MVC oblikovni obrasci u PHP razvojnom okviru Laravel

Štaba, Ivan

Undergraduate thesis / Završni rad

2016

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University North / Sveučilište Sjever**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:122:737834

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-16**

Repository / Repozitorij:

University North Digital Repository

Sveučilište
Sjever

Završni rad br. 505/MM/2016

MVC oblikovni obrasci u PHP razvojnom okviru Laravel

Ivan Štaba, 4369/601

Varaždin, rujan 2016. godine

Sveučilište
Sjever

Studij

Multimedija, oblikovanje i primjena

Završni rad br. 505/MM/2016

MVC oblikovni obrasci u PHP razvojnom okviru Laravel

Student

Ivan Štaba, 4369/601

Mentor

mr. sc. Vladimir Stanisavljević, dipl. ing.

Varaždin, rujan 2016. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Odjel za multimediju, oblikovanje i primjenu

PRISTUPNIK: Ivan Štaba | MATIČNI BROJ: 4369/601

DATUM: 14.09.2016. | KOLEGIJ: Programski alati 3

NASLOV RADA: MVC oblikovni obrasci u PHP razvojnom okviru Laravel

NASLOV RADA NA ENGL. JEZIKU: MVC design patterns with PHP framework Laravel

MENTOR: mr.sc. Vladimir Stanisavljević | ZVANJE: viši predavač

ČLANOVI POVJERENSTVA:

1. dr.sc. Ladislav Havaš, v. pred. - predsjednik
2. pred. Andrija Bernik, dipl.inf. - član
3. mr.sc. Vladimir Stanisavljević, v.pred. - mentor
4. doc.dr.sc. Dejan Valdec - zamjenski član
5.

Zadatak završnog rada

BROJ: 505/MM/2016

OPIS:

Popularna platforma za razvoj poslužiteljskog dijela web aplikacija uvijek se razmatra kroz razvojni slog poput LAMPa, WAMPa. Kod njih se programski dio izvodi programskim jezikom PHP. Zbog povijesnog razvoja PHP ima prilično neuredno strukturu i u njega su kroz vrijeme dodavane različite programske paradigme i brojne funkcije. Da bi se poboljšala strukturiranost aplikacija napisanih u PHP-u poželjno je koristiti neku naprednu programsku metodologiju poput MVC (od engl. Model-View-Controler) oblikovnih obrazaca. U PHP-u podršku za MVC može se ugraditi primjenom razvojnih okvira poput Laravel-a.

U ovom radu potrebno je:
* opisati osnove xAMP-a i ulogu programskog jezika PHP-a u njemu,
* detaljno obraditi MVC programsku arhitekturu podržanu kroz Laravel razvojni okvir,
* osmisliti i oblikovati web aplikaciju koja koristi Laravel za neku stvarnu primjenu s bazom podataka,
* istražiti dodatne web tehnologije kojima bi se mogle ostvariti dodatna poboljšanja stranica.

Detaljno opisati sve korištene tehnologije i korake u radu potrebne da bi se ostvarilo traženo te detaljno opisati stečena iskustva i postignute rezultate.

ZADATAK URUČEN: 20.09.2016. | POTPIS MENTORA: V. Stanisavljević

REPUBLIKA HRVATSKA 5 SVEUČILIŠTE SJEVER KOPRIVNICA SVEUČILIŠTE SJEVER

Predgovor

Razvoj web aplikacija u današnje vrijeme postao je višestruko zahtjevniji od onog prije 15-tak godina kada internet bazirane aplikacije i usluge preuzimaju dominaciju nad korisničkim tržištem. Razlog tome nije samo napredak u brzini internet veze, mogućnosti web preglednika ili značajni skok u preformansama računala. Veliku ulogu kod napretka u razvojnim tehnologijama ima pitanje sigurnosti, te brzina razvoja same web stranice.

Tu dolazi do potrebe za programskim arhitekturama ili oblikovnim obrascima, te razvojnim okvirima poput Laravela. MVC arhitektura pomaže logički odijeliti prava pristupa podacima i dobra je praksa koja omogućava unificirani pristup prema problemu unutar razvojnog tima, dok raznorazni razvojni okviri znatno skraćuju vrijeme potrebno za razvoj neke aplikacije.

Želio bih ovom prilikom zahvaliti profesoru Stanisavljeviću na predloženoj temi koja mi je poslužila kako bih proširio svoje znanje i vidike unutar struke.

Sažetak

Cilj ovog završnog rada bio je opisati princip rada Model View Controller arhitekture u programskom jeziku PHP. Između ostaloga opisati rad sa alatima (WAMP, Laravel), te upoznati pozadinu, programskog jezika PHP, MVC modela i alata.

Rad se sastoji od sedam poglavlja, u kojem se kroz prva upoznajemo sa poviješću alata koji su potrebni za izvršavanje zadatka. To je u prvom redu WAMP server. Nakon toga se opisuje princip rada Model View Controller modela, počevši od povijesnog pregleda.

Za izvršavanje praktičnog dijela koristi se razvojni okvir Laravel. Laravel je nešto noviji MVC razvojni okvir, dizajniran 2011., a tek je unatrag godinu, dvije stekao popularnost. Koristi vlastiti server (koji se doduše veže na, u ovom slučaju, Apache), što ga čini idealnim za razvoj više aplikacija u isto vrijeme, budući da je svaki direktorij jedna razvojna cjelina. Korištena je verzija 5.2.

U zadnjem dijelu se opisuje, kroz linije koda, odnos između MVC elemenata unutar aplikacije. Svrha zadatka jest napraviti web aplikaciju koja izračunava putne troškove. Pri tome je odijeljen korisnički od administratorskog dijela, te je također moguće dodati novog, mijenjati ili izbrisati postojećeg korisnika.

Sama aplikacija se nalazi na serveru Sveučilišta: http://arwen.velv.hr/~ivstaba/

Ključne riječi:

PHP, WAMP, Model View Controller, MySQL, Laravel, web programiranje, HTML builder, Form builder, Blade, web server, CRUD

Summary

The goal of this final work was to describe the working principle of Model-View-Controller architecture in the programming language PHP. Among other things to describe the work with tools (WAMP, Laravel), and to understand the background of PHP programming language, MVC model and other used tools.

The paper consists of seven chapters in which through first few we get familiarized with history of the tools required to carry out the task. This is primarily a WAMP server. Follows a working principle description of the Model-View-Controller models, starting with the historical review.

Laravel framework was used to develop a practical part. Laravel is newer MVC framework, initially designed in 2011, that gained popularity in the last year or two. It uses it's own server, bound to Apache in this case, making it ideal for development of more applications at the same time, since each directory is one development unit. Version used is 5.2.

The last part describes, through lines of code, the relationship between MVC elements within the application. The purpose of the exercise is to create a web application that calculates travel expenses. User and administrator privileges are separated, and adding new, modifying or deleting existing user is possible with the privileges of the latter.

App itself can be found on the University server: http://arwen.velv.hr/~ivstaba/

Keywords:

PHP, WAMP, Model View Controller, MySQL, Laravel, Web programming, HTML builder, Form builder, Blade, web server, CRUD

Sadržaj

1. Uvod 1
2. WAMPServer – „Windows, Apache, MySQL, PHP“ Server 3
   2.1. PHP: Hypertext Predprocessor 4
      2.1.1. PHP 1.0 i 2.0 6
      2.1.2. PHP 3.0 7
      2.1.3. PHP 4.0 8
      2.1.4 PHP 5.0 8
   2.2. MySQL 10
3. MVC – „Model-View-Controller“ 11
   3.1 Model 12
   3.2 View 12
   3.3 Controller 13
   3.4 Routing 13
   3.5 D-R-Y 13
4. Laravel – PHP MVC razvojni okvir 14
5. Praktična izvedba MVC arhitekture u Laravel razvojnom okviru 17
   5.1 Opis aplikacije 17
   5.2 Alati za izvođenje aplikacije – instalacija i opis 17
   5.3 Model izvođenja aplikacije 18
   6.4 Opis rada i koda aplikacije 19
   5.5 Model i routes 22
   5.6 Controller i middleware 24
   5.7. View 29
6. Zaključak 38
Literatura i linkovi 40
   Fotografije 40
Prilozi 42
   Model 42
   Controller 42
   Middleware 48
   Routes 49
   View 50
   CSS 68

Kratice i akronimi

PHP	Personal Home Page: Hypertext Preprocessor
MySQL	My Structured Query Language
xAMP	Windows, Linux, Macintosh Apache MySQL PHP
CGI	Common Gateway Interface
GNU	GNU's Not Unix!
UNIX	operativni sistem
XAMPP	Cross-platform Apache MySQL PHP Perl
IIS	Internet Information Server
FTP	File Transfer Protocol
OOP	Objektno Orjentirano Programiranje
CSS	Cascade Style Sheet
URL	Uniform Resource Locator
CMS	Content Management System
MVC	Model View Controller
DOS	Disk Operating System
GUI	Graphical User Interface
IDE	Integrated Development Enviroment

1. Uvod

„PHP's design goal from the very beginning is very simple. To solve the common web problem. That's it. People always look for more grandiose motivations, but they really aren't there. It is a simple tool to solve a simple problem. There are way too many overly complex tools out there to solve what is essentially a very simple problem.

And yes, the language itself by some standards is somewhat primitive, but then it was never designed to be a revolutionary language. The amount of academic masturbation has been kept to an absolute minimum, focusing instead on solving real-world problems.“ [1]

Rasmus Lerdorf, 2002.

PHP je danas najpopularniji i najzastupljeniji server-side skriptni jezik na svijetu. To rečeno, od 2010. ipak mu lagano pada popularnost.[1] Zaslugu za prvotno zanimanje programera i web developera za isti leži prvenstveno u tome što je PHP zamišljen kao jezik koji služi svrsi ili drugim riječima, sa čim manje koda čim kvalitetnije i brže doći do rješenja.

Sintaksom je (u početku) sličan C-u i Perlu, no kasnijim razvojem i implementacijom, sve više poprima oblik i filozofiju objektno orjentiranih programskih jezika poput Jave ili C++-a.

PHP je kreiran od strane Rasmusa Lerdorfa 1994. godine, iako je sam razvoj počeo 1993. inspiriran Mosaic web preglednikom [2][3]. 1997. godine Andi Gutmans i Zeev Suraski preuređuju PHP , te im kao podloga služi tzv. Zend engine.

Osim jednostavnosti, PHP je igrom slučaja postao jedan od glavnih aktera u sve većem prihvaćanju open source filozofije. Lerdorf je dijelio okolo CGI[2] pisan u C-u koji je bio okosnica PHP-a, a nerijetko su mu bili vraćani ispravci bugova i nadogradnje. Tako se krug korisnika sve više širio, a samim time i repozitorij skripti jezika. Open source zajednica je bila u to vrijeme daleko popularnija, budući da je web bio u ekstremno uzlaznoj putanji, pa su brzine prijenosa podataka srazmjerno tome rasle. U to vrijeme je Open source software bio gotovo sinonim za GNU [4]. Bio je to svojevrstan otpor, u duhu 90-ih, golemim korporacijama (prvenstveno Microsoft, Adobe, Macromedia) koje su dominirale razvojem računalne tehnologije, te diktirale trendove. A nemali razlog tome je bila i velika cijena razvojnih alata, poput Dreamweavera, Microsoft Visual Studia, Photoshopa koji su bili esencijalni (neki od njih još uvijek jesu) za izvedbu složenijih projekata. GNU zajednica je open source filozofijom poticala razvoj besplatnih zamjena komercijalnih alata (SharpDevelop, Blender, Eclipse, i sl.), te je obično gravitirala oko Linux operativnog sistema. Upravo se preko Linux zajednice PHP prvenstveno širio do ranih 2000-ih (ili bolje rečeno do

[1] http://pypl.github.io/PYPL.html
[2] Common Gateway Interface

izlaska verzije 5 2004.), kada počinju izlaziti stabilne (prije stabilnije) verzije za Microsoft Windows operativni sistem.

Prijelaz (implementacija) na Apple operativni sistem je bio daleko jednostavniji budući da se također radi o UNIXoidu.

Tema ovog rada jest MVC, tj. Model View Controller, što znači da se aplikacija dijeli u smislene dijelove koji međusobno komuniciraju kako bi što sigurnije i efikasnije izvršili radnje prema ili od strane korisnika.

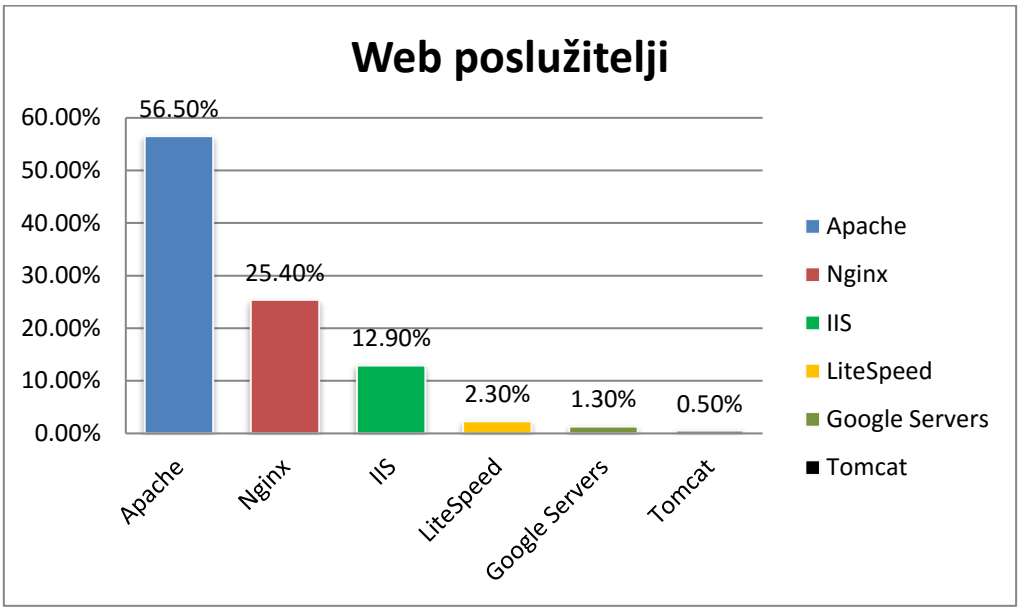
Moj prvi susret sa PHP-om je bio u ranim 2000-im, u verziji 4, no od tada do izrade ovog rada nisam imao neki aktivni kontakt s jezikom, dok o MVC modelu nisam znao ništa. Stoga je ovaj rad bio svojevrstan izazov koji sam prihvatio na prijedlog mentora.

Literatura kojom sam se služio je većinom preko video tutorijala sa YouTubea, PHP dokumentacije i web siteova. Literatura na hrvatskom jeziku je trenutno poprilično zastarjela, a razlog tome ću podrobnije objasniti kasnije u radu.

2. WAMPServer – „Windows, Apache, MySQL, PHP“ Server

WAMP jest serverska platforma za aplikacije koja je zapravo verzija LAMP [5] servera, no za Windows OS okruženje. Također popularan all-in-one server jest XAMPP [6] koji je nešto „teži“ u odnosu na WAMP, budući da sadrži između ostaloga i FTP[3] klijent, pa stoga ne zahtjeva instalaciju dodatnog software-a za prebacivanje sadržaja sa lokalnog računala na web server.

Apache server je najzastupljeniji server na svijetu i koristi se, prema zadnjim pokazateljima, na 56.5% web stranica. Zajedno sa IIS-om[4] čini 70% cjelokupnog udjela web poslužitelja.



Slika 2.1. Izvor: http://w3techs.com/technologies/overview/web_server/all

Apache je open source i besplatan poslužitelj, a prva je verzija izašla 1995. godine. Open source znači da su korisnici u mogućnosti mijenjati kod prema vlastiom nahođenju i potrebama, pa na taj način ispravljati greške, te nadograđivati sam software novim mogućnostima. Web server je program koji koristi HTTP[5] kako bi distribuirala sadržaj web stranica prema korisnicima. Osim samog programa, web serverom također smatramo računala na kojima se programi nalaze. Obično tu još spadaju FTP klijenti, te e-mail servisi [7][8].

Iako gornji graf pokazuje da ukupan broj instaliranih servera koristi Apache u više od 55% slučajeva, broj komercijalnih servera koji ga koriste je dosta manji prema Netcraft.com istraživanju za kolovoz 2015. Tek 37.51% u odnosu na Microsoftov IIS na koji otpada 26.47%,

[3] File Transfer Protocol
[4] Internet Information Services
[5] Hypertext Transfer Protocol

što je ipak impresivna brojka s time da je Microsoft Windows najzastupljeniji operativni sistem. Najveća prednost Apache servera je fleksibilnost, otvoreni kod, te besplatna distribucija, dok što se tiče brzine izvođenja malo zaostaje iza IIS-a i Nginxa. Najveći hendikep su mu neiskorištenost mogućnosti, budući da većina developera koristi tek djelić onog za što je Apache sposoban, te nešto sporije izvođenje radnji. Naime, Apache je process-based server, a ne event-based, što znači da svaka simultana konekcija traži novu nit (thread) kod izvršavanja neke radnje, za razliku od asinkroničnog servera gdje se radnja izvršava u relativno niskom broju threadova [9].

Zadnja stabilna verzija koja dolazi sa WAMP Serverom je 2.4.9.

2.1. PHP: Hypertext Predprocessor

Kao što citat Rasmusa Lerdorfa u uvodu kaže, smisao PHP-a je odraditi posao čim efikasnije. Od svojih skromnih početaka prije 20 godina, razvio se u poprilično moćan i višenamjenski jezik. Pomoću njega je moguće napraviti u rasponu od male, pomoćne skripte do velikih objektno orjentiranih aplikacija.

Neke od prednosti koje krase PHP su ponajprije jednostavnost. Iako je nešto kompliciraniji o Pythona, još uvijek je manje zahtjevan od, primjerice, Rubyja. Također postoji velika PHP zajednica koja pomaže, osim kod daljnjeg razvoja i napretka jezika, početnicima i programerima na raznim razinama znanja, kako bi unaprijedili svoje vještine.

Nažalost hrvatska PHP zajednica nije toliko aktivna. Na nekad popularnoj stranici http://php.com.hr/, zadnji članak je bio objavljen prije 7 godina. Literatura dostupna na hrvatskom jeziku je mahom zastarjela, te kod, sintaksa i prakse navedene u istima su većinom od male koristi.

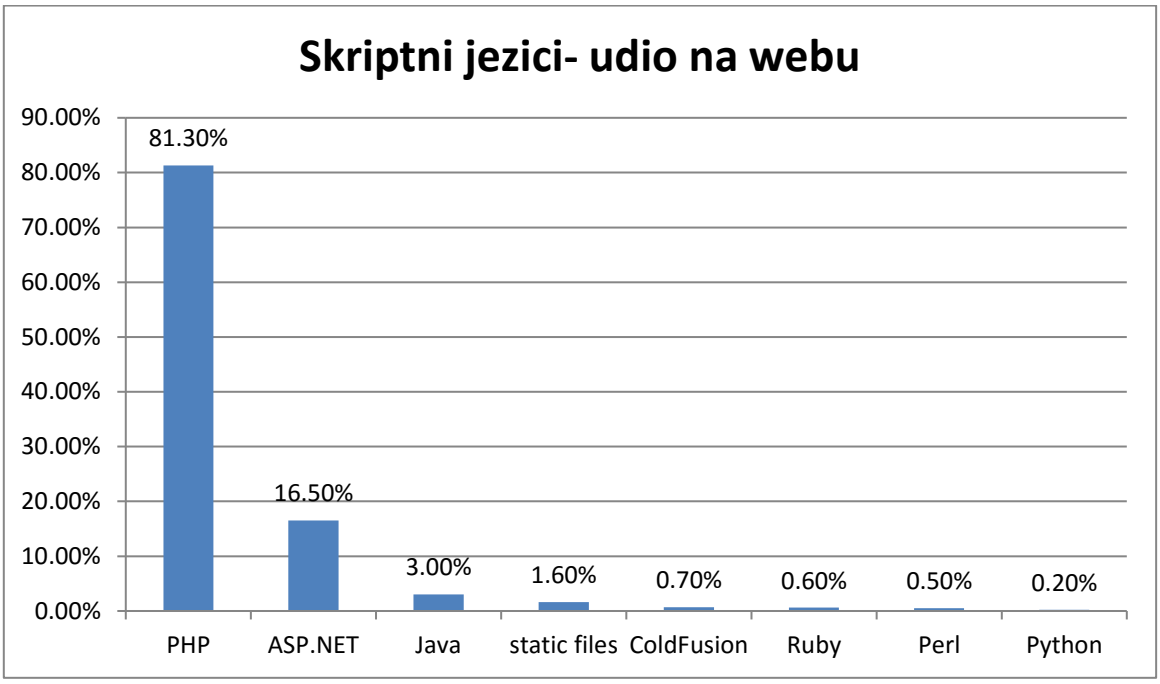
Osim jednostavnosti i pristupačnosti, za PHP postoji i velik broj kvalitetnih alata koji olakšavaju razvoj aplikacija. Preko jednostavnih tekst editora (ConTEXT, JEdit, SublimeText) do IDE[6] razvojnih okolina (NetBeans, Aptana Studio, Eclipse), korisnicima je omogućeno da biraju kako im najbolje odgovara razvijati aplikacije pisane u PHP-u.

PHP također krasi odlika da po potrebi inkorporira dinamiku i sintaksu iz drugih jezika. Primjerice, baznu sintaksu temelji na C-u, dok kasnije postupnom implementacijom Objektno Orjentirane arhitekture posuđuje sintaksu iz C++-a i Jave, a način rada MVC Razvojnog okvira koristi logiku i strukturu svojstvenu Ruby programskom jeziku.

Neke od popularnijih web stranica koje koriste PHP su Facebook (iako koristi svoju inačicu PHP-a, tzv. HPHPc [10]), Wikipedia, Yahoo!, Baidu (najpoznatiji kineski pretraživač), Flickr, Tumblr, Wordpress [11].

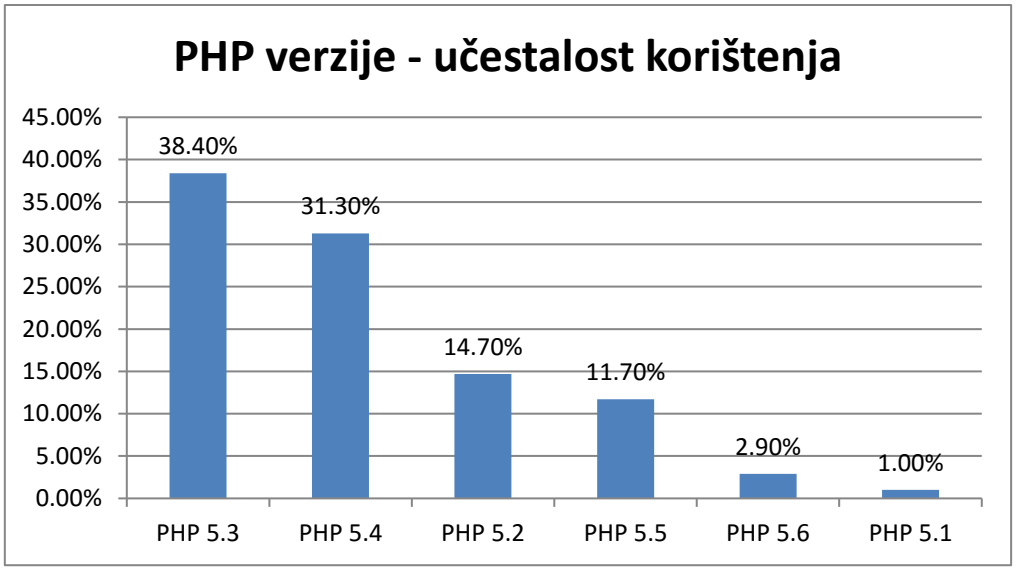
[6] Integrated Development Enviroment

Iako je, što se tiče web stranica sa velikim brojem protoka podataka, u nekoj zlatnoj sredini, velika većina nekomercijalnih i sl. stranica od ukupnog broja postavljenih na web.



Slika 2.2. Izvor: http://w3techs.com/technologies/overview/programming_language/all

No PHP, dakako, nije i bez svojih problema [12], od kojih su neki rješavani, dok se na drugima inzistira. Tu ponajprije mislim na odbacivanje cijele sfere prakticiranja određene arhitekture unutar jezika. Pod time prvenstveno mislim na izbacivanje sintakse oko *mysql_...()* komuniciranja sa MySQL bazom podataka od verzije 5.5. Potonja je, doduše, zamijenjena *mysqli_..()* [13] sintaksom, no promjene su zapravo bile poprilično veće od dodavanja samo jednog slova na kraju upita. Kao razlog se često navode problemi sa sigurnošću, no mislim da se to moglo riješiti na puno elegantniji način od milijuna redaka koda koji su postali gotovo neupotrebljivi. Zbog toga većina web stranica koristi verzije 5.3 i 5.4.



Slika 2.3. Izvor: http://w3techs.com/technologies/details/pl-php/5/all

Krajem prošle godine je izašao PHP 7 koji nastavlja sa tzv. *backward incompatible* trendom. Praksom, koja po mojem mišljenju nije dobra niti praktična. Windows operativni sistemi jedan dio svoje popularnosti i praktičnosti duguju upravo *backward compatible* podrški za velik broj programa. Istu praksu imaju i neki programski jezici poput Jave. S druge strane, uz podosta drugih promjena, uvedena je mogućnost inicijalizacije skalarnih deklaracija (int, float, string, bool).

2.1.1. PHP 1.0 i 2.0

Iako je ideja zaživjela više od godinu dana prije, Rasmus Lerdorf je prvu „stabilnu“ verziju PHP: FI-a (prvotni naziv) [14] pustio u opticaj 1995. godine. Radilo se zapravo o skriptama pisanim u C-u čiji je zadatak bio olakšati Lerdorfu integraciju skripti u HTML, s posebnim naglaskom na forme. Od tud i naziv „Personal Home Page Tools: Forms Interpreter“ ili skraćeno PHP:FI [14].

Prva verzija je služila isključivo za potrebe dizajna Lerdorfove web stranice.

Slika 2.4. Rasmus Lerdorf

U travnju 1996. godine izlazi verzija 2.0. Uključuje podršku za DBM, mSQL i Postres95 baze podataka [14]. Još u tom razdoblju su se PHP skripte integrirale u HTML preko komentara. Kako je internet postajao sve popularniji medij, prvenstveno zahvaljujući sve bržem prijenosu podataka, tako se i popularnost PHP-a širila. Do 1998. godine je broj web stranica, koji je na ovaj ili onaj način inkorporirao PHP kod, bio na oko 60 000, što je bilo otprilike 1% od ukupnog broja stranica prisutnih na webu. Loša praksa je bila da se razvoj jezika previše oslanjao na necentralizirani pristup, tj. da su bugovi i implementacije bile servisirane od strane entuzijasta. Takvim pristupom je, po samoj naravi stvari, doći do nejasne vizije daljnjeg napretka, bilo u sintaksi, bilo u dometu samog jezika.

2.1.2. PHP 3.0

Za potrebe svojeg projekta na sveučilištu u Tel Avivu, Zeev Suraski i Andi Gutmans su 1997. godine počeli raditi na vlastitom parseru PHP koda [15]. Nakon razgovora sa Lerdorfom, počinju sa kompletnim redizajnom PHP-a koji sada dobiva naziv „PHP: Hypertext Preprocesor“.

Osim izgleda, koji poprima oblik kakav danas poznajemo, PHP uvodi implementaciju u Apache server, kao robustan Apache modul [16]. Moderan izgled sučelja za komunikaciju sa više različitih baza podataka, protokola, API-a, te ekstenzibilnost je omogućilo nagli rast besplatnih modula koji su se mogli saljnje razvijati ili koristiti prema potrebi od strane korisnika.

Također jedna od značajki PHP-a 3.0 jest rudimentarna podrška za objektno orjentirano programiranje (OOP), iako pretežito praksu proceduralne forme jezik zadržava do verzije 5.0.

Zahvaljujući podršci za servere bazirane na Windows i Macintosh OS-u, zastupljenost web stranica kodiranih u PHP-u se penje na 10%.

2.1.3. PHP 4.0

Baziran na novom Zend Engine-u 1.0 (1999. g.), 22. svibnja 2000. godine [17] izlazi PHP 4.0. U kolovozu 2008. izašla je zadnja inačica, 4.4.9., te više nema podršku za zakrpe i nadogradnje.

Četvrtu verziju krasi konačno podrška za neki drugi server osim Apachea, a jedan od servera koji dobiva podršku je i Microsoftov IIS. Također je poboljšano rukovanje memorijom [16], kako bi se spriječio tzv. memory leak tj. postupni gubitak memorije računala zbog toga jer program ne uspijeva „vratiti“ memoriju koja mu biva dodijeljena za izvršavanje nekog zadatka. Nešto izraženiji primjer toga možemo vidjeti kod računalnih igara ili grafičkih programa, kada nakon nekog vremena računalo primjetno usporava rad, pa moramo ili izaći iz programa ili napraviti reset računala.

Dalje se nastavlja sa implementacijom OOP-a započetog u PHP 3.0 inačici, što omogućava razvoj kompleksnijih i zahtjevnijih projekata.

2.1.4 PHP 5.0

Izlazi 13. srpnja 2004. godine, te je originalan Zend Engine zamijenjen inačicom 2.0. Možda najvažnija osobina koju je ova verzija donijela jest PDO notacija ili PHP Data Objects. Do ove verzije, PHP je uglavnom bio proceduralan jezik. Učestalije se počinju posuđivati prakse iz drugih programskih jezika, pa tako se počinju oponašati MVC framework arhitekture po kojima je bio poznat Ruby on Rails. Prvo je bio napravljen CakePHP, pa Zend Razvojni okvir, dok su u zadnje vrijeme postali popularniji Lithium i Laravel, koji se baziraju, prvi na CakePHP-u dok potonji koristi kao jezgru Symfony [18].

Još neke od inovacija koje uvodi peta inačica je Class autoloading [19], redizajnirana XML podrška, uključujući SimpleXML, mysqli ekstenzija, SQLite podrška.

PHP 5.3 i 5.4

Verzija 5.3 je zaživjela tijekom 2009. godine dok je 5.4 puštena u optjecaj 1. ožujka 2012. Kroz to razdoblje, točnije 2010. prekinut je rad na razvoju verzije 6.0, iz koje su neke inovacije implementirane u verziju 5.5.

5.3 uvodi Namespaces, PHP Archives (PHAR) koje sliče Javinim JAR arhivama [20], dok inačica 5.4 miče *register_globals* uz nekolicinu legacy opcija, uvodi short array sintaksu, Built-in web server itd.

Ove dvije verzije su još uvijek najzastupljenije kod izrade web aplikacija i stranica, budući da podržavaju *mysql_* sintaksu koja je, nažalost, izbačena iz sljedeće inačice PHP-a.

PHP 5.5

Prvobitno odgođen na 3 mjeseca, konačno izlazi 20. lipnja 2013. godine, prvenstvano zbog dodatka Zend OpCache-a (poboljšava preformanse PHP-a tako što posprema prekompajlirane skripte u zajedničku memoriju, čime je suvišno učitavanje i analiza skripti kod svakog učitavanja [21]).

PHP 5.5 uvodi try/catch, po uzoru na Javu, te jednostavan password hashing API, a napušta *mysql_* sintaksu. Ova verzija dolazi sa WAMP serverom koji je korišten u izradi praktičnog rada.

Inačica 5.6 je izašla tijekom 2014. godine.

PHP 7

Verzija 7, koja je i konačna verzija PHP-a, je izdana 3. prosinca prošle godine (2015).

Neke od najavljivanih novina koje ova verzija donosi jest veća brzina izvođenja (ponekad i do 3 puta veća u Wordpress-u), 100%+, manje korištenje memorije, poboljšan OPCache, tj. moguće je napraviti backup cache-a na disku, pa ako kojim slučajem dolazi do nenadanog gašenja računala/servera, iako gubi određeni postotak na brzini kod ponovnog učitavanja, ipak je brže od učitavanja ispočetka.

Također će biti moguće odrediti tip vraćanja podataka iz metode, a iz SQL-a posuđuje NULL Coalesce Operator (??) koji vraća prvu TRUE vrijednost [22]. Radi se zapravo o *isset()* ternarnom operatoru. Primjerice, sljedeći dio koda iz prethodnih verzija:

```
$route = isset($_GET['route']) ? $_GET['route'] : 'index';
```

Sada se može pisati na sljedeći način:

```
$route = $_GET['route'] ?? 'index';
```

Još jedna od mnogih promjena dodataka je uvođenje *combined comparision operator-a* koji uspoređuje dva operanda, a povratna vrijednost je uvijek cijeli broj, bio on pozitivan, negativan ili pak nula (u slučaju jednakosti operanada). Objekte ne prepoznaje, te u slučaju pokušaja usporedbe dva objekta vraća grešku undefined.

Također je dodana mogućnost deklariranja skalarnih tipova varijabli: int, float, string i bool.

2.2. MySQL

Baza podataka osnovana 1995. godine od strane švedske tvrtke MySQL AB, te je bila u njihovu vlasništvu do 2008. dok biva prodana Sun Microsystems-u za otprilike 1 milijardu $ [23].

Baziran je na mSQL-u, a većinom pisan u C/C++. MySQL je open source relacijska baza podataka široke primjene i obično dolazi predpakiran u raznorazne LAMP spin-off all-in-one instalacije za razvoj weba. Zapravo, „M“ u LAMP-u označava MySQL.

Sintaksom je blizak klasičnoj SQL sintaksi, te ovu bazu podataka koriste neke od najprometnijih web stranica, poput Facebooka, YouTube-a, Twittera, Flickr ili Google, iako Google ne koristi MySQL za pretraživanje [24].

Većinom ne dolazi sa grafičkim sučeljem, osim na Windows OS-u. U WAMP instalaciji MySQL bazi podataka se pristupa preko phpMyAdmin-a ili komandne linije. Također je moguće samostalno instalirati MySQL Workbench, ako se primjerice razvija aplikacija u Ruby-u.

U našem slučaju je moguće izraditi novu bazu, te tablice i atribute, preko phpMyAdmina, komandne linije ili direktno iz PHP koda. Obično je praksa, ako se ne radi o nekom vlastitom razvojnom okviru ili CMS-u kada se stvara nova baza podataka preko GUI forme prema nahođenju korisnika, izraditi sve elemente u phpMyAdminu. Također je dobra praksa za svaku novu bazu podataka napraviti novi master user korisničko ime i lozinku sa svim privilegijama, a korisničko ime bi trebalo odgovarati imenu baze. Tako se bolje štite podaci od mogućih napada.

Preko PHP-a se umeću podaci ili se njima manipulira preko *prepare()* i *execute()* PDO metoda.

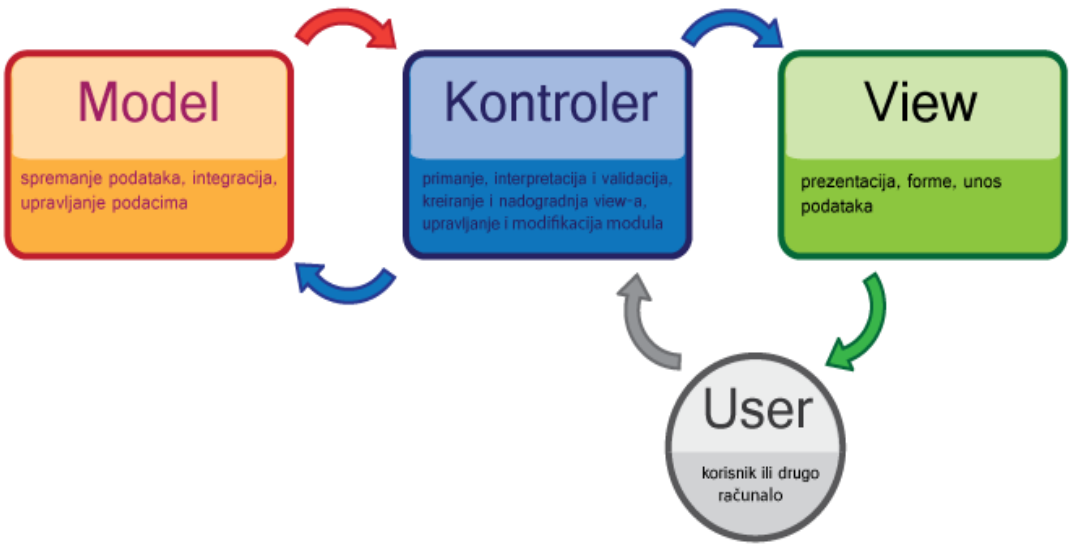
Za izradu ovog rada je korištena verzija 5.6 uz verziju phpMyAdmina 4.1.

3. MVC – „Model-View-Controller“

Od početka 90-ih prošlog stoljeća zastupljenost korisničkih sučelja u primjeni počelo je nesrazmjerno rasti u odnosu na prošla desetljeća. Usprkos postojanju korisničkih sučelja na računalu, pa čak i naprednijih poput Amige, i u prijašnjim razdobljima, ipak su dominirala sučelja kao što je DOS. Ta brojka se višestruko povećala od ranih 2000-ih rastom interneta i brzih računala. To je imalo za posljedicu sve veću potrebu za organizacijom podataka koji su se odvijali na relaciji korisnik-sučelje-baza podataka. MVC arhitektura upravo u tom razdoblju dobiva na važnosti kod razvoja aplikacija.

1979. godine Trygve Reenskaug, norveški računalni znanstvenik, je formulirao naziv MVC i to prvenstveno zbog GUI sotwarea. Radilo se o implementaciji za programski jezik SmallTalk. Iako zamišljen kao podrška SmallTalk aplikacijama, alati SmallTalka su bili implementirani preko MVC predloška. MVC model je bio jedan od prvih koji je koristio protokole za definiranje komponenti. Svaka klasa je imala set poruka (podataka) na koje je trebalo dati odgovarajuću radnju, isto ako i view klase, no nisu imale ograničenja što se tiče načina izvršavanja posla.

MVC arhitektura se sastoji od tri bazne komponente: Model, View i Controller. Njihov odnos najbolje prikazuje sljedeća ilustracija.



Slika 3.1. MVC model

Relacija između njih samih naspram korisnika je prikazana po principu: Model prima direktno informacije samo iz Controllera, dok je korisniku dopušteno međusobno komunicirati i sa Controllerom i Viewom. Model također može vraćati podatke u View.

3.1 Model

Model je centralna komponenta kojoj je primarna uloga rješavanje problema i upita iz Viewa. Preko Modela pristupamo podacima u bazi podataka i esencijelno je da u dobro organiziranoj aplikaciji, korisnik nikad nema direktan pristup Modelu. Ponajprije zbog sigurnosnih razloga.

No ne bavi se samo regulacijom podataka iz baze od <-> prema korisniku. Ova komponenta se bavi i logikom što činiti sa podacima. Ako se stanje podataka mijenja, Model daje do znanja View-u o toj promjeni.

Aplikacija može sadržavati jedan ili više Modela. Idealno je da za svaki Model postoji jedan Controller. No, Model ne ovisi o Controlleru ili View-u. Njegova jedina zadaće je obrada podataka i nema potrebu znati što se događa niti u View niti u Controlleru.

3.2 View

View je vizualan dio web aplikacije. Budući da se PHP uglavnom koristi za razvoj web aplikacija, više ne govorimo o desktop softwareu.

Za razliku od Modela koji se bavi isključivo podacima, pa je zbog toga manje podložan promjenama, View je u stanju stalnih promjena. Pod time ne mislim samo na razmjenu podataka sa samom bazom, nego prvenstveno na evoluciju korisničkog sučelja. Zbog toga je više podložan vanjskim utjecajima i traži veće prilagodbe.

Primjerice, razvoj vizualnog identiteta web stranica se drastično promijenio u zadnjih 15 godina. Od jednostavnog HTML-a, preko CSS-a, pa do naprednijih alata kao što je SASS[7]. Samim time je porasla potreba za interakcijom između korisnika i Viewa, budući da je web postao novi desktop. View stoga treba zadovoljiti sve zahtjevnijeg korisnika po pitanju količine i raznolikosti podataka.

PHP obično ima malo veze sa ovim dijelom MVC arhitekture te tu obično dolaze do izražaja client-side skriptni jezici, kao što je JavaScript.

[7] http://sass-lang.com/

3.3 Controller

Onaj dio obrasca koji kontrolira pristup web stranici jest Controller. Controller provjerava i prosuđuje odnos između Modela i Viewa. Točnije, Controller prosljeđuje upite iz Viewa Modelu, koji na temelju toga vadi podatke iz baze i prikazuje ih korisniku.

Controller zapravo odrađuje najviše posla u aplikaciji. Primjerice, svaki unos podataka ili pritisak na gumb stranice, odlazi na provjeru u Controller koji odredi radnju koju treba proslijediti Modelu. U nekim slučajevim su Controller i View ista stvar, no to nije idealno, zbog sigurnosnih razloga.

Sam Controller je zapravo ovisan o Viewu i Modelu i sam po sebi nema drugu svrhu osim što služi kao poveznica između Viewa i Modela.

3.4 Routing

Onaj dio koji fali klasičnoj desktop aplikaciji jest routing, točnije url routing. Da bi web aplikacija funkcionirala kako treba potrebno je odrediti koje stranice su i dostupne i kojim korisnicima. Primjerice, o razini pristupa korisnika ovisi širina pristupa sadržaju.

Postoji statični i dinamični routing. Statični je onaj u kojem je pristup određenim stranicama predodređen statičnim vrijednostima u kodu, dok se dinamičan oslanja na promjenjive elemente kao što su, primjerice, varijable. Dinamičan obično dodatno opterećuje Controller.

3.5 D-R-Y

D-R-Y ili Don't Repeat Yoursel je jedan od glavnih argumenata u korist korištenja MVC modela. Igra veliku ulogu u organiziranju koda u manje logičke strukture koje, kao što to i ima sugerira[8], imaju svrhu opetovanog korištenja. Također je važan aspekt optimizacije, jer je lakše kontrolirati kod i samim time smanjiti vrijeme potrebno za izvršavanja zadataka unutar web stranice. Također, pravilno izveden D-R-Y raspoređuje resurse po principu da su elementi neovisni jedan o drugom, pa tako ako jedan dio zakaže, nema utjecaj na neki drugi. Otud sličnost i korelacija sa MVC-om.

Neki oblikovni obrasci slični MVC-u su MVA (Model-View-Adapter), PAC (Presentation-Abstraction-Control), HMVC (Hijerarhijski MVC), MVP (Model-View-Presenter).

[8] D-R-Y (Don't Repeat Yourself) – nemoj se ponavljati

4. Laravel – PHP MVC razvojni okvir

Kako je zadnjih godina porastao broj korisnika računalnih tehnologija (desktop, mobilno, konzole), a svijet doslovce živi on-line, aplikacije postaju sve kompleksnije kako bi se zadovoljile potrebe istih. U takvom okruženju nerijetko je prava komocija raditi aplikaciju od same nule.

Zadaća razvojnog okvira jest kako bi se premostio jaz između dobre, stabilne aplikacije i količine utrošenog vremena. Neke od njihovoh zajedničkih karakteristika su [25]:

- brz razvoj aplikacija
- dobro organiziran, čitljiv, te reuzabilan kod
- riješeno je pitanje low level sigurnosti
- slijedi MVC arhitekturu koja garantira odvojenost prezentacije i logike
- promoviraju prakse modernog web razvoja poput objektno orijentiranih alata

PHP ima nekoliko dobrih rješenja kada su u pitanju razvojni okviri. Zend, CakePHP, Codeigniter, Symfony samo su neka od imena koja su se pojavila puno prije Laravela. No unatrag koju godinu, dvije Laravel bilježi nagli porast primjene bilo za osobnu ili komercijalnu primjenu [26].

Laravel je predstavio u lipnju 2011. godine Taylor Otwell. Ideja je bila napraviti razvojni okvir koji bi u potpunosti prigrlio MVC arhitekturu, laku manipulaciju relacijskim bazama, modularnost paketa, te specifičnu sintaksu koja bi omogućila lakše, „prirodnije“ razumijevanje početnicima, a manje pisanja bespotrebnog koda iskusnima.

Prvoj verziji razvojnog okvira MVC još nije bio u potpunosti implementiran. Korisnik je mogao napraviti Model i View, no ne i Controller. Također je imao podršku za lokalizaciju, autentikaciju, te routing i session-e.

Tri mjeseca kasnije ili u rujnu 2011. izlazi Laravel 2. Podržavao je i Controller, pa stoga postaje punokrvan MVC razvojni okvir. Za ovu inačicu je isto tako važno da je uveden Blade templating sistem. Radi se o View komponenti koja koristi svoju vlastitu verziju PHP jezika kako bi krajnji korisnik manipulirao podacima aplikacije. Sintaksa je poprilično intuitivna i logički dobro postavljena, a isto tako vrši low level provjeru sigurnosti.

Laravel 3 izlazi početkom (veljača) 2012. godine. Ono bitno u odnosu na prethodnu verziju jest da je uveden Artisan Command Line Interface (CLI), podrška za veći broj baza podataka, te migracije.

Artisan je iznimno koristan dodatak razvojnom okviru, budući da se koristi za gotovo sve važnije operacije kod kreiranja novih komponenti aplikacije. U mogućnosti je napraviti kostur billo kojeg dijela, uključujući u kod i biblioteke koje su potrebne za nativno izvršavanje

zadatka. Primjerice, kod CRUD kontrolera je moguće Artisanom napraviti sve potrebne klase i metode.

Migracije se također mogu izraditi iz Artisana. One preko *Schema()* klase i pripadajućih metoda prave nacrt stupaca baze podataka.

Ova verzija također počinje koristiti podršku za modularne pakete. Radi se o korisnim paketima koji olakšavaju rad i mogu se koristiti prema potrebi bez da se razvojni okvir u startu prepuni bespotrebnim stvarima.

U svibnju 2013. predstavljen je Laravel 4. Osim što je bio iz korijena ponovno pisan, također je uključio Composer kao manager paketa. Composer koriste i još neki PHP razvojni okviri kao što su: Symfony 2, CakePHP, Drupal, CodeIgniter.

Isto tako počinje podržavati veći broj e-mailova, te database seeding. Database seeding je mogućnost upisivanja lažnih podataka u bazu obično u svrhu provjere rada baze podataka prije nego što se krene sa razvojem programske logike u kontroleru. U Laravelu se u tu svrhu obično koristi paket Faker.

Laravel 4 također implementira mogućnost, tzv. soft deletion-a. To je opcija kod brisanja podataka koja ne briše podatke definitivno nego ima mogućnost povratka na staro. To se radi na način da se preko Eloquenta u zaglavlju dokumenta uključi biblioteka *Illuminate\Database\Eloquent\SoftDeletes;*

U kodu se definira na sljedeći način:

```
protected $user = ['deleted_at'];
```

Potrebno je i napraviti stupac u tablici preko schema buildera (*$table->softDeletes()*), te prizvati kasnije „izbrisane“ podatke koristeći *trashed()* metodu.

Zadnja verzija Laravela je izašla prošle godine u veljači. Najvažnija promjena jest unutarnje restrukturiranje direktorija, te neke važnije promjene u sintaksi. Primjerice, više se ne koristi Input nego Request za dohvaćanje podataka od strane korisnika. Također su uvedene i minimalne promjene sintakse u Blade-u.

Nažalost, veća negativna promjena jest izbacivanje nativne podrške za Html i Form builder koje je sada potrebno naknadno uključiti u projekt kao vanjski paket.

Inačica 5.1 dobiva i svoju LTS (Long Term Support) podršku na dvije godine.

Verzija 5.2 implementira cjelokupni sustav autentikacije koji je moguće prenamijeniti prema potrebi ili koristiti predefiniranog. Preko Artisana (make:auth) Laravel sam odradi dizajn i sav potreban kod za registraciju, login, te vraćanje zaboravljene lozinke preko maila.

Također i sav route sistem biva uvršten u web middleware grupu bez potrebe eksplicitnog navođenja. Web middleware se brine za cookies, session, CSRF zaštitu i sl.

Laravel je u vrlo kratkom vremenu od svojih nešto skromnijih početaka zahvaljujući jasno zacrtanom putu postao iznimno dinamičan i popularan razvojni okvir.

Velik dio popularnosti može zahvaliti i sluhu za pametna rješenja iz drugih razvojnih okvira, poput Railsa, te podršku za korisničke pakete.

Osobno mislim da mu je možda najveća mana za sada, uz pokoji bug, nedovoljan broj literature i/ili instrukcijskih videa koji bi išli u tančine. Pod time posebno mislim na middleware koji je iznimno važan, no isto tako nedovoljno zastupljen. No zadnjih mjeseci i to se pomalo mijenja, pa je opet moguće naći oveći broj kvalitetnog materijala koji se tiču zadnje verzije Laravela.

5. Praktična izvedba MVC arhitekture u Laravel razvojnom okviru

5.1 Opis aplikacije

Na prijedlog mentora za praktični dio rada odlučio sam napraviti aplikaciju za izračun putnih troškova profesora sveučilišta. Aplikacija se sastoji od početne stranice, login sustava, te odvojenih privilegija pristupa podacima od strane korisnika i administratora. Korisnik je nakon autentikacije preusmjeren na korisničku stranicu gdje može provjeriti svoje osobne podatke, dodati novi putni trošak ili pogledati ispis starijih.

Administrator ima mogućnost dodavanja novog korisnika, te brisanja i editiranja postojećeg. Također je u mogućnosti provjeriti putni trošak svakog korisnika, te dodijeliti administratorska prava prema potrebi.

Za potrebe ovog rada administratorska prava su dodijeljena meni i profesoru Stanisavljeviću.

Aplikacija je postavljena na server sveučilišta: http://arwen.velv.hr/~ivstaba/

5.2 Alati za izvođenje aplikacije – instalacija i opis

Osim WAMP servera, koristio sam Laravel 5.2 MVC Razvojni okvir. Da bi mogli koristiti Laravel razvojni okvir, prvo moramo imati instaliran Composer. Budući da smo na Windows 10 OS-u dovoljno je preuzeti Composer instalacijski paket.

Sam Laravel se može instalirati na dva načina preko Composera. Ako dodamo u PATH OS-a *~/.composer/vendor/bin* možemo koristiti naredbu u Command prompt-u:

>laravel new *#ime_aplikacije*

Drugi način je da ne dodajemo ništa u PATH nego preko komandne linije koristimo create-project naredbu:

> composer create-project –prefer-dist laravel/laravel *#ime_aplikacije*

Ono što smo upisali pod #ime_aplikacije, Composer koristi kao referencu preko koje stvara direktorij pod istim imenom unutar putanje komandne linije. Koristio sam, umjesto defaultne Windows komandne linije Command prompt i/ili Power shell, Git Bash koji, osim što nije potebno upisivati ili kopirati putanju u komandnu liniju nego je dovoljan desni klik miša na ikonu direktorija kako bi se pristupilo opciji otvaranja u Git Bashu, omogućava korištenje UNIX komandi na Windows operativnom sistemu.

Git Bash čini lakšim korištenje built-in PHP servera budući da Laravel projekt nije vezan na strogu strukturu izvođenja aplikacija na lokalnom serveru. Primjerice, kod WAMP-a svi projekti se smještaju u */www* direktorij, dok je kod XAMPP-a to */http* direktorij. U našem

slučaju je dosta koristiti naredbu **>php artisan serve** koja pokreće lokalni server na adresi *http://localhost:8000.* Iz servera se izlazi kombinacijom CTRL + c tipki na tastaturi.

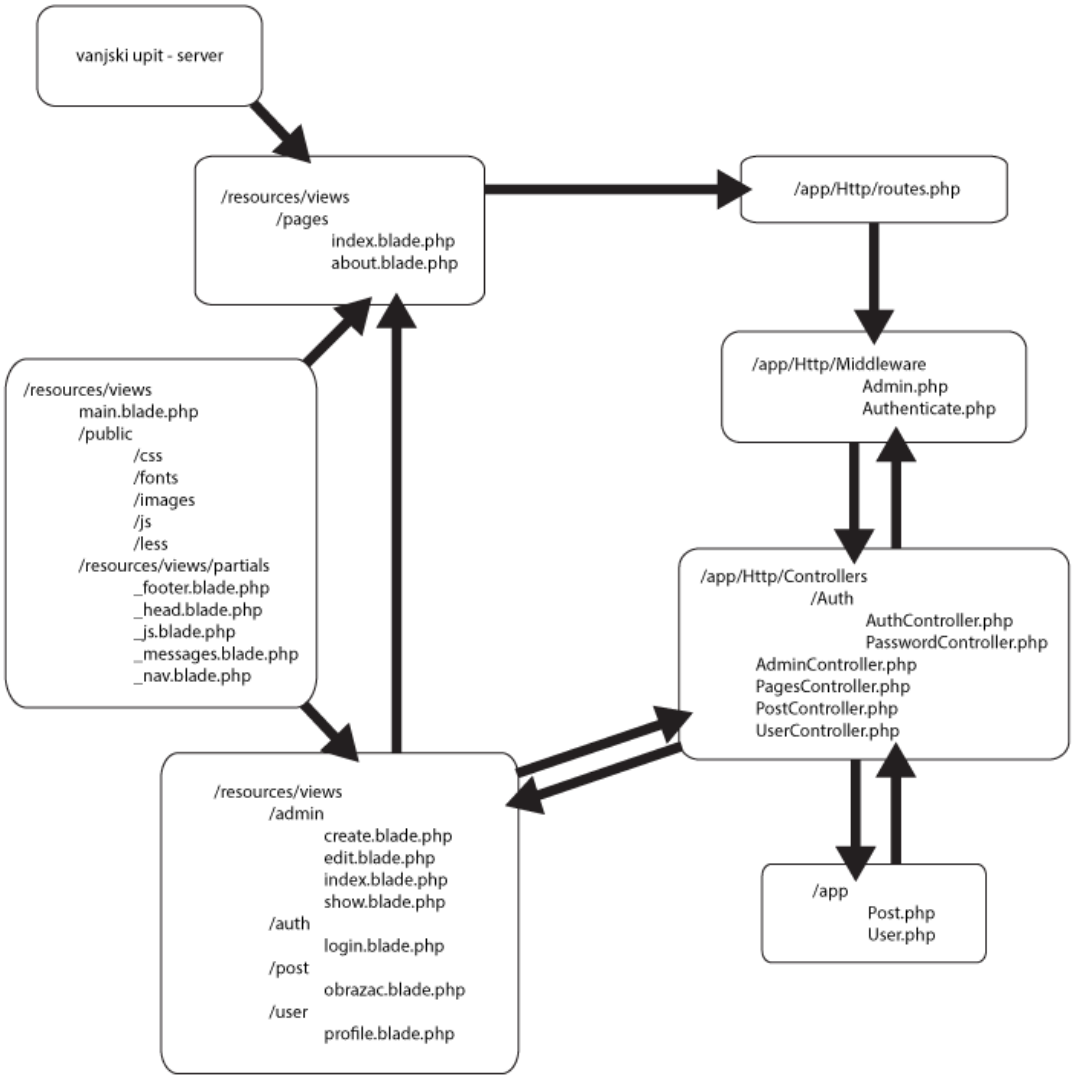
Ako želimo promijeniti port koristimo *–port* naredbu u sklopu artisan naredbe. Primjerice:

>php artisan serve –port=8080

Laravel također koristi Homestead/Vagrant virtualno okruženje koje se preporuča za razvoj kompleksnijih aplikacija.

5.3 Model izvođenja aplikacije

Aplikacija se izvodi na serveru prema sljedećem principu:



Slika 5.1. Shematski prikaz web aplikacije

Ulaz u aplikaciju je preko /resources/views/pages direktorija koji sadrži indeks.blade.php i about.blade.php, a služi kao frontend aplikacije. Routing je od verzije 5.2.3 (inačica korištena za ovaj praktični rad) po defaultu web middleware, pa stoga ne treba to izričito zadati kod definiranja. Routing se provjerava middlewareom, ovisno o privilegijama, te prema tome omogućuje korisniku komunikaciju sa bazom podataka preko Controllera i Modela. View datoteke koriste zajedničke vizualne resurse koji se nalaze u public direktoriju(u verziji 4.2 se sav frontend nalazio na istom mjestu, no sada je ta logika odijeljena), te partials, praksa preuzeta iz Ruby on Rails-a, tj. dijelovi stranice su zasebno definirani da bi opet bili povezani preko main.blade.php datoteke.

6.4 Opis rada i koda aplikacije

Prije nego što krenemo sa razvojem aplikacije, trebamo prvo napraviti bazu podataka u MySQL-u. Naša baza se zove „troskovi“, no prije nego ju popunimo tablicama, trebamo dati do znanja Laravelu ime baze i autentikaciju pristupa bazi. U Laravelu 5.2 to se radi u .env datoteci koja se nalazi u početnom direktoriju. Konfiguracija ove baze jest:

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=troskovi
DB_USERNAME=root
DB_PASSWORD=

Korisničko ime i lozinka su ostavljeni u zadanim vrijednostima što inače nije preporučljivo zbog sigurnosti aplikacije.

Tablice teoretski možemo definirati preko phpMyAdmina, no u Laravelu se preporuča korištenje artisana, sučelja komandne linije koje dolazi u paketu sa razvojnim okvirom, te jedna od korisnih mogućnosti artisana je automatsko generiranje tzv. migracija .

U migracijama (database/migrations) generiramo tablice baze podataka. Za potrebe aplikacije trebaju se napraviti dvije tablice. Laravel ima implementirano ophođenje sa nazivima elemenata prema nekom uvriježenom standardu, stoga sam dao nazive tablicama *users* i *posts*.

Kod koji se upisuje u komandnu liniju:

>php artisan make:migration create_users_table –create=users

>php artisan make:migration create_posts_table –create=posts

Artisan generira kostur koda u koji trebamo dopuniti željenim stupcima tablice.

```
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;
class CreateUsersTable extends Migration {
public function up() {
Schema::create('users', function (Blueprint $table)) {
      $table->increments('id');
      $table->string('name');
      $table->string('position');
      $table->string('degree');
      $table->string('email')->unique();
      $table->string('phone');
      $table->boolean('admin');
      $table->string('username')->unique();
      $table->string('password');
      $table->rememberToken();
      $table->timestamps();
      });
}
public function down() {
      Schema::drop('users');
      }
}
```

Laravel generira migracijsku PHP skriptu koju trebamo ubaciti u bazu podataka. Klasa ima dvije funkcije od kojih jedna služi za kreiranje tablice *users*, a druga za njeno brisanje (*down()*). Ono što Laravel kreira po defaultu je *id* i *timestamps* redak, koji zapravo stvara dva stupca u tablici; *created_at* i *updated_at,* te inkrementirani id. *Username i email* je *unique* zbog toga da ne bi dva korisnika imala istu email adresu, te korisničko ime.

Za tablicu *posts up()* metoda ima sljedeće članove:

```
Schema::create('posts', function (Blueprint $table) {
      $table->increments('id');
      $table->string('polaziste');
      $table->string('odrediste');
      $table->float('km');
      $table->float('naknada');
      $table->float('iznos');
      $table->integer('user_id')->unsigned();
      $table->timestamps();
});
```

User_id je unsigned jer nam ne trebaju negativne vrijednosti, a udvostručujemo mogući broj mjesta u bazi podataka.

U slučaju naknadnog dodavanja stupca u tablicu koristi se artisan naredba:

>php artisan make:migration add_#ime_stupca_to_#ime_tablice
–table=#ime_tablice

Migracije se upisuju u bazu podataka sljedećom naredbom:

> php artisan migrate

U slučaju da želimo iz izbrisati migracije iz baze koristimo neku od sljedećih naredbi:

> php artisan migrate:rollback *// briše zadnju migraciju*

> php artisan reset *// briše sve migracije*

> php artisan refresh *// briše migracije i opet pokreće naredbu migrate*

5.5 Model i routes

Zadaća modela jest biti svojevrstan most, poveznica između korisnika i baze podataka. Ova aplikacija ima dva modela budući da ima dvije tablice, što je svojevrsna norma u Laravelu. Idealno je za svaku tablicu imati jedan model. Ako neki model koristi dvije ili više tablice potrebno je to naglasiti unutar samog koda (protected $table = '#ime_tablice').

Standard je da ime modela odgovara imenu tablice, s time da model treba biti u jednini, a ime tablice u pluralu engleskog jezika.

Post.php i User.php modeli se nalaze unutar */app* direktorija.

```
class Post extends Model {
public function user() {
      return $this->belongsTo('App\User');
      }
}
```

User.php:

```
class User extends Authenticatable {
      protected $fillable = [
            'name', 'email', 'password',
      ];
      protected $hidden = [
            'password', 'remember_token',
      ];
      public function posts(){
            return $this->hasMany('App\Post');
      }
}
```

Sa bazom komuniciramo preko Eloquent ORM [27] sučelja.

Bitna razlika između User.php i Post.php modela, osim što je prvi automatski generiran od strane Laravela, jest da User.php upravlja i autentikacijom korisnika, otud extends Authenticable, a ne Model. Varijabla *$fillable* odnosi se na polja koja je dopušteno ispuniti kod logiranja. Ako ne želimo da se neki unos sprema ili provjerava koristimo $guarded.

Iznimno je važno pravilno definirati relacije među modelima, u ovom slučaju *one-to-many*, budući da imamo situaciju gdje svaki korisnik ponaosob ima jedan ili više postova, tj. putnih troškova.

U modelu User.php deklariramo metodu *posts()* u kojoj definiramo relaciju prema Post.php linijom koda:

```
return $this->hasMany('App\Post');
```

Dok odnos prema pojedinom korisniku definiramo unutar metode user(), jednina jer se radi jednom korisniku, sljedećim retkom:

```
return $this->belongsTo('App\User');
```

Vlastiti model se izrađuje isto preko artisan komandne linije. Postoje dva načina na koja se to radi iako je moguće svaku datoteku modela napraviti ručno.

>php artisan make:model #ime_modela

>php artisan make:model #ime_modela –migration *// isto se može napisati samo -m*

Potonji odmah pravi migraciju modela.

Rute (routes) određuju navigaciju unutar web aplikacije. Sve nisu vidljive u URL-u preglednika, no svejedno služe kao svojevrstan autoput aplikacije.

```
Route::get('auth/login', ['as' => 'login', 'uses' =>
'Auth\AuthController@getLogin']);
Route::post('auth/login', 'Auth\AuthController@postLogin');
Route::get('auth/logout',['as'=>'logout','uses'=>'Auth\AuthControl
ler@getLogout']);
Route::resource('post', 'PostController', ['only' => ['create',
'store']]);
Route::resource('user', 'UserController', ['only' => ['show']]);
Route::resource('admin', 'AdminController');
Route::get('about', 'PagesController@getAbout');
Route::get('/', 'PagesController@getIndex');
```

Prve tri rute se odnose na login autentikaciju korisnika. Koristio sam djelomično Laravel login sustav koji je dio frameworka od verzije 5.

Resource routes, koje su sljedeće tri, su rute na **CRUD** controller. **CRUD** je akronim za **C**reate-**R**ead-**U**pdate-**D**elete, tj. kad se u artisanu napravi novi controller Laravel automatski pretpostavlja ovaj način korištenja controllera.

Budući da u slučaju *post* i *user* nije bilo potrebe definirati kompletan routing, koristio sam opciju *only* koja nam omogućava definirati samo one rute koje su mi bile potrebne. Isto tako možemo koristiti *except* kojom definiramo polje ruta koje ne želimo koristiti.

Administrator koristi cjelokupan CRUD sustav, pa je zbog toga bilo potrebno registrirati sve rute.

5.6 Controller i middleware

Zadaća kontrolera jest odrediti logiku komunikacije između korisnika i baze podataka. Kontroler zapravo odrađuje manje-više sav programerski posao.

Iako je preporučena praksa princip jedan model jedan controller, skloniji sam odijeliti logiku u zasebne controllere prema namjeni, pa stoga iako imam samo dva modela kontrolera jest pet.

Dva od toga su automatski generirani kod instalacije Laravela: *AuthController.php* i *UserController.php*.

AuthController.php je ostao gotovo nepromijenjen s moje strane jer mu je zadaća provjeravati korisnika kod logiranja. Ono što sam dodao jest ruta nakon uspješne autentikacije:

```
protected $redirectTo = ('user/{user}');
```

Također sam morao promijeniti izuzetak u middleware-u unutar __*construct* metode, budući da se radi o manjem bugu.

```
public function __construct() {
// defaultna vrijednost izuzetka je bila 'Logout'
$this->middleware($this->guestMiddleware(), ['except' =>
'getLogout']);
}
```

UserController.php provjerava je li korisnik koji pokušava pristupiti podacima pravilno autenticiran, te vraća već postojeće vrijednosti iz baze podataka isto kao što omogućava upis novih.

```
public function show($id) {
   $user = Auth::user($id);
   if($user){
       $posts = Post::where('user_id','=',$user->id)-
       >orderBy('created_at', 'asc')-> paginate(30);
       return view('user.profile')->withUser($user)-
       >withPosts($posts);
   }
}
```

Laravel ima implementirane metode poput *orderBy()* i *paginate()* preko kojih je poprilično lako i inuitivno baratati podacima iz baze podataka, u ovom slučaju sam odredio da se podaci koji su vraćeni strukturirani prema datumu do njih 30 na jednoj stranici.

Ako želimo napraviti vlastiti **CRUD** controller to radimo preko naredbe:

> php artisan make:controller #ime_controllera –resource

AdminController.php koristi CRUD u cijelosti.

```
class AdminController extends Controller {
  public function __construct(){
     $this->middleware('admin');
  }
  public function index() {
     $users = User::orderBy('id', 'asc')->paginate(20);
     return view('admin.index')->withUsers($users);
  }
  public function create() {
     return view('admin.create');
  }
  public function store(Request $request) {
     $this->validate($request, array(
        'name' => 'required|max:150',
        'degree' => 'required',
        'position' => 'required',
        'email' => 'required|unique:users',
        'phone' => 'required',
        'username' => 'required|min:3|max:20|unique:users',
        'admin' => 'required',
        'password' => 'required|min:5|max:30'
     ));
     $user = new User;
     $user->name = $request->name;
     $user->degree = $request->degree;
     $user->position = $request->position;
     $user->email = $request->email;
     $user->phone = $request->phone;
     $user->username = $request->username;
     $user->admin = $request->admin;
     $user->password = bcrypt($request->password);
     $user->save();
     Session::flash('success', 'Korisnik je uspješno kreiran!');
     return redirect()->route('admin.index');
  }
```

```
public function show($id) {
    $user = User::find($id);
    if($user){
        $posts = Post::where('user_id', '=', $user->id)-
        >orderBy('created_at', 'asc')->paginate(30);
        return view('admin.show')->withUser($user)-
        >withPosts($posts);
    }
}
public function edit($id) {
    $user = User::find($id);
    return view('admin.edit')->withUser($user);
}
public function update(Request $request, $id) {
    $user = User::find($id);
    if($request->input('email') === $user->email && $request-
    >input('username') === $user->username) {
        $this->validate($request, array(
        'name' => 'required|max:150',
        'degree' => 'required',
        'position' => 'required',
        'email' => 'required|email',
        'phone' => 'required',
        'username' => 'required|min:3|max:20',
        'admin' => 'required',
        'password' => 'required|min:5|max:30'
        ));
    } else if($request->input('email') === $user->email) {
        $this->validate($request, array(
        'name' => 'required|max:150',
        'degree' => 'required',
        'position' => 'required',
        'email' => 'required|email',
        'phone' => 'required',
        'username' => 'required|unique:users|min:3|max:20',
        'admin' => 'required',
        'password' => 'required|min:5|max:30'
        ));
    } else if($request->input('username') === $user->username){
```

```
            $this->validate($request, array(
            'name' => 'required|max:150',
            'degree' => 'required',
            'position' => 'required',
            'email' => 'required|email|unique:users',
            'phone' => 'required',
            'username' => 'required|min:3|max:20',
            'admin' => 'required',
            'password' => 'required|min:5|max:30'
            ));
        } else {
            $this->validate($request, array(
            'name' => 'required|max:150',
            'degree' => 'required',
            'position' => 'required',
            'email' => 'required|email|unique:users',
            'phone' => 'required',
            'username' => 'required|unique:users|min:3|max:20',
            'admin' => 'required',
            'password' => 'required|min:5|max:30'
            ));
        }
        $user->name = $request->name;
        $user->degree = $request->degree;
        $user->position = $request->position;
        $user->email = $request->email;
        $user->phone = $request->phone;
        $user->username = $request->username;
        $user->admin = $request->admin;
        $user->password = bcrypt($request->password);
        $user->update();
        Session::flash('success', 'Korisniku su uspješno nadograđeni
        podaci!');
        return redirect()->route('admin.index');
    }
    public function destroy($id) {
        $user = User::find($id);
        $user->delete();
```

```
        Session::flash('success', 'Korisnik je uspješno izbrisan.');
        return redirect()->route('admin.index');
    }
}
```

Prije nego što korisnik može pristupiti administratorskom sučelju, metoda __construct() provjerava koje privilegije taj korisnik ima.

__construct() je tzv. *magic method*, koja se priziva kad je neki objekt koji koristi tu klasu instanciran. U ovom slučaju je to poziv na middleware 'admin'.

Uloga middlewarea u MVC arhitekturi jest svojevrstan controller controllera. On kontrolira razne stupnjeve pristupa sadržaju, te stvara „mrežu“ ili bolje rečeno štit oko onog što želimo zaštititi. *Authenticate.php* dolazi sa Laravel instalacijom, a *Admin.php* je napravljen preko komandne linije:

> php artisan make:middleware Admin

Samu putanju middlewarea je trebalo dodatno definirati u Kernel.php datoteci.

```
'admin' => \App\Http\Middleware\Admin::class,
```

Princip provjere administratorskih prava je zapravo krajnje jednostavan u svojoj srži. Vrši se provjera autenticiranog korisnika je li njegov *admin* privilegij u bazi podataka *true* ili *false*. Ako je *true* onda je proslijeđen dalje, a u suprotnom vraćen na prethodnu stranicu.

Nakon što je uvjet zadovoljen korisnik je preusmjeren na administratorsko sučelje gdje može dalje manipulirati podacima prema potrebi.

U Laravelu 5 dolazi do nekoliko bitnih promjena u odnosu na prijašnje verzije.

Validator() metodu više nije potrebno definirati iz korijena, te instancirati unutar nove klase prije validacije. Sada je dosta koristiti *$this->validate* na polje sa članovima nad kojima vršimo neki od mogućih pravila validacije [28].

Osim server side koristio sam i client side validaciju. Pri tome sam se služio Parsley.js [29] knjižnicom.

Laravel 5 isto tako više ne koristi *$input* deklaraciju nego *$request* za dohvaćanje unosa korisnika.

PostController koristi *auth* middleware, koji provjerava jeli korisnik ulogiran i postoji li uopće, te ga prosljeđuje na korisničke stranice gdje može provjeriti jesu li postojeći podaci ispravni i dodati izračun novog putnog troška.

PagesController samo vrši management frontenda.

S programerskog aspekta controller je najzahtjevniji dio MVC arhitekture i tu se provodi većina vremena. Dobro je uvijek imati jasan pregled i plan izvedbe prema modelu ili viewu iz razloga jer kod većih projekata broj broj korištenih klasa i metoda eksponencijalno raste. Zbog toga sam sklon praksi da je bolje logički odijeliti controllere prema svrsi, ma kako god ona trivijalna bila, nego prezasititi nekolicinu njih metodama koje nerijetko nemaju neke druge poveznice osim zajedničkog modela.

5.7. View

U *resources/views* direktoriju Laravel razvojnog okvira se nalaze obrasci (forme) i ostale frontend PHP datoteke koje komuniciraju direktno sa korisnikom. Raspodijeljene su po direktorijima prema namjeni, a jedino se u glavnom views direktoriju nalazi main.blade.php koja služi kao predložak i drži strukturu cijele stranice.

Vizualni identitet sam temeljio na Bootstrap [30] frontend razvojnom okviru, dok su ikone dio Font Awesome paketa[9]. Fotografije, ilustracije, CSS, Javascript, te bilo koji sadržaj koji se tiče dizajna web aplikacije sprema se u *public* direktorij i u Laravelu se poddirektoriji uključuju automatski, pa tako nije potrebno navoditi putanju dalje od navedenog jer u protivnom sadržaj neće biti pravilno prikazan jednom kada je aplikacija pokrenuta na vanjskom serveru.

Laravel koristi *Blade* sustav predložaka. PHP kod ili od nekih od generatora, poput *HTML::* ili *Form::*, dolaze između *{{ }}* znakovlja, koji automatski vrši *htmlentities()* provjeru na upisani sadržaj. Ako nam to nije potrebno koriste se *{!! !!}* znakovi.

Blade koristi svoj vlastiti sistem naredbi preko kojih se regulira sadržaj među MVC komponentama aplikacije. U main.blade.php, temeljnom templateu, se vidi HTML struktura unutar koje su uključeni i raspoređeni resursi iz drugih direktorija i datoteka. U tu svrhu se koriste ključne riječi *@include* i *@yield*.

```
<!DOCTYPE html>
<html lang="hr">
   <head>
      @include('partials._head')
   </head>
   <body>
      @include('partials._nav')
      <div class="container">
         @include('partials._messages')
         @yield('content')
         @include('partials._footer')
```

[9] http://fontawesome.io/

```
        </div>
        @include('partials._js')
        @yield('scripts')
    </body>
</html>
```

@include ključna riječ uključuje neki vanjski element, u ovom slučaju aktivne HTML elemente. Ta praksa je dobra zbog preglednosti koda, te eventualnih naknadnih promjena.

@yield uključuje u *blade* template dinamični, promjenjivi sadržaj. U slučaju *main.blade.php* radi se o glavnom sadržaju, te Javascript-u.

Budući da je svaki razvojni okvir zapravo svojevrstan dijalekt izvornog programskog jezika, tako i Laravel koristi vlastiti sustav za komunikaciju i obnašanje sa podacima.

Petlje i uvjeti se pišu po principu: *@if() / @endif, @foreach / @endforeach*, itd. Strukturirani su po principu blokova tako da nije potrebno, primjerice nakon nekog @if() uvjeta, navoditi vitičaste zagrade ({}).

Komentare pišemo između *{{-- --}}* znakova, a ako je neki znakovni niz unutar dvostrukih vitičastih zagrada, a ne želimo da se tretira kao naredba, potrebno je napraviti escape znakova pomoću @ znaka (@{{ *Ne izvršava se kao dio koda* }}) [31].

Smisao View-a unutar MVC arhitekture jest da čim manje programerskog koda, koji bi potencijalno ugrožavao sigurnost cijelog sustava, bude pisano unutar blade predložaka. Za to je zadužen Controller.

Prvo što je prezentirano kod dolaska na web aplikaciju jest *indeks.blade.php* i *about.blade.php*.

Slika 5.2 index.blade.php

Indeks.blade.php ilustrira na koji način su organizirani dinamični podaci unutar View-a u Laravelu.

```
@extends('main')
@section('title', ' | Naslovna')
@section('content')
   <div class="row">
      <div class="col-md-12">
      <div class="jumbotron">
         <img src="/images/putni_troskovi.gif" alt="Putni
         Troškovi">
      </div>
   </div>
</div>
@endsection
```

Naredba @extends daje do znanja frameworku da je indeks.blade.php child datoteka u odnosu na main.

Sadržajem unutar View templatea u Laravelu se upravlja preko *@section* naredbe. Ona je povezana sa *@yield* naredbom preko atributa. U ovom slučaju imamo dva, a to su '*title*' i '*content*'. To znači da se svaka sekcija sa tim atributom prikazuje na mjestu gdje je *@yield('')* prizvan. *@section* možemo zatvoriti bilo sa *@endsection* ili sa *@stop*.

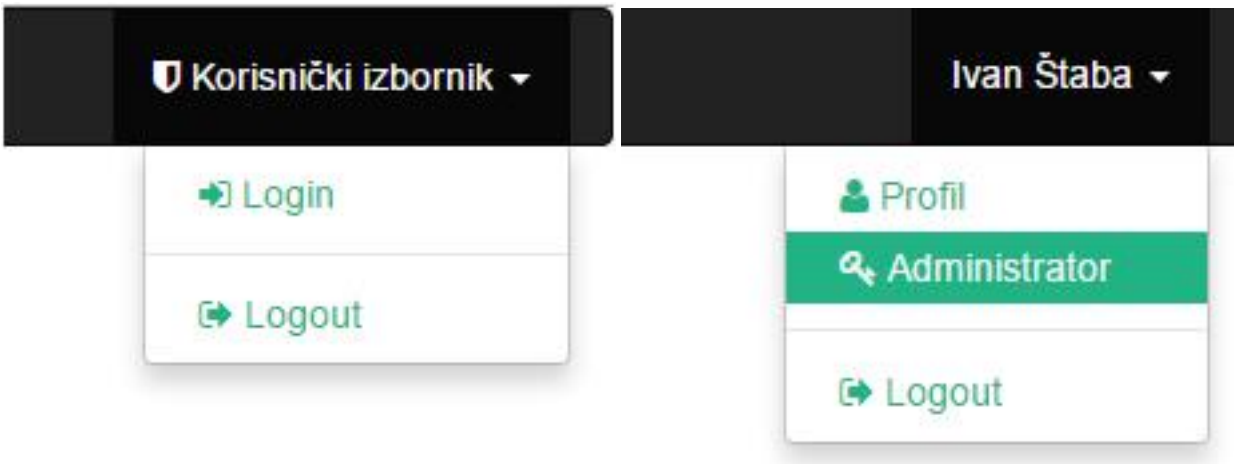
U gornjem desnom kutu nalazi se izbornik aplikacije u kojem korisnik ima mogućnost ulogirati se, te nakon uspješne provjere privilegija izbor između regularnog korisničkog sučelja ili administratorskog. Ispis imena registriranog korisnika je dobar i koristan dizajnerski trik.

Sama navigacija se nalazi unutar *partials* direktorija. Ta praksa je prenesena iz Railsa u Laravel i pridonosi preglednosti koda.

Dio _nav.blade.php koda koji je zadužen za taj dio, a ujedno prikazuje način izvršavanja uvjeta unutar blade templatea, te laku implementaciju unutar Bootstrap razvojnog okvira:

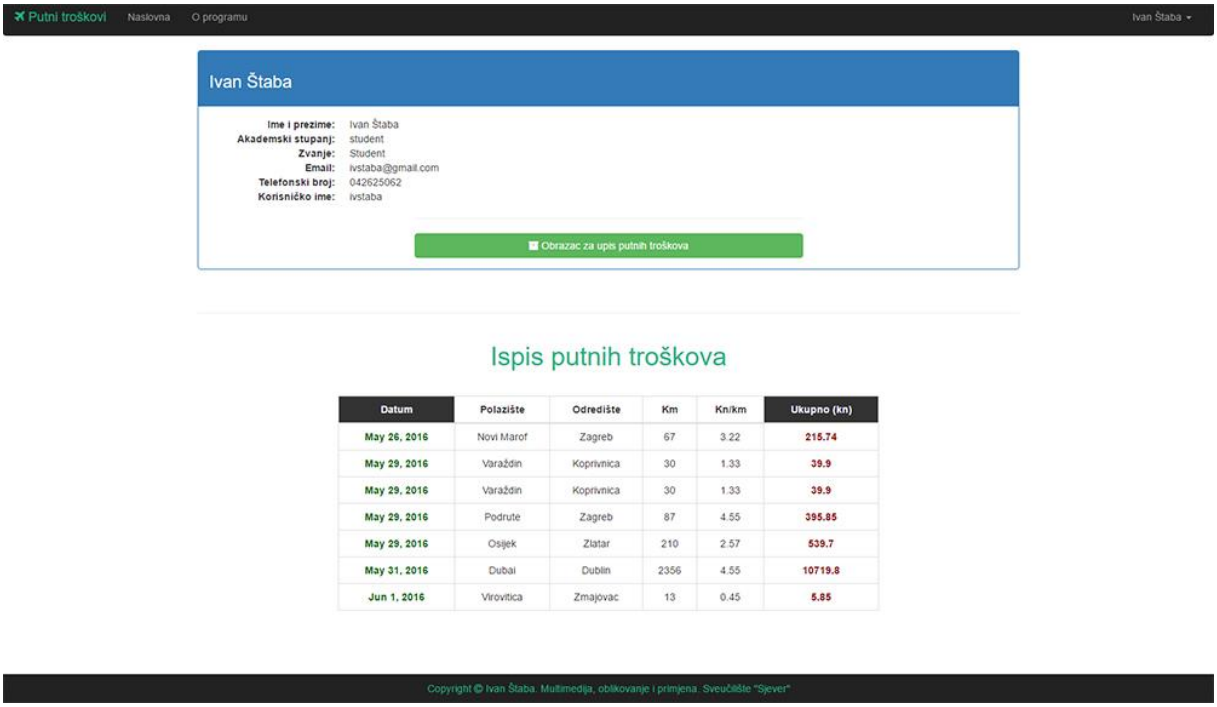
```
<ul class="dropdown-menu">
@if(Auth::check())
   <li><a href="{{ route('user.show', auth()->user()->id) }}"><i
   class="fa fa-user" aria-hidden="true"></i> Profil</a></li>
   <li><a href="{{ route('admin.index') }}"><i class="fa fa-key"
   aria-hidden="true"></i> Administrator</a>
@else
   <li><a href="{{ route('login') }}"><i class="fa fa-sign-in"
   aria-hidden="true"></i> Login</a></li>
@endif
   <li role="separator" class="divider"></li>
   <li><a href="{{ route('logout') }}"><i class="fa fa-sign-out"
   aria-hidden="true"></i> Logout</a></li>
</ul>
```

Metoda *check()* klase *Auth* provjerava je li korisnik logiran i u tom slučaju daje pristup rutama unutar *@if()* uvjeta.



Razlika između rute *user.show* i *admin.index* jest da prva ima i argumente koji su opcionalni. U ovom slučaju dohvaćaju id logiranog korisnika, te tako ruta čini prečac na profil korisnika. Laravel koristi točku (.) kako bi definirao putanju bilo na URL ili direktorij. Može se koristiti i standardna kosa crta(/), no preporuča se točka zbog objektno orijentirane notacije.

Nakon što je izvršena provjera korisnikovih akreditacija preusmjerava se na stranicu profila.



Na profilnoj stranici korisnik može provjeriti svoje podatke, te dodati novi putni trošak.

Linija koda koja ispisuje ime korisnika u zaglavlju dokumenta jest nešto drukčija od ostalih. Naime drugi argument treba biti pisan između dvostrukih navodnika („“), a ne jednostrukih ('') kako se kod ne bi interpretirao kao običan string nego objekt iz baze podataka.

```
@section('title', "| $user->name")
```

Ternarni operator pod stavkom *Korisničko ime* provjerava administratorska prava i u skladu s time ispisuje korisničko ime.

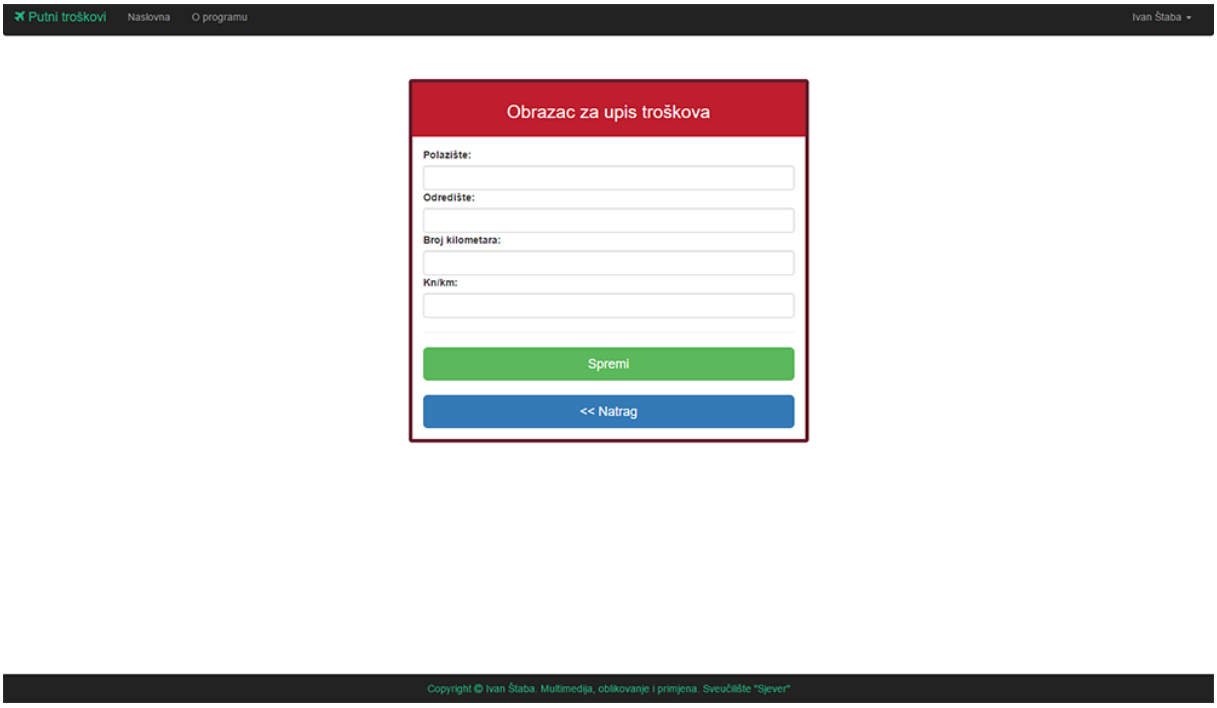
```
{{ ($user->admin === 1) ? $user->username : 'Korisnik nije
Administrator' }}
```

Za dohvaćanje podataka iz baze vrši se iteracija kroz programsku petlju *@foreach*. Kako bi se tekstualni datum iz baze pretvorio u UNIX timestamps koristi se kombinacija

date()[32] i *strtotime()*[33] metoda. *Date()* ima dva argumenta. Prvi formatira prikaz datuma, a drugi je *strtotime().*

```
@foreach($posts as $post)
   <tr>
   <th class="datum-body">{{ date('M j, Y', strtotime($post-
   >created_at)) }} </th>
   <td>{{ $post->polaziste }}</td>
   <td>{{ $post->odrediste }}</td>
   <td>{{ $post->km }}</td>
   <td>{{ $post->naknada }}</td>
   <th class="ukupno-body">{{ $post->iznos }}</th>
   </tr>
@endforeach
```

Gumb za ispunu obrasca putnih troškova vodi na view *obrazac.blade.php*.



Vanjski CSS i skripte se uključuju u dokument preko sekcija 'stylesheets' i 'scripts'. Laravel koristi vlastite kolekcije (Html, Form) koje je od verzije 5 potrebno posebno instalirati unutar composer.json datoteke [34].

Stoga reci koda koji uključuju CSS i Parsley izgledaju ovako:

```
{!! Html::style('css/parsley.css') !!}
{!! Html::script('js/parsley.min.js') !!}
```

Parsley je client-side validator koji se lako inkorporira unutar Laravela. To se radi unutar forme, a u Laravelu postoji izbor između klasične HTML sintakse ili korištenja kolekcija. Ako se koriste kolekcije neke stvari, poput CSRF zaštite, jesu automatski izvedene.

```
{!! Form::open(['route' => 'post.store', 'data-parsley-validate'
=> '']) !!}
   {{ Form::label('polaziste', 'Polazište:') }}
   {{ Form::text('polaziste', null, ['class' => 'form-control',
   'required' => '']) }}
   {{ Form::label('odrediste', 'Odredište:') }}
   {{ Form::text('odrediste', null, ['class' => 'form-control',
   'required' => '']) }}
   {{ Form::label('km', 'Broj kilometara:') }}
   {{ Form::text('km', null, ['class' => 'form-control', 'required'
   => '']) }}
   {{ Form::label('naknada', 'Kn/km:') }}
   {{ Form::text('naknada', null, ['class' => 'form-control',
   'required' => '']) }}
   <hr>
   {{ Form::submit('Spremi', ['class' => 'btn btn-success btn-lg
   btn-block', 'style' => 'margin-top:20px']) }}
{!! Form::close() !!}
```

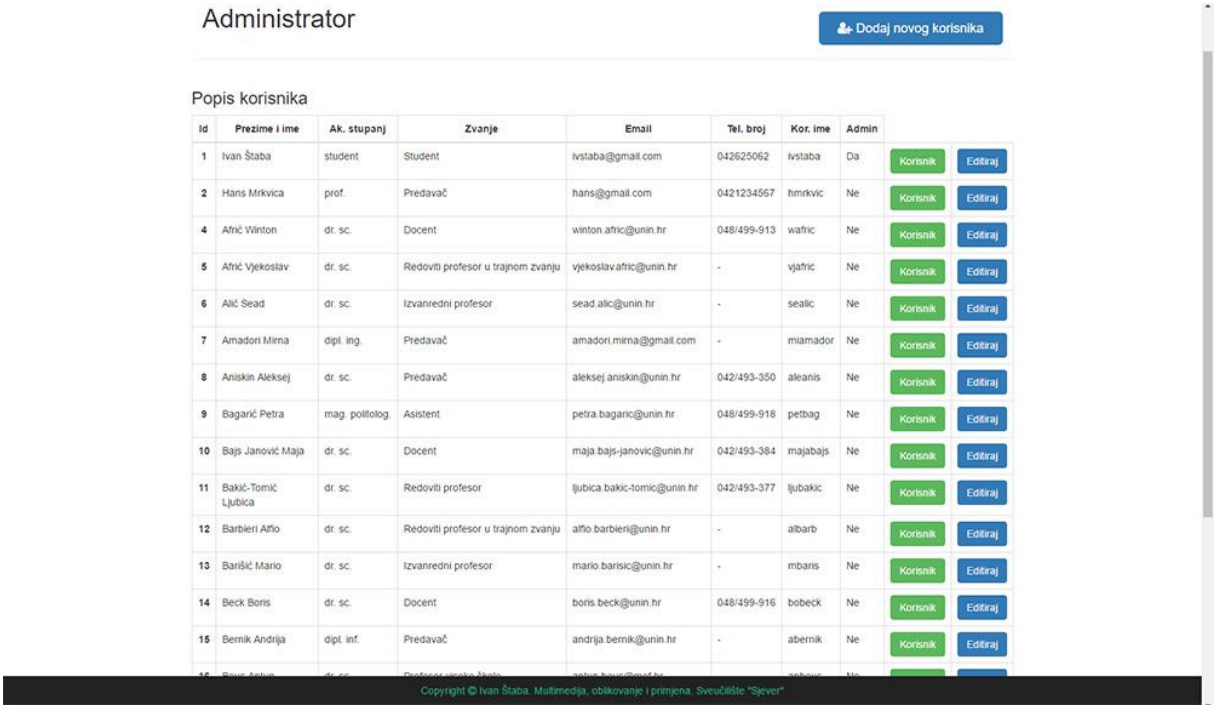
Između *Form::open()* i *Form::close()* se definiraju elementi forme. *Form::open()* ima nekoliko mogućih članova, u ovom slučaju je to imenovana ruta (named route) *'route' => 'post.store'*, te Parsley validacija.

Osim imenovane rute prvi član može i biti pokazivač na kontroler ili URL. Metoda se ne navodi ako je POST jer se to pretpostavlja. Ako je u pitanju GET, PUT i DELETE potrebno je to navesti ('method' => '#').

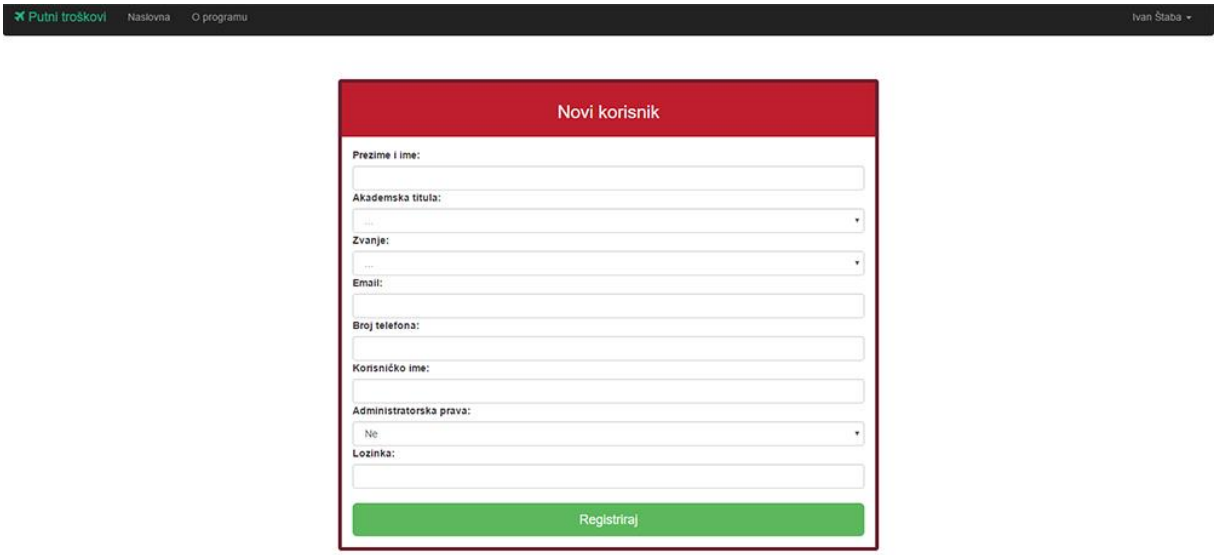
Form::label() i *Form::text()* moraju imati isto ime kako bi odnos između njih bio valjan.

Parsley validacija za svaki pojedini unos se nalazi unutar polja u kojem definiramo CSS klase.

Onom korisniku koji ima i administratorska prava omogućen je pristup administratorskoj stranici. Tu se može vidjeti popis svih korisnika, te dodati novi ili izbrisati i izmijeniti podatke već postojećem.

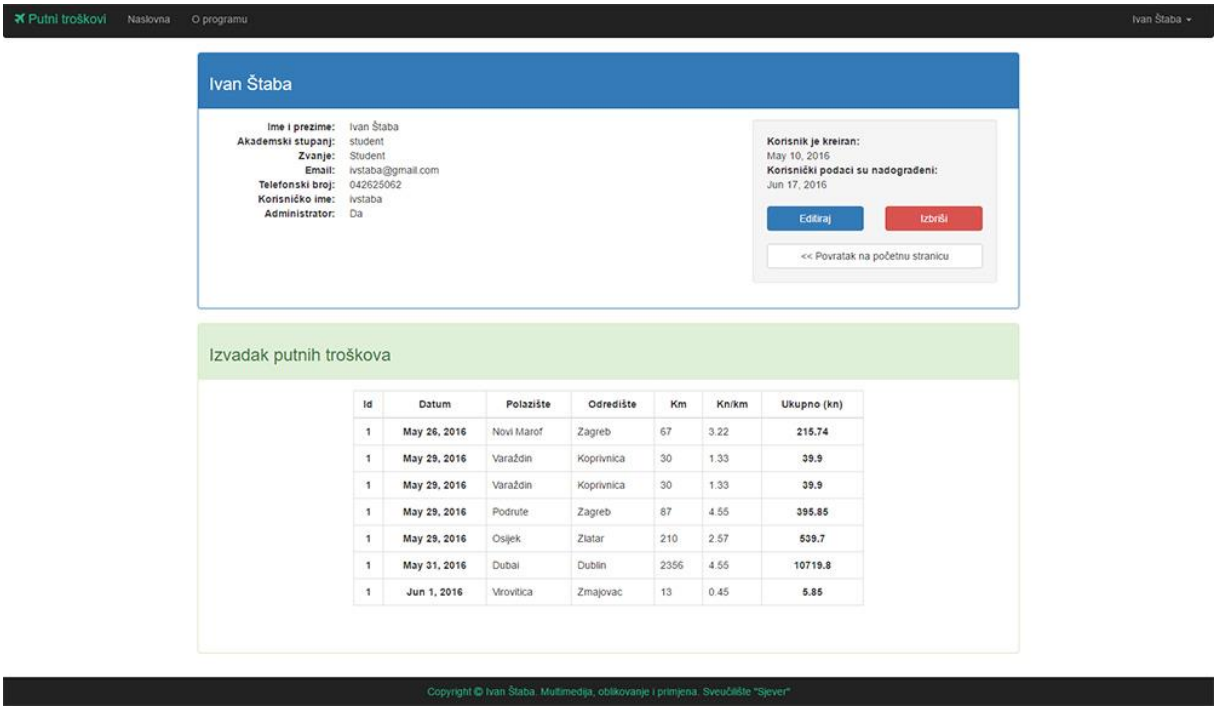


Novog korisnika se registrira koristeći CRUD controller preko forme unutar viewa *create.blade.php*.



Forma za editiranje korisnika je gotovo identična sa jednom bitnom razlikom da se u prazna polja već ispisuju postojeći podaci. To radimo tako da umjesto *null* vraćamo objekt vrijednosti iz baze podataka, primjerice *$user->name,* unutar *Form::text()* metode.

Korisnik se jedino može izbrisati u *admin/show.blade.php*.



View u Laravelu je poprilično intuitivan i dobro balansiran, te je sa minimalnom količinom koda moguće odraditi kompleksne zadaće. U verziji 5.2 su mnoge stvari promijenjene na bolje u odnosu na Laravel 4, te je implementirana automatska podrška za poznate frontend razvojne okvire, poput Bootstrapa. Jedna od najvećih mana mu je ipak izbacivanje nativne podrške za *collective* forme.

6. Zaključak

MVC arhitektura jest dominantan koncept u pisanju web aplikacija i ne naziru se neke bitne promjene po tome pitanju. No pitanje jest je li najbolja za PHP ili samo najprihvatljivija? O tome su mišljenja podijeljena unutar programerske zajednice. Čak i sam Rasmus Lerdorf smatra da je uopće smisao postojanja PHP-a razbijanje forme u korist slobode i više intuitivnijem pristupu rješavanju problema. Na kraju krajeva, programiranje nije ništa drugo doli premošćivanje postojećih zadataka i problema, te kompenzacija za buduće. MVC je dobar koncept, iako u svojem šturom obliku više odgovara desktop software-u. Web je sklon stalnim promjenama, sigurnosnim rizicima, podilaženju korisnicima. Model-View-Controller arhitektura ima zadaću donijeti red u pomalo kaotičan svijet internet aplikacija.

U tom nekom sukobu uniformiranosti i improvizacije, MVC razvojni okviri, poput Laravela, dođu nešto kao srednji put. S jedne strane koriste strukturalnu organiziranost MVC-a, dok s druge strane preskaču beskrajno nadmudrivanje sa klasama objektno orijentiranog programiranja, te se programer više može posvetiti rješavanju problema.

Sam PHP je s obzirom na druge programske jezike relativno kasno uveo MVC arhitekturu, te poprilično posuđuje iz, primjerice Jave, Ruby-ja ili Pythona. Zbog toga ima iznenađujuće malo literature što se tiče teme ovog rada. Većinom sam se služio instrukcijskim videima sa YouTube-a.

Budući da je tek krajem sljedeće godine izašla zadnja, i konačna, verzija PHP-a, još vrijeme treba pokazati na koji način će se to odraziti na MVC razvojne okvire kao što je Laravel. Za sada je prema prvim procjenama rad u frameworku ili CMS-u koji je baziran na PHP 7 dosta brži.

Kroz ovaj rad sam naučio puno toga o programiranju, budući da sam prethodno tome imao nešto školskog iskustva sa programskim jezikom C, te svojevoljno sa PHP-om 4. Takoreći sam učio jezik iz početka, a posebice što se tiče objektno orijentiranog programiranja, na kojem je MVC arhitektura bazirana. Zahvaljujući mentorovom prijedlogu teme, imao sam priliku učiti raditi u PHP razvojnom okviru kao što je Laravel, što mi bude olakšalo migraciju u neko drugo programersko okruženje. MVC arhitektura ima budućnost u PHP-u, a nadam se i u mojem profesionalnom životu, budući da sam kroz ovaj rad i studij Multimedije, oblikovanja i primjene na sveučilištu „Sjever“ stekao dobar set vještina za profesionalni razvoj web aplikacija.

U Varaždinu, srpanj 2016.

vlastoručan potpis studenta

UNIVERSITY NORTH

Sveučilište Sjever

VŽKC

MMI

SVEUČILIŠTE SJEVER

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, Ivan Štaba pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor završnog rada pod naslovom "MVC oblikovni obrasci u PHP razvojnom okviru Laravel" te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student:
Ivan Štaba

Štaba Ivan

(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, Ivan Štaba neopozivo izjavljujem da sam suglasan s javnom objavom završnog rada pod "MVC oblikovni obrasci u PHP razvojnom okviru Laravel" čiji sam autor.

Student:
Ivan Štaba

Štaba Ivan

(vlastoručni potpis)

Literatura i linkovi

1. http://www.advogato.org/article/470.html#3
2. https://www.youtube.com/watch?v=YIGRXEzjE6c
3. https://en.wikipedia.org/wiki/Mosaic_(web_browser)
4. http://www.gnu.org/philosophy/free-software-for-freedom.html
5. https://en.wikipedia.org/wiki/LAMP_(software_bundle)#WAMP
6. https://www.apachefriends.org/index.html
7. http://httpd.apache.org/
8. http://whatis.techtarget.com/definition/Web-server
9. https://www.scriptrock.com/articles/iis-apache
10. https://en.wikipedia.org/wiki/HipHop_for_PHP
11. https://coderfactory.com/posts/top-sites-built-with-php
12. http://eev.ee/blog/2012/04/09/php-a-fractal-of-bad-design/
13. http://php.net/manual/en/book.mysqli.php
14. http://php.net/manual/en/history.php.php
15. https://www.zend.com/en/community/php
16. http://www.techotopia.com/index.php/The_History_of_PHP
17. https://en.wikipedia.org/wiki/PHP
18. https://blog.engineyard.com/2014/php-5-10th-anniversary
19. http://php.net/manual/en/language.oop5.autoload.php
20. http://php.net/manual/en/phar.using.intro.php
21. http://php.net/manual/en/intro.opcache.php
22. Speeding up the Web with PHP 7 - Rasmus Lerdorf
23. http://buytaert.net/the-history-of-mysql-ab
24. https://en.wikipedia.org/wiki/MySQL
25. http://www.hongkiat.com/blog/best-php-frameworks/
26. https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/
27. https://laravel.com/docs/5.2/eloquent
28. https://laravel.com/docs/5.2/validation
29. http://parsleyjs.org/
30. http://getbootstrap.com/
31. https://laravel.com/docs/5.0/templates
32. http://php.net/manual/en/function.date.php
33. http://php.net/manual/en/function.strtotime.php
34. https://laravelcollective.com/docs/5.2/html
35. Mindspace YouTube tutorijali
36. http://laravel-recipes.com/
37. https://github.com/laravel/laravel
38. Jacurtis YouTube tutorijali
39. Matt Staufer: Laravel- Up & Runnig, O'Reilly, 2016.

40. Taylor Otwell: Laravel: From Apprentice To Artisan, Leanpub, 2013.
41. Dayle Rees: Laravel: Code Smart. The Laravel Framework Version 5 for Beginners, Leanpub, 2016.

Fotografije

Slika 1. Rasmus Lerdorf, https://www.flickr.com/photos/x-foto/4926322352

Prilozi

Model

Post.php

```
namespace App;
use Illuminate\Database\Eloquent\Model;
class Post extends Model{
   public function user(){
       return $this->belongsTo('App\User');
   }
}
```

User.php

```
namespace App;
use Illuminate\Foundation\Auth\User as Authenticatable;
class User extends Authenticatable {
      protected $fillable = [
      'name', 'email', 'password',
      ];
      protected $hidden = [
      'password', 'remember_token',
      ];
      public function posts(){
      return $this->hasMany('App\Post');
   }
}
```

Controller

AdminController.php

```
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use App\Http\Requests;
use App\User;
use App\Post;
use Session;
```

```
class AdminController extends Controller {
   public function __construct(){
      $this->middleware('admin');
   }
   public function index() {
      $users = User::orderBy('id', 'asc')->paginate(20);
      return view('admin.index')->withUsers($users);
   }
   public function create() {
      return view('admin.create');
   }
   public function store(Request $request) {
         $this->validate($request, array(
         'name' => 'required|max:150',
         'degree' => 'required',
         'position' => 'required',
         'email' => 'required|unique:users',
         'phone' => 'required',
         'username' => 'required|min:3|max:20|unique:users',
         'admin' => 'required',
         'password' => 'required|min:5|max:30'
      ));
      $user = new User;
      $user->name = $request->name;
      $user->degree = $request->degree;
      $user->position = $request->position;
      $user->email = $request->email;
      $user->phone = $request->phone;
      $user->username = $request->username;
      $user->admin = $request->admin;
      $user->password = bcrypt($request->password);
      $user->save();
      Session::flash('success', 'Korisnik je uspješno kreiran!');
      return redirect()->route('admin.index');
   }
   public function show($id) {
      $user = User::find($id);
      if($user){
```

```
    $posts = Post::where('user_id', '=', $user->id)-
    >orderBy('created_at', 'asc')->paginate(30);
    return view('admin.show')->withUser($user)-
    >withPosts($posts);
    }
}
public function edit($id) {
    $user = User::find($id);
    return view('admin.edit')->withUser($user);
}
public function update(Request $request, $id) {
    $user = User::find($id);
    if($request->input('email') === $user->email && $request-
    >input('username') === $user->username) {
        $this->validate($request, array(
            'name' => 'required|max:150',
            'degree' => 'required',
            'position' => 'required',
            'email' => 'required|email',
            'phone' => 'required',
            'username' => 'required|min:3|max:20',
            'admin' => 'required',
            'password' => 'required|min:5|max:30'
        ));
} else if($request->input('email') === $user->email) {
    $this->validate($request, array(
        'name' => 'required|max:150',
        'degree' => 'required',
        'position' => 'required',
        'email' => 'required|email',
        'phone' => 'required',
        'username' => 'required|unique:users|min:3|max:20',
        'admin' => 'required',
        'password' => 'required|min:5|max:30'
    ));
} else if($request->input('username') === $user->username){
    $this->validate($request, array(
        'name' => 'required|max:150',
        'degree' => 'required',
```

```
            'position' => 'required',
            'email' => 'required|email|unique:users',
            'phone' => 'required',
            'username' => 'required|min:3|max:20',
            'admin' => 'required',
            'password' => 'required|min:5|max:30'
        ));
    } else {
        $this->validate($request, array(
            'name' => 'required|max:150',
            'degree' => 'required',
            'position' => 'required',
            'email' => 'required|email|unique:users',
            'phone' => 'required',
            'username' => 'required|unique:users|min:3|max:20',
            'admin' => 'required',
            'password' => 'required|min:5|max:30'
        ));
    }
        $user->name = $request->name;
        $user->degree = $request->degree;
        $user->position = $request->position;
        $user->email = $request->email;
        $user->phone = $request->phone;
        $user->username = $request->username;
        $user->admin = $request->admin;
        $user->password = bcrypt($request->password);
        $user->update();
        Session::flash('success', 'Korisniku su uspješno nadograđeni
        podaci!');
        return redirect()->route('admin.index');
    }
    public function destroy($id) {
        $user = User::find($id);
        $user->delete();
        Session::flash('success', 'Korisnik je uspješno izbrisan.');
        return redirect()->route('admin.index');
    }
}
```

PagesController.php

```
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Http\Requests;
class PagesController extends Controller{
   public function getIndex(){
       return view('pages.index');
   }
   public function getAbout(){
       return view('pages.about');
   }
}
```

PostController.php

```
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Http\Requests;
use Auth;
use App\User;
use App\Post;
use Session;
class PostController extends Controller{
   public function __construct(){
      $this->middleware('auth');
   }
   public function create(){
      $user = Auth::user();
      return view('post.obrazac')->withUser($user);
   }
   public function store(Request $request){
      $this->validate($request, array(
         'polaziste' => 'required',
         'odrediste' => 'required',
         'km' => 'required|numeric',
         'naknada' => 'required|numeric'
      ));
      $user = Auth::user();
      $post = new Post;
      $post->polaziste = $request->polaziste;
      $post->odrediste = $request->odrediste;
      $post->km = $request->km;
      $post->naknada = $request->naknada;
      $post->iznos = $request->km * $request->naknada;
      $post->user_id = $user->id;
      $post->save();
      Session::flash('success', 'Vaši putni troškovi su uspješno
      spremljeni.');
      return redirect()->back()->withUser($user);
   }
}
```

UserController.php

```
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Http\Requests;
use Auth;
use App\Post;
use App\User;
class UserController extends Controller{
   public function show($id){
       $user = Auth::user($id);
       if($user){
           $posts = Post::where('user_id', '=', $user->id)-
           >orderBy('created_at', 'asc')->paginate(30);
           return view('user.profile')->withUser($user)-
           >withPosts($posts);
       }
   }
}
```

Middleware

Admin.php

```
namespace App\Http\Middleware;
use Closure;
use Illuminate\Support\Facades\Auth;
class Admin{
   public function handle($request, Closure $next){
       if (!Auth::guest() && Auth::user()->admin === 1) {
           return $next($request);
       }
       return redirect()->back();
   }
}
```

Authenticate.php

```
namespace App\Http\Middleware;
use Closure;
use Illuminate\Support\Facades\Auth;
class Authenticate{
   public function handle($request, Closure $next, $guard = null){
```

```
        if (Auth::guard($guard)->guest()) {
            if ($request->ajax() || $request->wantsJson()) {
            return response('Unauthorized.', 401);
        } else {
            return redirect()->guest('auth/login');
        }
        }
        return $next($request);
    }
}
```

Routes

```
Route::get('auth/login', ['as' => 'login', 'uses' =>
'Auth\AuthController@getLogin']);

Route::post('auth/login', 'Auth\AuthController@postLogin');

Route::get('auth/logout', ['as' => 'logout', 'uses' =>
'Auth\AuthController@getLogout']);

Route::resource('post', 'PostController', ['only' => ['create',
'store']]);

Route::resource('user', 'UserController', ['only' => ['show']]);

Route::resource('admin', 'AdminController');

Route::get('about', 'PagesController@getAbout');

Route::get('/', 'PagesController@getIndex');
```

View

main.blade.php

```
<!DOCTYPE html>
<html lang="hr">
   <head>
      @include('partials._head')
   </head>
   <body>
      @include('partials._nav')
      <div class="container">
         @include('partials._messages')
         @yield('content')
         @include('partials._footer')
      </div>
      @include('partials._js')
      @yield('scripts')
   </body>
</html>
```

admin/create.blade.php

```
@extends('main')
@section('title', '| Kreiranje novog korisnika')
@section('stylesheets')
   {!! Html::style('css/parsley.css') !!}
@endsection
@section('content')
   <div class="row">
     <div class="col-md-8 col-md-offset-2">
       <div class="formular-head">Novi korisnik</div>
       <div class="formular-body">
         {!! Form::open(['route' => 'admin.store', 'data-parsley-
         validate' => '']) !!}
         {{ Form::label('name', 'Prezime i ime:') }}
         {{ Form::text('name', null, ['class' => 'form-control',
         'required' => '']) }}
         {{ Form::label('degree', 'Akademska titula:') }}
         {{ Form::select('degree', array(
         'akad. slikar' => 'akad. slikar',
```

```
'akad. slikar - grafičar' => 'akad. slikar - grafičar',
'bacc. med. techn.' => 'bacc. med. techn.',
'bacc. prim.' => 'bacc. prim.',
'dipl. def.' => 'dipl. def.',
'dipl. diz.' => 'dipl. diz.',
'dipl. inf.' => 'dipl. inf',
'dipl. ing.' => 'dipl. ing',
'dipl. iur.' => 'dipl. iur.',
'dipl. mag. techn.' => 'dipl. mag. techn.',
'dipl. med. techn.' => 'dipl. med. techn.',
'dipl. nov.' => 'dipl. nov.',
'dipl. oec.' => 'dipl. oec.',
'dipl. učit.' => 'dipl. učit.',
'dr. med.' => 'dr. med.',
'dr. phil.' => 'dr. phil.',
'dr. sc.' => 'dr. sc.',
'mag. educ. mat. et phys.' => 'mag. educ. mat. et phys.',
'mag. inf.' => 'mag. inf.',
'mag. ing. techn. graph.' => 'mag. ing. techn. graph.',
'mag. ing. mech.' => 'mag. ing. mech.',
'mag. ing. inf. et comm. techn.' => 'mag. ing. inf. et
comm. techn.',
'mag. med. techn.' => 'mag. med. techn.',
'mag. nov.' => 'mag. nov.',
'mag. oec.' => 'mag. oec.',
'mag. philol.' => 'mag. philol.',
'mag. philol. angl.' => 'mag. philol. ang.',
'mag. politolog.' => 'mag. politolog.',
'mag. prim. educ.' => 'mag. prim. educ.',
'mag. psih.' => 'mag. psih.',
'mr. sc.' => 'mr. sc.',
'prof.' => 'prof.',
'struc. spec. ing. tech. inf.' => 'struc. spec. ing. tech.
inf.',
'struč. spec. ing. el.' => 'struč. spec. ing. el.',
'struč. spec. ing. građ.' => 'struč. spec. ing. građ.',
'student' => 'student',
'univ. spec. oec.' => 'univ. spec. oec.'
```

```
    ), null, ['class' => 'form-control', 'placeholder' =>
    '...', 'required' => '']) }}

    {{ Form::label('position', 'Zvanje:') }}

    {{ Form::select('position', array(

    'Asistent' => 'Asistent',

    'Docent' => 'Docent',

    'Izvanredni profesor' => 'Izvanredni profesor',

    'Mentor vježbovne nastave' => 'Mentor vježbovne nastave',

    'Predavač' => 'Predavač',

    'Profesor visoke škole' => 'Profesor visoke škole',

    'Profesor visoke škole u trajnom zvanju' => 'Profesor
    visoke škole u trajnom zvanju',

    'Redoviti profesor' => 'Redoviti profesor',

    'Redoviti profesor u trajnom zvanju' => 'Redoviti profesor
    u trajnom zvanju',

    'Student' => 'Student',

    'Viši predavač' => 'Viši predavač',

    'Viši predavač - Docent' => 'Viši predavač - Docent'

    ), null, ['class' => 'form-control', 'placeholder' =>
    '...', 'required' => '']) }}

    {{ Form::label('email', 'Email:') }}

    {{ Form::email('email', null, ['class' => 'form-control',
    'required' => '', 'type' => 'email']) }}

    {{ Form::label('phone', 'Broj telefona:') }}

    {{ Form::text('phone', null, ['class' => 'form-control',
    'required' => '']) }}

    {{ Form::label('username', 'Korisničko ime:') }}

    {{ Form::text('username', null, ['class' => 'form-
    control', 'required' => '', 'minlength' => '6']) }}

    {{ Form::label('admin', 'Administratorska prava:') }}

    {{ Form::select('admin', array('0' => 'Ne','1' => 'Da'),
    null, ['class' => 'form-control']) }}

    {{ Form::label('password', 'Lozinka:') }}

    {{ Form::password('password', ['class' => 'form-control',
    'required' => '']) }}

    {{ Form::submit('Registriraj', ['class' => 'btn btn-
    success btn-lg btn-block', 'style' => 'margin-top:20px'])
    }}

    {!! Form::close() !!}

  </div>

</div>
```

```
    </div>
@endsection
@section('scripts')
    {!! Html::script('js/parsley.min.js') !!}
@endsection
```

admin/edit.blade.php

```
@extends('main')
@section('title', '| Editiranje korisnika')
@section('stylesheets')
    {!! Html::style('css/parsley.css') !!}
@endsection
@section('content')
    <div class="row">
        <div class="col-md-8 col-md-offset-2">
        <div class="formular-head">Editiranje korisnika</div>
            <div class="formular-body">
                {!! Form::open([$user, 'route' => ['admin.update', $user-
                >id], 'method' => 'PUT', 'data-parsley-validate' => ''])
                !!}
                {{ Form::label('name', 'Prezime i ime:') }}
                {{ Form::text('name', $user->name, ['class' => 'form-
                control', 'required' => '']) }}
                {{ Form::label('degree', 'Akademska titula:') }}
                {{ Form::select('degree', array(
                'akad. slikar' => 'akad. slikar',
                'akad. slikar - grafičar' => 'akad. slikar - grafičar',
                'bacc. med. techn.' => 'bacc. med. techn.',
                'bacc. prim.' => 'bacc. prim.',
                'dipl. def.' => 'dipl. def.',
                'dipl. diz.' => 'dipl. diz.',
                'dipl. inf.' => 'dipl. inf',
                'dipl. ing.' => 'dipl. ing',
                'dipl. iur.' => 'dipl. iur.',
                'dipl. mag. techn.' => 'dipl. mag. techn.',
                'dipl. med. techn.' => 'dipl. med. techn.',
                'dipl. nov.' => 'dipl. nov.',
                'dipl. oec.' => 'dipl. oec.',
                'dipl. učit.' => 'dipl. učit.',
```

```
'dr. med.' => 'dr. med.',
'dr. phil.' => 'dr. phil.',
'dr. sc.' => 'dr. sc.',
'mag. educ. mat. et phys.' => 'mag. educ. mat. et phys.',
'mag. inf.' => 'mag. inf.',
'mag. ing. techn. graph.' => 'mag. ing. techn. graph.',
'mag. ing. mech.' => 'mag. ing. mech.',
'mag. ing. inf. et comm. techn.' => 'mag. ing. inf. et
comm. techn.',
'mag. med. techn.' => 'mag. med. techn.',
'mag. nov.' => 'mag. nov.',
'mag. oec.' => 'mag. oec.',
'mag. philol.' => 'mag. philol.',
'mag. philol. angl.' => 'mag. philol. ang.',
'mag. politolog.' => 'mag. politolog.',
'mag. prim. educ.' => 'mag. prim. educ.',
'mag. psih.' => 'mag. psih.',
'mr. sc.' => 'mr. sc.',
'prof.' => 'prof.',
'struc. spec. ing. tech. inf.' => 'struc. spec. ing. tech.
inf.',
'struč. spec. ing. el.' => 'struč. spec. ing. el.',
'struč. spec. ing. građ.' => 'struč. spec. ing. građ.',
'student' => 'student',
'univ. spec. oec.' => 'univ. spec. oec.'
), $user->degree, ['class' => 'form-control', 'required'
=> '']) }}
{{ Form::label('position', 'Zvanje:') }}
{{ Form::select('position', array(
'Asistent' => 'Asistent',
'Docent' => 'Docent',
'Izvanredni profesor' => 'Izvanredni profesor',
'Mentor vježbovne nastave' => 'Mentor vježbovne nastave',
'Predavač' => 'Predavač',
'Profesor visoke škole' => 'Profesor visoke škole',
'Redoviti profesor' => 'Redoviti profesor',
'Redoviti profesor u trajnom zvanju' => 'Redoviti profesor
u trajnom zvanju',
'Student' => 'Student',
```

```
            'Viši predavač' => 'Viši predavač',
            'Viši predavač - Docent' => 'Viši predavač - Docent'
            ), $user->position, ['class' => 'form-control', 'required'
            => '']) }}
            {{ Form::label('email', 'Email:') }}
            {{ Form::email('email', $user->email, ['class' => 'form-
            control', 'required' => '', 'type' => 'email']) }}
            {{ Form::label('phone', 'Broj telefona:') }}
            {{ Form::text('phone', $user->phone, ['class' => 'form-
            control', 'required' => '']) }}
            {{ Form::label('username', 'Korisničko ime:') }}
            {{ Form::text('username', $user->username, ['class' =>
            'form-control', 'required' => '', 'minlength' => '6']) }}
            {{ Form::label('admin', 'Administratorska prava:') }}
            {{ Form::select('admin', array('0' => 'Ne','1' => 'Da'),
            $user->admin, ['class' => 'form-control']) }}
            {{ Form::label('password', 'Lozinka:') }}
            {{ Form::password('password', ['class' => 'form-control',
            'required' => '']) }}
            {{ Form::submit('Završi editiranje', ['class' => 'btn btn-
            success btn-lg btn-block', 'style' => 'margin-top:20px'])
            }}
            {!! Form::close() !!}
        </div>
      </div>
    </div>
@endsection
@section('scripts')
    {!! Html::script('js/parsley.min.js') !!}
@endsection
```

admin/indeks.blade.php

```
@extends('main')
@section('title', '| Administrator')
@section('content')
   <div class="row">
      <div class="col-md-12">
         <div class="col-md-8">
            <h1>Administrator</h1>
         </div>
         <div class="col-md-3 col-md-offset-1">
            <a href="{{ route('admin.create') }}" class="btn btn-
            primary btn-block btn-lg" style="margin-top: 30px"><i
            class="fa fa-user-plus" aria-hidden="true"></i>Dodaj
            novog korisnika</a>
         </div>
      </div>
   </div>
   <hr>
   <div class="row">
      <div class="col-md-12">
         <h3>Popis korisnika</h3>
         <table class="table table-bordered">
         <thead>
         <th>Id</th>
         <th>Prezime i ime</th>
         <th>Ak. stupanj</th>
         <th>Zvanje</th>
         <th>Email</th>
         <th>Tel. broj</th>
         <th>Kor. ime</th>
         <th>Admin</th>
         </thead>
         <tbody>
         @foreach($users as $user)
            <tr>
            <th>{{ $user->id }}</th>
            <td>{{ $user->name }}</td>
            <td>{{ $user->degree }}</td>
            <td>{{ $user->position }}</td>
```

```
            <td>{{ $user->email }}</td>
            <td>{{ $user->phone }}</td>
            <td>{{ $user->username }}</td>
            <td>{{ ($user->admin === 1) ? 'Da' : 'Ne' }}</td>
            <td><a href="{{ route('admin.show', $user->id) }}"
            class="btn btn-success">Korisnik</a></td>
            <td><a href="{{ route('admin.edit', $user->id) }}"
            class="btn btn-primary">Editiraj</a></td>
            </tr>
        @endforeach
        </tbody>
        </table>
        <div class="text-center paginacija">
            {!! $users->links() !!}
        </div>
      </div>
   </div>
@endsection
```

admin/show.blade.php

```
@extends('main')
@section('title', "| $user->name" )
@section('content')
   <div class="row">
     <div class="col-md-12">
     <div class="panel panel-primary">
       <div class="panel-heading">
         <h3>{{ $user->name }}</h3>
       </div>
       <div class="panel-body">
         <div class="col-md-8">
         <div class="dl-horizontal">
           <dt>Ime i prezime:</dt>
           <dd>{{ $user->name }}</dd>
           <dt>Akademski stupanj:</dt>
           <dd>{{ $user->degree }}</dd>
           <dt>Zvanje:</dt>
           <dd>{{ $user->position }}</dd>
           <dt>Email:</dt>
```

```
            <dd>{{ $user->email }}</dd>
            <dt>Telefonski broj:</dt>
            <dd>{{ $user->phone }}</dd>
            <dt>Korisničko ime:</dt>
            <dd>{{ $user->username }}</dd>
            <dt>Administrator:</dt>
            <dd>{{ ($user->admin === 1) ? 'Da' : 'Ne' }}</dd>
          </div>
        </div>
        <div class="col-md-4">
          <div class="well">
            <dl>
            <dt>Korisnik je kreiran:</dt>
            <dd>{{ date('M j, Y', strtotime($user->created_at))
            }}</dd>
            <dt>Korisnički podaci su nadograđeni:</dt>
            <dd>{{ date('M j, Y', strtotime($user->updated_at))
            }}</dd>
            </dl>
            <div class="row">
            <div class="col-sm-6">
            {!! Html::linkRoute('admin.edit', 'Editiraj', [$user-
            >id], ['class' => 'btn btn-primary btn-block']) !!}
          </div>
          <div class="col-sm-6">
            {!! Form::open(['route' => ['admin.destroy', $user-
            >id], 'method' => 'DELETE']) !!}
            {!! Form::submit('Izbriši', ['class' => 'btn btn-danger
            btn-block']) !!}
            {!! Form::close() !!}
          </div>
        </div>
        <div class="row">
        <div class="col-md-12">
          {{ Html::linkRoute('admin.index', '<< Povratak na početnu
          stranicu', [], ['class' => 'btn btn-default btn-block btn-
          h1-spacing']) }}
        </div>
      </div>
    </div>
```

```
    </div>
  </div>
</div>
<div class="panel panel-success">
  <div class="panel-heading">
    <h3>Izvadak putnih troškova</h3>
  </div>
  <div class="panel-body">
    <div class="col-md-8 col-md-offset-2">
      <table class="table table-bordered">
      <thead>
      <th>Id</th>
      <th>Datum</th>
      <th>Polazište</th>
      <th>Odredište</th>
      <th>Km</th>
      <th>Kn/km</th>
      <th>Ukupno (kn)</th>
      </thead>
      <tbody>
      @foreach($posts as $post)
        <tr>
        <th>{{ $post->user_id }}</th>
        <th>{{ date('M j, Y', strtotime($post->created_at))
        }}</th>
        <td>{{ $post->polaziste }}</td>
        <td>{{ $post->odrediste }}</td>
        <td>{{ $post->km }}</td>
        <td>{{ $post->naknada }}</td>
        <th>{{ $post->iznos }}</th>
        </tr>
      @endforeach
      </tbody>
      </table>
      <div class="text-center paginacija">
        {!! $posts->links() !!}
      </div>
    </div>
```

```
        </div>
        </div>
      </div>
    </div>
@stop()
```

auth/login.blade.php

```
@extends('main')
@section('title', '| Logiranje korisnika')
@section('stylesheets')
   {!! Html::style('css/parsley.css') !!}
@endsection
@section('content')
   <div class="row">
      <div class="col-md-6 col-md-offset-3">
      <div class="panel panel-primary" id="login-panel">
         <div class="panel-heading">
            <h1 class="text-center"><i class="fa fa-lock fa-2x">
            Login</i></h1>
         </div>
         <hr>
         <div class="panel-body">
            {!! Form::open(['data-parsley-validate' => '']) !!}
            {{ Form::label('email', 'Email:') }}
            {{ Form::email('email', null, ['class' => 'form-
            control', 'required' => '']) }}
            {{ Form::label('password', 'Lozinka:') }}
            {{ Form::password('password', ['class' => 'form-
            control', 'required' => '']) }}
            <br>
            {{ Form::submit('Ulogiraj se', ['class' => 'btn btn-
            primary btn-block']) }}
            {!! Form::close() !!}
         </div>
         </div>
      </div>
   </div>
@endsection
@section('scripts')
```

```
    {!! Html::script('js/parsley.min.js') !!}
@endsection
```

pages/about.blade.php

```
@extends('main')
@section('title', ' | O programu')
@section('content')
    <div class="row">
        <div class="column-md-12">
            <h1>O programu</h1>
            <p>Program za izračun putnih troškova. Praktični dio
            završnog rada iz predmeta "Programski alati 3".</p>
        </div>
    </div>
@endsection
```

pages/indeks.blade.php

```
@extends('main')
@section('title', ' | Naslovna')
@section('content')
    <div class="row">
        <div class="col-md-12">
            <div class="jumbotron">
                <img src="/images/putni_troskovi.gif" alt="Putni
                Troškovi">
            </div>
        </div>
    </div> <!-- end of header .row -->
@endsection
```

partials/_footer.blade.php

```
<div class="footer navbar-inverse navbar-fixed-bottom">
   <p class="text-center">Copyright <i class="fa fa-copyright"
   aria-hidden="true"></i> Ivan Štaba. Multimedija, oblikovanje i
   primjena. Sveučilište "Sjever"</p>
</div>
```

partials/_head.blade.php

```
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-
scale=1">
<title>Troškovi @yield('title')</title>
<!-- Latest compiled and minified CSS -->
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstra
p.min.css" integrity="sha384-
1q8mTJOASx8j1Au+a5WDVnPi2lkFfwwEAa8hDDdjZlpLegxhjVME1fgjWPGmkzs7"
crossorigin="anonymous">
{{ Html::style('css/styles.css') }}
@yield('stylesheets')
```

partials/_js.blade.php

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.mi
n.js"></script>
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.
min.js" integrity="sha384-
0mSbJDEHialfmuBBQP6A4Qrprq5OVfW37PRR3j5ELqxss1yVqOtnepnHVP9aJ7xS"
crossorigin="anonymous"></script>
<script src="https://use.fontawesome.com/f78be9d52b.js"></script>
```

partials/_messages.blade.php

```
@if (Session::has('success'))
   <div class="alert alert-success" role="alert">
      <strong>Success:</strong> {{ Session::get('success') }}
   </div>
@endif
@if (count($errors) > 0)
   <div class="alert alert-danger" role="alert">
      <strong>Errors:</strong>
      <ul>
      @foreach($errors->all() as $error)
         <li>{{ $error }}</li>
      @endforeach
      </ul>
   </div>
@endif
```

partials/_nav.blade.php

```
<nav class="navbar navbar-inverse">
   <div class="container-fluid">
      <!-- Brand and toggle get grouped for better mobile display -
      ->
      <div class="navbar-header">
         <button type="button" class="navbar-toggle collapsed"
         data-toggle="collapse" data-target="#bs-example-navbar-
         collapse-1" aria-expanded="false">
         <span class="sr-only">Toggle navigation</span>
         <span class="icon-bar"></span>
         <span class="icon-bar"></span>
         <span class="icon-bar"></span>
         </button>
         <a class="navbar-brand" href="{{ url('/') }}"
         style="color:#21b384;"><i class="fa fa-plane" aria-
         hidden="true"></i> Putni troškovi</a>
      </div>
      <!-- Collect the nav links, forms, and other content for
      toggling -->
      <div class="collapse navbar-collapse" id="bs-example-navbar-
      collapse-1">
         <ul class="nav navbar-nav">
```

```
            <li class="{{ Request::is('/') ? "active" : "" }}"><a
            href="/">Naslovna </a></span></li>

            <li class="{{ Request::is('about') ? "active" : "" }}"><a
            href="/about">O programu</a></li>

            </ul>

            <ul class="nav navbar-nav navbar-right">

            <li class="dropdown">

            @if(Auth::check())

              <a href="#" class="dropdown-toggle" data-
              toggle="dropdown" role="button" aria-haspopup="true"
              aria-expanded="false">{{ Auth::user()->name }} <span
              class="caret"></span></a>

            @else

              <a href="#" class="dropdown-toggle" data-
              toggle="dropdown" role="button" aria-haspopup="true"
              aria-expanded="false"><i class="fa fa-shield" aria-
              hidden="true"></i> Korisnički izbornik <span
              class="caret"></span></a>

            @endif

            <ul class="dropdown-menu">

            @if(Auth::check())

              <li><a href="{{ route('user.show', auth()->user()->id)
              }}"><i class="fa fa-user" aria-hidden="true"></i>
              Profil</a></li>

              <li><a href="{{ route('admin.index') }}"><i class="fa
              fa-key" aria-hidden="true"></i> Administrator</a>

            @else

              <li><a href="{{ route('login') }}"><i class="fa fa-
              sign-in" aria-hidden="true"></i> Login</a></li>

            @endif

            <li role="separator" class="divider"></li>

            <li><a href="{{ route('logout') }}"><i class="fa fa-sign-
            out" aria-hidden="true"></i> Logout</a></li>

            </ul>

            </li>

            </ul>

        </div><!-- /.navbar-collapse -->

    </div><!-- /.container-fluid -->

</nav>
```

post/obrazac.blade.php

```
@extends('main')
@section('title', '| Obrazac za unos putnih troškova')
@section('stylesheets')
   {!! Html::style('css/parsley.css') !!}
@endsection
@section('content')
   <div class="row">
      <div class="col-md-6 col-md-offset-3">
         <div class="formular-head"><p>Obrazac za upis
         troškova</p></div>
         <div class="formular-body">
            {!! Form::open(['route' => 'post.store', 'data-
            parsley-validate' => '']) !!}
            {{ Form::label('polaziste', 'Polazište:') }}
            {{ Form::text('polaziste', null, ['class' => 'form-
            control', 'required' => '']) }}
            {{ Form::label('odrediste', 'Odredište:') }}
            {{ Form::text('odrediste', null, ['class' => 'form-
            control', 'required' => '']) }}
            {{ Form::label('km', 'Broj kilometara:') }}
            {{ Form::text('km', null, ['class' => 'form-control',
            'required' => '']) }}
            {{ Form::label('naknada', 'Kn/km:') }}
            {{ Form::text('naknada', null, ['class' => 'form-
            control', 'required' => '']) }}
            <hr>
            {{ Form::submit('Spremi', ['class' => 'btn btn-success
            btn-lg btn-block', 'style' => 'margin-top:20px']) }}
            {!! Form::close() !!}
            <a href="{{ url('user/{user}') }}" class="btn btn-
            primary btn btn-block btn-lg" style="margin-top:
            20px"><< Natrag</a>
         </div>
      </div>
   </div>
@endsection
@section('scripts')
   {!! Html::script('js/parsley.min.js') !!}
@endsection
```

user/profile.blade.php

```
@extends('main')
@section('title', "| $user->name")
@section('content')
  <div class="row">
    <div class="col-md-12">
      <div class="panel panel-primary">
        <div class="panel-heading">
          <h3>{{ $user->name }}</h3>
        </div>
        <div class="panel-body">
          <div class="col-md-8">
          <div class="dl-horizontal">
          <dt>Ime i prezime:</dt>
          <dd>{{ $user->name }}</dd>
          <dt>Akademski stupanj:</dt>
          <dd>{{ $user->degree }}</dd>
          <dt>Zvanje:</dt>
          <dd>{{ $user->position }}</dd>
          <dt>Email:</dt>
          <dd>{{ $user->email }}</dd>
          <dt>Telefonski broj:</dt>
          <dd>{{ $user->phone }}</dd>
          <dt>Korisničko ime:</dt>
          <dd>{{ ($user->admin === 1) ? $user->username :
          'Korisnik nije Administrator' }}</dd>
          </div>
          </div>
          <div class="row">
          <div class="col-md-6 col-md-offset-3">
          <hr>
          <a href="{{ route('post.create') }}" class="form-
          control btn btn-success"><i class="fa fa-archive" aria-
          hidden="true"></i> Obrazac za upis putnih troškova</a>
        </div>
      </div>
    </div>
  </div>
  <br>
```

```
  <hr>
  <div class="row text-center">
    <h1 class="ispis-head">Ispis putnih troškova</h1>
  </div>
  <div class="row tablica">
    <div class="col-md-8 col-md-offset-2">
        <table class="table table-bordered text-center">
          <thead>
          <th class="datum-head">Datum</th>
          <th>Polazište</th>
          <th>Odredište</th>
          <th>Km</th>
          <th>Kn/km</th>
          <th class="ukupno-head">Ukupno (kn)</th>
          </thead>
          <tbody>
          @foreach($posts as $post)
            <tr>
            <th class="datum-body">{{ date('M j, Y',
            strtotime($post->created_at)) }}</th>
            <td>{{ $post->polaziste }}</td>
            <td>{{ $post->odrediste }}</td>
            <td>{{ $post->km }}</td>
            <td>{{ $post->naknada }}</td>
            <th class="ukupno-body">{{ $post->iznos }}</th>
            </tr>
          @endforeach
          </tbody>
          </table>
          <div class="text-center paginacija">
            {!! $posts->links() !!}
          </div>
        </div>
      </div>
    </div>
  </div>
@endsection
```

CSS

parsley.css

```
input.parsley-success,
select.parsley-success,
textarea.parsley-success {
   color: #468847;
   background-color: #DFF0D8;
   border: 1px solid #D6E9C6;
}
input.parsley-error,
select.parsley-error,
textarea.parsley-error {
   color: #B94A48;
   background-color: #F2DEDE;
   border: 1px solid #ff0000;
}
.parsley-errors-list {
   margin: 2px 0 3px;
   padding: 0;
   list-style-type: none;
   font-size: 0.9em;
   line-height: 0.9em;
   opacity: 0;
   transition: all .3s ease-in;
   -o-transition: all .3s ease-in;
   -moz-transition: all .3s ease-in;
   -webkit-transition: all .3s ease-in;
}
.parsley-errors-list.filled {
   opacity: 1;
}
```

styles.css

```
.btn-h1-spacing{
  margin-top: 18px;
}
.form-spacing-top {
  margin-top: 30px;
}
.user-login {
  padding: 20px;
}
.dropdown-menu li a {
  color: #21b384;
}
.dropdown-menu li a:hover {
  color: whitesmoke;
  background-color: #21b384;
}
#login-panel {
  margin-top: 100px;
}
.ispis-head {
  color: #21b384;
}
.tablica {
  margin-top: 25px;
  margin-bottom: 50px;
}
.datum-head {
  color: aliceblue;
  background-color: #333;
}
.ukupno-head {
  color: aliceblue;
  background-color: #333;
}
.datum-body {
  color:darkgreen;
}
```

```
.ukupno-body {
   color:darkred;
}
th {
   text-align: center;
}
.formular-head {
   background-color: #be1e2d;
   color: #fff;
   margin-top: 40px;
   height: 80px;
   line-height: 80px;
   font-size: 25px;
   border-radius: 5px 5px 0 0;
   border: solid 5px #661e2d;
   text-align: center;
   border-bottom: none;
}
.formular-body {
   margin-top: 0;
   border: solid 5px #661e2d;
   border-top: none;
   padding: 15px;
   border-radius: 0 0 5px 5px;
}
.paginacija {
   margin-bottom: 50px;
}
.jumbotron {
   background-color: #fff;
}
.footer {
   height: 50px;
   padding: 1rem;
   color: #21b384;
}
```