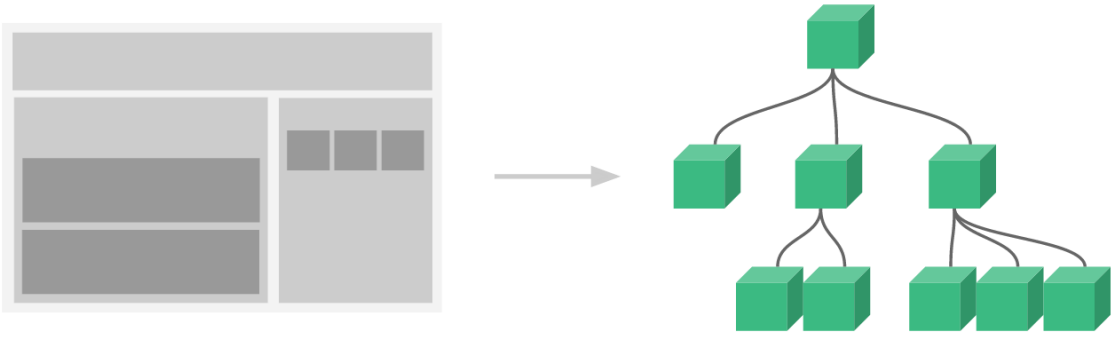
Model, a ViewModel predstavlja instanca razvojnog okvira *Vue.js* koja je zadužena za svu logiku povezivanja[25].



Slika 2: Grafički prikaz Model-View-ViewModel strukture u Vue.js razvojnom okviru

Sustav komponenti

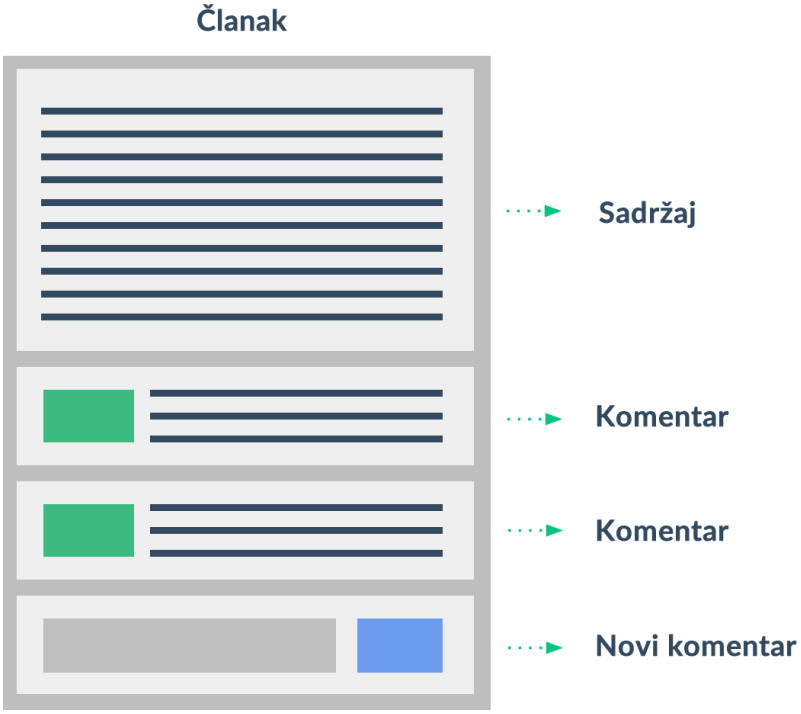
Vue.js razvio se na dva temeljna bloka. Jedan je već pojašnjeno reaktivno vezanje podataka, dok drugi temeljni blok čini sustav komponenti. Sustav komponenti nije isključivo vezan uz *Vue.js* i već je spomenut u poglavlju *3.1.3. Raslojavanje aplikacije na komponente*. Aplikacije se razvijaju kao skup komponenti od kojih svaka prima samo potrebne podatke te implementira samo vlastitu funkcionalnost. Komponente su hijerarhijski posložene.[26]



[25] Vue.js documentation: Overview. URL: <http://vuejs.org/guide/overview.html> (12.09.2016.)
[26] Vue.js documentation: Components. URL: <http://vuejs.org/guide/components.html> (12.09.2016.)

Slika 3: Hijerarhijsko raslojavanje aplikacije na komponente

Konvencija o imenovanju i korištenju komponenti ne postoji, ali se dogovorno temelji na komponentama koje jedna web aplikacija ili stranica može sadržavati. Osnovni oblik sustava komponenti može se prikazati na primjeru članka teksta i komentara koji se prikazuju ispod članka, a učestali primjer su blogovi.



Slika 4: Sustav komponenti na primjeru članka

Članak je komponenta zadužena za prikaz jednog članka teksta i vezanih funkcionalnosti. *Članak* kao komponenta sastoji se od nekoliko pod-komponenti. Neizostavna komponenta je *Sadržaj* koja prikazuje sadržaj članka. Također moguće je da članak sadrži *n* komentara. Komentari se prikazuju u sklopu istoimene komponente *Komentari*. Funkcija komponente *Komentari* jest prikaz jednog komentara. Za rad komponente *Komentari* možemo reći da su potrebni podaci: autor komentara, tekst komentara i datum komentara. Komponenta *Komentari* tako uz primljene podatke može izvršiti svoju funkcionalnost nevezano uz njenu okolinu. Takav pristup omogućuje da se komponenta *Komentari* iskoristi i u drugim segmentima web aplikacije ili web stranice, ne nužno vezano uz članak teksta. Konačno, komponenta *Članak* može

sadržavati pod-komponentu *Novi članak* čija je isključiva funkcija pohranjivanje komentara na temelju jedinstvenog identifikatora (ID) *nekog* entiteta.

Sustav komponenti snažna je funkcionalnost koja omogućuje apstrakciju aplikacije na velik broj zasebnih cjelina koje u ukupnosti rezultiraju web aplikacijom ili web stranicom. Koriste je svi moderni sustavi, a jedan primjer je društvena mreža Facebook i razvojni okvir React koji je razvijen u sklopu tvrtke.

3.4. Razvoj korisničkog sučelja Sustava za evaluaciju kolegija

3.4.1. Gulp za ubrzanje razvojnog okruženja

Prilikom razvoja *Sustava za evaluaciju kolegija* korišten je širok niz tehnologija. Neke od korištenih tehnologija zahtjevaju dodatnu obradu kako bi producirale željeni razultat. Najočitija primjer toga je *Sass* (*3.2.2. CSS predobrađivači*) koji zahtjeva interpretiranje SCSS sintakse kako bi se pretvorila u CSS sintaksu koju preglednici mogu obraditi. Još jedan primjer je *Vue.js* i sustav komponenti. Kako ne bi sve komponente bile u jednoj Javascript datoteci moguće je uključiti HTML i Javascript komponente u treću Javascript datoteku. Ta funkcionalnost nije dostupna sama po sebi, ali ju je moguće postići korištenjem razvojnog alatara *Browserifyja*. Rezultat će u konačnici biti velika Javascript datoteka sa svim komponentama, iako se i tako nešto može izbjeći asinkronim učitavanjem, ali zbog jednostavnosti kôda korištenjem *Browserifyja* moguće je eksterno uključivanje. *Browserify* funkcionira na sličnom principu kao i *Sass* interpreter te je komandu potrebno izvršiti putem naredbene linije (*eng. command line*).[27]

Gulp[28] je alat za izvršavanje vremenski zahtjevnih repetitivnih zadataka poput interpretiranja Sass datoka u CSS, minimizacije kôda za produkciju, generiranja ikona metodom dostojne degradacije (.svg, .png i .jpg) i sličih repetitivnih zadataka. Instalacijom *Gulpa* koristeći NPM (*Node package manager*) može se koristiti naredba *gulp-watch.* Ta naredba izvršavat će sve naredbe zapisane u *gulp.js* datoteci unutar projekta.

[27] Embrey, Blake. *Introduction to Browserify*. URL: <http://blakeembrey.com/articles/2013/09/introduction-to-browserify/> (12.09.2016.)
[28] Gulp.js. URL: <http://gulpjs.com/> (12.09.2016.)

Pošto *Sustav za evaluaciju kolegija* koristi *Laravel* razvojni okvir, moguće je koristiti i *Elixir*, *Laravelov* API za jednostavno izvršavanje *Gulp* naredbi[29]. *Gulp.js* datoteka u sklopu Sustava sljedećeg je sadržaja:

```
/* gulp.js */
var elixir = require('laravel-elixir');
elixir(function(mix) {
    mix.sass('global.scss', 'public/ui/css');
});

require('laravel-elixir-vueify');
elixir(function(mix) {
    mix.browserify('app.js', 'public/ui/js');
}
```

Izvršavanjem naredbe *gulp-watch* razvoj Sustava uvelike je olakšan. Svakom izmjenom *global.scss* datoteke, ili bilo koje pod-datoteka uključene u nju, *Gulp* će izvršiti naredbu za interpretiranje *Sass* datoteka i konačni CSS pohraniti u mapu *public/ui/css.*

Svakom izmjenom *app.js* datoteke ili bilo koje pod-datoteka uključene u nju, *Gulp* će izvršiti *browserify* naredbe koja će interpretirati sve *Javascript* datoteke, te konačnu *Javascript* datoteku pohraniti u mapu *public/ui/css*.

3.4.2. Vueify i Vue.js komponente na jednom mjestu

Koristeći *Browserify* i dodatak za podršku .vue datoteka imena *Vueify*[30] sustav komponenti *Sustava za evaluaciju kolegija* odvojen je u zasebne .vue datoteke.

Koristeći *Vue.js* izrađene su komponente za svaku odvojenu funkcionalnost Sustava. Tako je svaka komponenta odvojena u zasebnu datoteku koja nosi ime komponente, a sam sadržaj odvojen je na dvije cjeline.

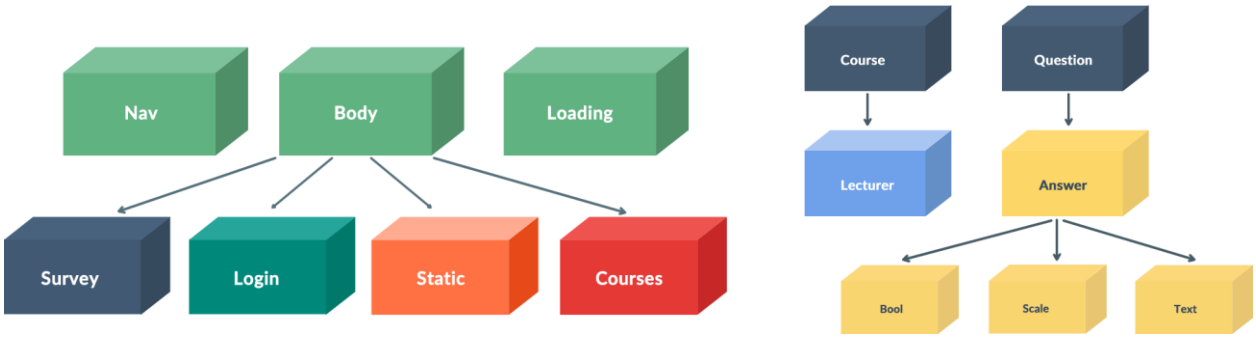
[29] Laravel documentation: Elixir. URL: <https://laravel.com/docs/5.3/elixir> (12.09.2016.)
[30] Github: Vueify. URL: <https://github.com/vuejs/vueify> (12.09.2016.)

Prva cjelina predstavlja HTML kôd komponente i *Vue.js* veze unutar same komponente koje povezuju Model.

Druga cjelina predstavlja odgovarajući *Javascript* segment komponente. Unutar nje navedena je funkcionalnost same komponente.

Važno je za napomenuti kako svaka komponente koristi `Vue.component()` metodu koja stvara zasebnu instancu *Vuea* čime je svaka komponenta *de facto* neovisna instanca Model-View-ViewModel paradigme. Dijeljenje podataka između *roditelja* i *djeteta* komponente postiže se korištenjem svojstava (*eng. properites*), unutar *Vuea* nazvanih "props". Svojstva komponente navode informacije koje komponenta može zaprimiti u obliku HTML atributa, a potrebni su za rad same komponente. Tako recimo komponenta *Članak* može sadržavati pod-komponentu *Komentar* kojoj će proslijediti vrijednosti vlastitog Modela pod ključem *komentar*. Nadalje, sve čime se bavi komponenta *Komentar* jest prikazivanjem samog komentara na temelju svojstva *komentar*. Podrijetlo tog svojstva ne ulazi u jurisdikciju same komponente.

Takvom hijerarhijskom strukturom komponenti postignut je završni oblik *Sustava za evaluaciju kolegija*. Grafički gledano Sustav se može prikazati na sljedeći način:



Slika 5: Grafički prikaz hijerarhije komponenti u Sustavu za evaluaciju kolegija

Komunikacija između komponenti funkcionira u jednom smjeru. Ideja iza Sustava jest da pod-komponenta ne smije biti zadužena za izmjenu vrijednosti u Modelu *roditeljske* komponente. Sve što *dijete* komponenta može jest zaprimiti vrijednosti putem svojstava.

Ipak, u određenim situacijama *dijete* komponenta sadrži funkcionalnost koja zahtjeva izmjene koje nisu isključivo unutar same komponente već i na *roditeljskoj* komponenti, pa i globalno u sklopu cijelog Sustava. Takva komunikacija omogućena je uz pomoć dispečerskog sustava događaja. U slučaju promjene na globalnoj razini, koristeći *Event Javascript* API komponenta će okinuti *događaj* (*eng. event*) koji mogu pratiti roditeljske komponente. Kada roditeljska komponenta uhvati okinuti *događaj* na njoj je da obradi događaj, dohvati okinutu promjenu i izvede potrebnu logiku na vlastitoj razini. Ta izmjena u konačnici se može propagirati nazad niz lanac do same *dijete* komponente koja je i okinula događaj, ali važno je za napomenuti kako sama komponenta nema mogućnost izmjene Modela izvan svog dosega.[31]

3.4.3. SPA i povezivanje komponenti

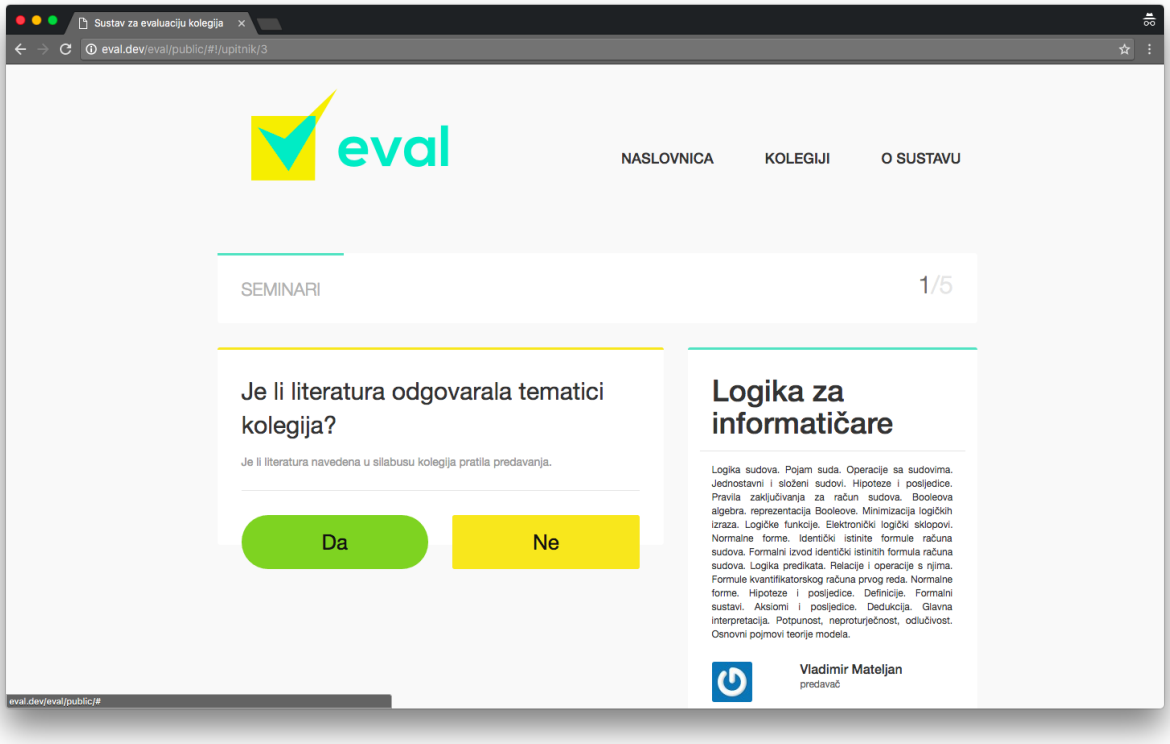
Kao što je je već spomenuto u *3.3.1. Single page applications (SPA)* povezivanje i učitavanje željenih informacija u *aplikacijama jednog dokumenta* prethodno je bio velik problem. Neki razvojni okviri poput *Angular.js* u samoj jezgri omogućuju funkcionalnost povezivanja komponenti i dinamičkog prikaza sadržaja. Međutim *Vue* je u temelju izrađen kao razvojni okvir koji ne pretpostavlja izradu SPA, te stoga ne sadrži nikakvu funkcionalnost u jezgri vezanu uz povezivanje koje zahtijevaju SPA.[32]

Za funkcionalnost povezivanja korišten je *Vue-router*[33] paket koji na jednostavan način omogućuje povezivanje sadržaja. Funkcionira na principu objekta koji mapira definirane adrese s odabranim komponentama. Funkcionalnost dohvaćanja samog sadržaja preostaje pak na samoj komponenti i izlazi izvan opsega *Vue-router* paketa, radi čega je potrebno uvesti još jedan, završni paket zadužen za učitavanje i obradu *XMLHTTP* zahtjeva. Paket korišten za spomenutu funkcionalnost jest *Vue-resource* koji kroz jednostavan niz metoda omogućuju asinkrono

[31] Vue.js documentation: Components. URL: <http://vuejs.org/guide/components.html> (12.09.2016.)
[32] Vue.js documentation: Comparison with Other Frameworks. URL: <http://vuejs.org/guide/comparison.html> (12.09.2016.)
[33] Github: Vue-router. URL: <https://github.com/vuejs/vue-router> (12.09.2016.)

učitavanje sadržaja (AJAX), te dodatno podržava *Javascript Promise*[34] za fluidniji asinkroni rad aplikacije.

Spajanjem svih komponenti i korištenjem *Vue.js* razvojnog okvira na kraju je postignuta funkcionalnost korisničkog sučelja *Sustava za evaluaciju kolegija.*



Slika 6: Sustav za evaluaciju kvalitete kolegija, anketni dio i da/ne pitanje.

[34] Javascript Promise object. URL: <https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Promise> (12.09.2016.)

4. ZAKLJUČAK

U konačnici, korištenjem niza tehnologija razvijen je *Sustav za evaluaciju kolegija* do razine potpune funkcionalnosti. Razvoj Sustava uključivao je širok niz tehnologija i još veću količinu dodatnih paketa što govori o gotovoj nemogućnosti i opsegu potrebnih znanja za razvoj jedne moderne aplikacije od samog početka. Na svu sreću korištenjem otvorenog kôda i znanja ostalih programera moguć je brz i efikasan način razvoja novih web stranica i web aplikacija.

Ovaj rad također prikazuje i značajan napredak web tehnologija koji se dogodio u posljednjih pet godina, ali i trend kojim se tehnologije dalje razvijaju, posebice unutar PHP i Javascript zajednica. Trend pokazuje sve veće približavanje nativnim aplikacijama i sve manje razlike u mogućnostima i funkcionalnosti web aplikacija. Iz svega viđenog, slobodno se može reći da je razvoj web aplikacija u sve većem zamahu, te da je u budućnosti realno vidjeti sve veći broj SaaS (eng. *Software as a Service*) aplikacija koje su dostupne primarno putem Interneta, kao najrasprostranjenijeg i najdostupnijeg medija.

5. LITERATURA

5.1. Knjige

1. Lerdorf, Rasmus; MacIntyre Peter; Tatroe, Kevin. *Programming PHP, 3rd Edition.* O'Reilly media: 2013.
2. Pilgrim, Mark: *HTML5 Up and Running*. O'Reilly media: 2010.
3. Rauschmayer, Axel. *Speaking JavaScript: An In-Depth Guide for Programmers*. O'Reilly media: 2014.

5.2. Poveznice

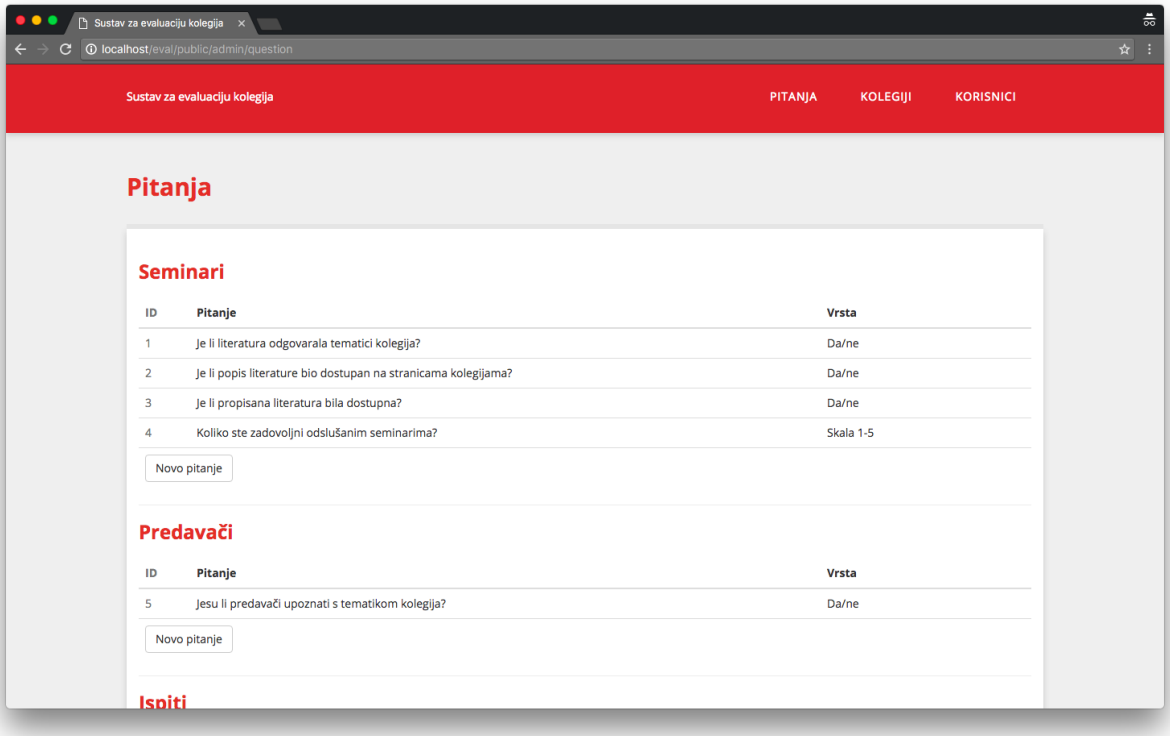
1. Embrey, Blake. Introduction to Browserify. URL: <http://blakeembrey.com/articles/2013/09/introduction-to-browserify> (12.09.2016.)
2. Github: Vue-router. URL: <https://github.com/vuejs/vue-router> (12.09.2016.)
3. Github: Vueify. URL: <https://github.com/vuejs/vueify> (12.09.2016.)
4. Github: Laravue. URL: <https://github.com/laravue/laravue> (12.09.2016.)
5. Github: Director. URL: <https://github.com/flatiron/director> (12.09.2016.)
6. Gulp.js. URL: <http://gulpjs.com> (12.09.2016.)
7. Javascript Promise object. URL: <https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Promise> (12.09.2016.)
8. Laravel documentation: Blade templates. URL: <https://laravel.com/docs/5.3/blade> (12.09.2016.)
9. Laravel documentation: Console commands. URL: <https://laravel.com/docs/5.3/artisan> (12.09.2016.)
10. Laravel documentation: Elixir. URL: <https://laravel.com/docs/5.3/elixir> (12.09.2016.)
11. Laravel documentation: Migrations. URL: <https://laravel.com/docs/5.3/migrations> (12.09.2016.)
12. Laravel: projects using Symfony. URL: <http://symfony.com/projects/laravel> (12.09.2016.)
13. Less CSS. URL: <http://lesscss.org> (12.09.2016.)
14. Phalcon project, URL: <https://phalconphp.com/en> (12.09.2016.)

15. PHP manual: History of PHP. URL: <http://php.net/manual/en/history.php.php> (12.09.2016.)
16. PHP: A fractal of bad design. URL: <http://eev.ee/blog/2012/04/09/php-a-fractal-of-bad-design> (12.09.2016.)
17. SASS: Syntactically Awesome Style Sheets. URL: <http://sass-lang.com> (12.09.2016.)
18. Understanding NPM. URL: <https://unpm.nodesource.com> (12.09.2016.)
19. Usage statistics and market share of MooTools for websites. URL: <https://w3techs.com/technologies/details/js-mootools/all/all> (12.09.2016.)
20. Vue.js documentation: Comparison with Other Frameworks. URL: <http://vuejs.org/guide/comparison.html> (12.09.2016.)
21. Vue.js documentation: Components. URL: <http://vuejs.org/guide/components.html> (12.09.2016.)
22. Vue.js documentation: Getting started. URL: <http://vuejs.org/guide/#Hello-World> (12.09.2016.)
23. Vue.js documentation: Overview. URL: <http://vuejs.org/guide/overview.html> (12.09.2016.)
24. Vue.js documentation: Two-way binding. URL: <http://vuejs.org/guide/#Two-way-Binding> (12.09.2016.)
25. W3C Web components. URL: <https://www.w3.org/standards/techs/components#w3c_all> (12.09.2016.)
26. What is LAMP. URL <http://searchenterpriselinux.techtarget.com/definition/LAMP> (12.09.2016.)
27. Yii framework documentation: automatic code generation. URL: <http://www.yiiframework.com/doc/guide/1.1/en/topics.gii> (12.09.2016.)

6. PRILOZI

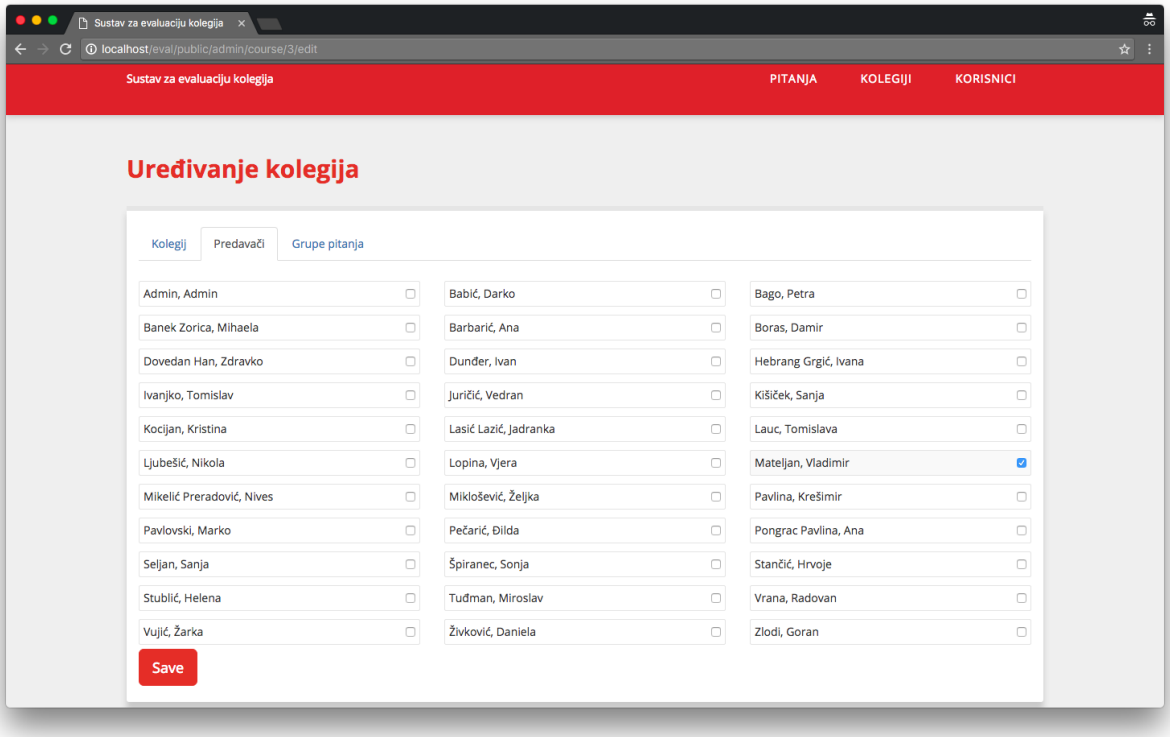
Prilog 1.

Popis grupa pitanja u administrativnom sučelju.



Prilog 2.

Odabir predavača vezanog kolegija u administrativnom sučelju.

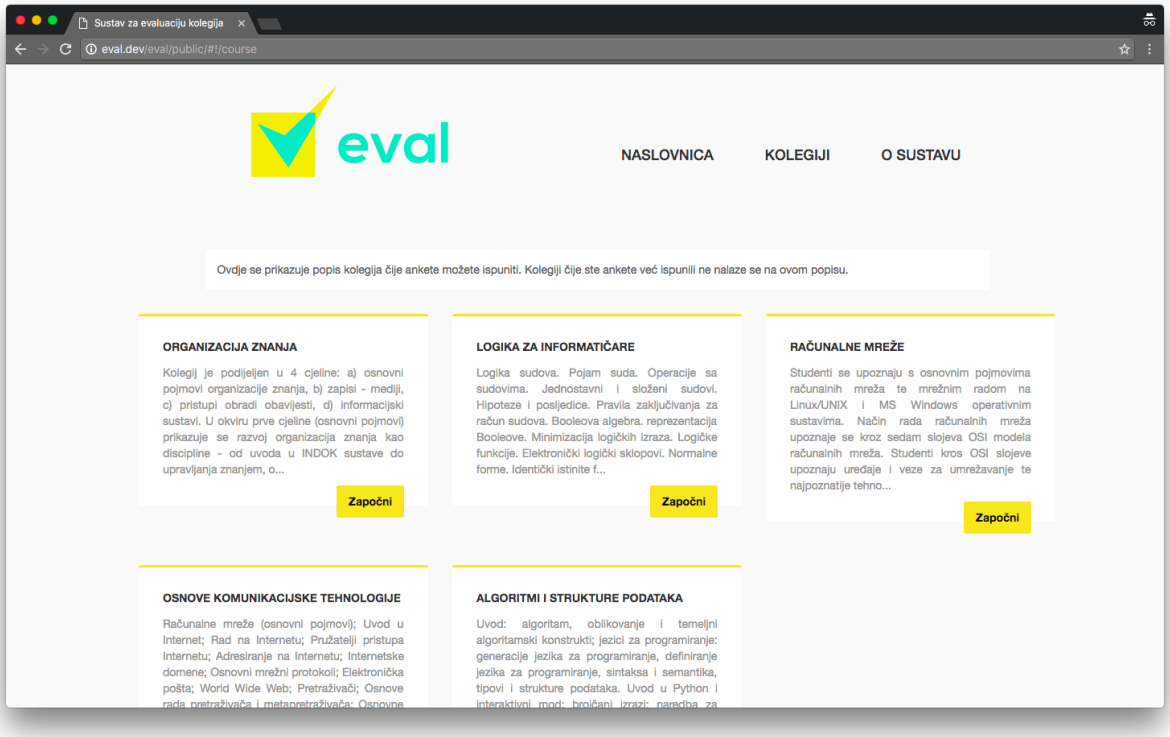


Prilog 3.

Primjer anketnog pitanja u korisničkom sučelju.

Prilog 4.

Popis dostupnih kolegija u korisničkom sučelju.



Prilog 5.

Vidljivi dio korisničkog sučelja pregledavanjem izvornog kôda stranice.