

Univerzitet u Beogradu
Elektrotehnički fakultet

MASTER RAD

**Realizacija veb aplikacije za pronalaženje
odgovarajućeg majstora**

Mentor:

dr. Marija Punt, vanr. prof.

Kandidat:

Nikola Krstić 2022/3014

Beograd, 2024.

Majci i ocu,
da me ne pitaju više:
“Kad će master?”

Sadržaj

1	Uvod.....	1
2	Postojeće rešenje i predlog novog.....	3
2.1	Postojeća rešenja	3
2.2	Predlog novog rešenja	9
3	Tehnologije korišćene za izradu veb aplikacije.....	10
3.1	JavaScript.....	10
3.2	Angular JS.....	12
3.3	Java.....	15
3.4	SpringBoot.....	16
3.5	MySQL	18
3.6	Relacione baze podataka	19
4	Pregled detalja rešenja	20
4.1	Arhitektura sistema.....	20
4.2	Klasni dijagram sistema	22
4.3	JSON Web Token (JWT).....	23
4.4	Flyway migracije	26
4.5	Axios servis	27
4.6	Dvojezičnost.....	29
4.7	Učitavanje slike	30
5	Korisničko uputstvo.....	32
5.1	Neregistrovani korisnici	32
5.2	Registrovani korisnici	32
5.3	Prijavljivanje postojećih korisnika	33
5.4	Registracija korisnika	33
5.5	Pretraga majstora	34
5.6	Profil korisnika/majstora	35
5.7	Dovanje usluge i recenzije	36
5.8	Blogovi	37
6	Zaključak.....	38
	Literatura.....	40
	Spisak skraćenica	41
	Spisak slika.....	42
	Spisak tabela.....	44

1 Uvod

Ovaj master rad se bavi projektovanjem veb aplikacije za pronalaženje majstora pod nazivom „Majstor“. Potreba za ovom aplikacijom proizašla je iz sve većih poteškoća s kojima se suočavaju ljudi u velikim gradovima pri pronalaženju kvalitetnih i proverenih majstora. Aplikacija je namenjena korisnicima koji traže majstore i ima za cilj da olakša celokupan postupak pronalaženja majstora po meri. Jedan od ključnih problema koji aplikacija rešava je eliminacija rizika angažovanja nepouzdanih majstora, što je čest slučaj kada korisnici nemaju pouzdane reference.

Cilj ovog rada je realizacija internet aplikacije koja će poboljšati iskustvo korisnicima i omogućiti im jednostavan pristup visoko kvalitetnim majstorima sa proverenim referencama i ocenama, čime se osigurava pouzdanost i zadovoljstvo korisnika. Aplikacija će pružiti transparentnost korisnicima prikazivanjem detaljnih informacija o majstorima, njihove ocene, recenzije, cene i dostupnost, kako bi mogli doneti što bolje odluke. Pored toga, aplikacija će podstaći lokalnu ekonomiju povezujući korisnike sa majstorima iz njihove okoline, čime se dodatno doprinosi zajednici. Ona će pomoći i pridošlicama u velikim gradovima za pronalazak sigurnih majstora, bez potrebe za lokalnim poznanstvima ili referencama, čime se značajno smanjuje stres prilikom prilagođavanja novoj sredini. Svedoci smo da je otuđenost u društvu sve veća, ne samo kod novih članova zajednica, već i kod stalnih stanovnika. Ova aplikacija će stremiti ka tome da izgradi most između korisnika i majstora, stvarajući poverenje i olakšavajući svakodnevne životne situacije koje zahtevaju profesionalnu pomoć.

Ovaj rad će analizirati postojeća rešenja, sagledati njihove nedostatke, sa ciljem da dođe do optimalnog rešenja prateći najnovije svetske tehnologije. Aplikacija će biti realizovana korišćenjem *Angular* radnog okvira na klijentskoj i *SpringBoot* radnog okvira na serverskoj strani i kao grafički korisnički interfejs koristiće se *Intellij IDEA*. Za bazu podataka biće korišćen *MySQL*, čime se obezbeđuje stabilna i pouzdana arhitektura za skladištenje podataka. Pored osnovnih funkcionalnosti, posebna pažnja biće posvećena skalabilnosti i mogućnosti daljeg razvoja aplikacije, uključujući integraciju sa eksternim servisima za skladištenje podataka i autentifikaciju korisnika.

U glavi dva biće predstavljena postojeća rešenja za pronalaženje majstora sa svojim dobrim i lošim stranama. U nastavku druge glave biće opisano kako će se loše strane datih aplikacija otkloniti, koristeći inovativne pristupe i tehnološka rešenja koja će doprineti unapređenju korisničkog iskustva. Na osnovu analize, biće donete odluke o najboljim praksama koje će se implementirati u ovoj aplikaciji, sa ciljem da se obezbedi optimalno korisničko iskustvo.

U glavi tri će biti navedene i obrazložene tehnologije koje će se koristiti na oba dela ove aplikacije. Poseban akcenat će biti stavljen na sigurnost podataka, performanse aplikacije i jednostavnost korišćenja.

U glavi četiri, biće predstavljena arhitektura sistema pomoću klasnog dijagrama. Kompletna veb aplikacija je razdvojena u dva projekta, serverski deo projekta koji obezbeđuje usluge i klijentski deo projekta koji zahteva usluge i sadrži korisnički interfejs, dok je sam projekat deo većeg projekta koji sadrži i mobilnu aplikaciju. Ovaj pristup omogućava lakšu skalabilnost i održavanje sistema, čime se obezbeđuje dugoročna stabilnost aplikacije.

Biće predstavljeni neki od najinteresantnijih delova koda klijentskog, odnosno serverskog dela aplikacije: *JSON* token, *Flyway* migracije na serverskom delu, *Axios* servis, dvojezičnost na klijentskom delu i učitavna slika koje se prostire na oba dela aplikacije.

U glavi pet, u prva dva poglavlja će detaljno biti opisane kategorije korisnika dok će se u nastavku detaljno opisati svaka od funkcionalnosti aplikacije. Ovo poglavlje će pružiti uvid u različite korisničke scenarije, kao i način na koji aplikacija odgovara na specifične potrebe svakog tipa korisnika.

U glavi šest će biti dat zaključak, biće predstavljeni izazovi koji su se pojavili prilikom implementacije. Posebna pažnja biće posvećena planovima za budući razvoj aplikacije, uključujući potencijalne nove funkcionalnosti i tehnologije koje bi mogle unaprediti korisničko iskustvo i proširiti tržište aplikacije. Zaključak će takođe pružiti refleksiju o celokupnom procesu razvoja, uključujući lekcije koje su naučene i preporuke za buduće projekte slične prirode.

2 Postojeće rešenje i predlog novog

U ovoj glavi biće predstavljeno postojeća rešenje za pronalaženje majstora, a nakon toga biće dat predlog novog rešenja.

2.1 Postojeća rešenja

Profimajstor

Na sajtu „Profimajstor“ primećuje se značajan nedostatak u pogledu transparentnosti, jer klijenti nemaju mogućnost da sami odaberu majstora. Umesto toga, operateri sajta dodeljuju majstora po svom nahođenju, što može dovesti do različitih problema.

Ovakav pristup nije optimalan iz nekoliko ključnih razloga. Prvo, iz sigurnosne perspektive, klijenti ne znaju ko dolazi u njihovu kuću, što može izazvati nelagodu i nepoverenje. Drugo, nedostatak informacija o prethodnim radovima tog majstora, uključujući slike i ocene prethodnih klijenata, dodatno otežava klijentima da procene kvalitet usluge koju će dobiti.

Još jedan problem je što, kada se uđe u određenu kategoriju majstora, kao što je npr. moler, umesto konkretnih informacija o majstorima, dostupna su samo opšta mesta i floskule o tome šta taj majstor radi (Slika 1). Ovaj nedostatak specifičnih i relevantnih informacija dodatno smanjuje transparentnost i otežava klijentima donošenje informisane odluke.

Sve ove nedostatke je moguće iskoristiti kao osnovu za unapređenje aplikacije, omogućavajući klijentima da biraju majstore na osnovu detaljnih profila, ocena, i transparentnog prikaza njihovih prethodnih radova, čime bi se povećalo poverenje i zadovoljstvo korisnika.

VODOINSTALATERSKE USLUGE

Nudimo sveobuhvatne usluge, od instalacija do popravki, osiguravajući siguran i efikasan rad vaših sistema.

<ul style="list-style-type: none">✓ Odgušenje kanalizacione vertikalne ili horizontalne teže✓ Odgušenje kanalizacione vertikalne ili horizontalne lakše✓ Odgušenje mono bloka✓ Odgušenje WC šolje obične✓ Odgušenje sudoperske vertikalne ili horizontalne teže✓ Odgušenje sudoperske vertikalne ili horizontalne lakše✓ Odgušenje sudopere✓ Odgušenje umivaonika✓ Odgušenje podne rešetke✓ Odgušenje spoljnog slivnika✓ Izrada kupatila	<ul style="list-style-type: none">✓ Montaža pumpe za povećanje pritiska✓ Montaža umivaonika✓ Montaža umivaonika sa stalkom✓ Montaža zidne baterije za umivaonik✓ Montaža stojeće baterije za umivaonik✓ Montaža vodikotlića običnog✓ Montaža vodikotlića keramičkog✓ Montaža WC šolje✓ Montaža mono bloka✓ Montaža WC ispruća✓ Montaža bidea✓ Montaža bojlera od 30 - 80 lit. vertikalnog✓ Montaža bojlera od 30 - 80 lit. horizontalnog✓ Montaža kompletne sanitarije u kupatilu	<ul style="list-style-type: none">✓ Montaža protočnog bojlera od 5 - 10 lit. običnog✓ Montaža zidne baterije za protočni bojler✓ Montaža zidne baterije za sudoperu✓ Montaža sudopere✓ Montaža tuš baterije za kadu✓ Montaža tuš kabine za kadu✓ Montaža tuš kabine za tuš kadu✓ Montaža česme za hladnu vodu✓ Montaža priključka 3/4 ili 1/2 za veš mašinu na bilo koju bateriju✓ Montaža priključka 3/4 ili 1/2 za mašinu za pranje sudova na bilo koju bateriju✓ Montaža ormarića za kupatilo✓ Montaža sitne galanterije za kupatilo po elementu
--	--	--

Slika 1 Podkategorija vodoinstalaterske usluge

Postaj

Mobilna aplikacija „Postaj“, pruža korisnicima mogućnost pretrage majstora po različitim kategorijama i filterima. Na sajtu „Postaj“ postoji opcija da klijenti postavе oglas u kojem opisuju šta im je potrebno da se uradi, a zatim se majstori javljaju na te oglase. Iako ovo rešenje omogućava korisnicima da dobiju ponude od različitih majstora, smatram da je obrnuti pristup, gde klijenti biraju majstore na osnovu njihovih prethodnih radova, iskustva, i recenzija, logičniji i efikasniji.

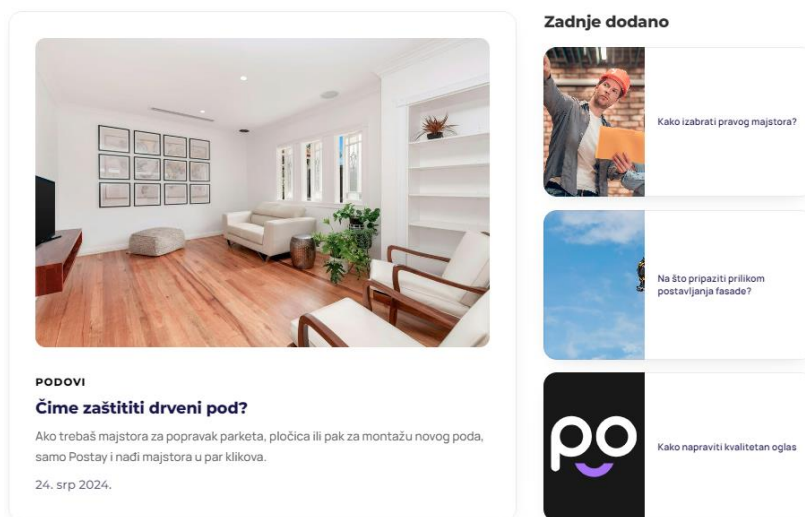
Ovaj pristup omogućava klijentima da donesu informisanu odluku na osnovu stvarnog rada i reputacije majstora, umesto da se oslanjaju na ponude koje mogu varirati u kvalitetu i ceni. Na taj način, klijenti imaju veću kontrolu nad izborom, što može povećati njihovo zadovoljstvo uslugom. Takođe, majstori koji su uložili trud u izgradnju portfolija i održavanje visokih ocena mogu biti bolje nagrađeni za svoj rad, jer će privući više klijenata bez potrebe za natjecanjem u formi nadmetanja za posao.

Ovaj model takođe može motivisati majstore da kontinuirano unapređuju svoje veštine i svoje usluge, jer će im to direktno povećati šanse za dobijanje posla. S obzirom na to, implementacija ovakvog sistema u tvojoj aplikaciji može doprineti stvaranju kvalitetnijeg i transparentnijeg tržišta usluga, što je korisno za sve strane.

Na sajtu „Postaj“ postoji izuzetno korisna opcija u vidu bloga gde majstori mogu deliti svoja iskustva, savete, i preporuke (Slika 2). Ova funkcionalnost ne samo da omogućava majstorima da predstavе svoje znanje i stručnost, već pruža i veliku vrednost korisnicima koji su u procesu renoviranja ili popravki u svom domu. Ljudi koji traže majstore mogu pronaći mnoštvo korisnih saveta i informacija koje im mogu pomoći u donošenju odluka ili u samostalnom obavljanju određenih poslova. Ovakva opcija čini sajt informativnim i korisnim za sve posetioce, što predstavlja značajnu prednost u odnosu na druga rešenja na tržištu.

Postay blog

Saznaj sve zanimljivosti vezane za aplikaciju i usluge koje tražiš



Slika 2 Blogovi

Pozovi majstora

Sajt „Pozovi majstora“ pokazuje poboljšanja u odnosu na prethodno analizirane sajtove „Profimajstor“ i „Postaj“. Prvo, za razliku od „Profimajstora“, „Pozovi Majstora“ omogućava klijentima da biraju majstore na osnovu njihovih prethodnih radova i ocena, čime se povećava transparentnost i sigurnost. Klijenti imaju pristup informacijama o majstorima, uključujući slike njihovih prethodnih radova (Slika 3), ocene od strane prethodnih klijenata, i detalje o njihovom iskustvu, što im omogućava donošenje informisanih odluka.

Takođe, „Pozovi majstora“ rešava i problem koji je prisutan na sajtu „Postaj“ gde klijenti postavljaju oglase, a majstori se javljaju na njih. Umesto toga, na sajtu „Pozovi Majstora“, klijenti imaju slobodu da sami izaberu majstora na osnovu raspoloživih informacija, što im daje veću kontrolu u procesu izbora.

Međutim, u poređenju sa „Postaj“ sajtom, „Pozovi Majstora“ ima jedan nedostatak - nedostatak blogova. Blogovi na „Postaj“ sajtu omogućavaju majstorima da dele svoja iskustva i savete, što je veoma korisno za ljude koji su u procesu radova na svom domu. Ovaj nedostatak čini „Pozovi Majstora“ manje informativnim i korisnim u tom aspektu.

Miljko Antić



Miljko Antić

Vršimo usluge gletovanja, krećenja, postavljanje izolacije, špaletni, kao i postavljanje laminata. Takođe, u ponudi radimo i farbanje stolarije. Brzo, Kvalitetno i Povoljno.

 Osnovno  Slike  Ocene

Slika 3 Profil majstora i galerija njegovih radova

Nadji Majstora

Sajt „Nadji Majstora“ pruža klijentima mogućnost da biraju majstore na osnovu ocena (Slika 4) i dostupnih informacija, što značajno povećava nivo transparentnosti i sigurnosti. Ovaj pristup omogućava korisnicima da imaju veću kontrolu nad procesom izbora majstora, jer mogu odabrati onog koji najbolje odgovara njihovim potrebama i očekivanjima.

Međutim, postoji značajan nedostatak—klijenti ne mogu videti slike prethodnih radova majstora. Ovaj nedostatak otežava procenu kvaliteta rada majstora, što može smanjiti poverenje i sigurnost prilikom donošenja odluka. Slike prethodnih radova su ključne za klijente da bi mogli vizuelno proceniti da li majstor ispunjava njihove standarde.

Sajt „Nadji Majstora“ nudi nekoliko pozitivnih aspekata. Veliki broj kategorija omogućava precizno pronalaženje majstora za specifične zadatke, što je korisno za klijente koji tačno znaju šta traže. Detaljna kategorizacija pomaže u lakoj pretrazi i pronalaženju odgovarajućih majstora. Međutim, ovaj pristup može imati negativne posledice. Ako je zajednica majstora na sajtu mala, neke od podkategorija mogu ostati prazne, što otežava pretragu i navigaciju za klijente. Previše granulirane kategorije mogu učiniti proces pretrage zamornim.

Pored toga, nedostatak blogova na sajtu „Nadji Majstora“ znači da klijenti ne mogu dobiti korisne informacije i savete koji bi mogli pomoći u donošenju odluka. Odsustvo slika prethodnih radova dodatno otežava procenu kvaliteta usluga pre nego što se donese odluka o angažovanju majstora.

Poslednji komentari

8 delatnosti • 86 zanimanja • 2648 majstora

Ime	Posao	Ocjena	Komentar
Marija	Parketar	10.0	sve kako je dogovoreno i u roku, cena povoljna sve preporuke
Miki Maricic	Građevinski limar	10.0	Sve pohvale, vrhunski majstori, prefektan rad, momci su od reci...
Miloš	Vodoinstalater	10.0	Došao na adresu, našao kvar, popravio. Cene sasvim korektne. Ljubazan, profesionalan, tačan.

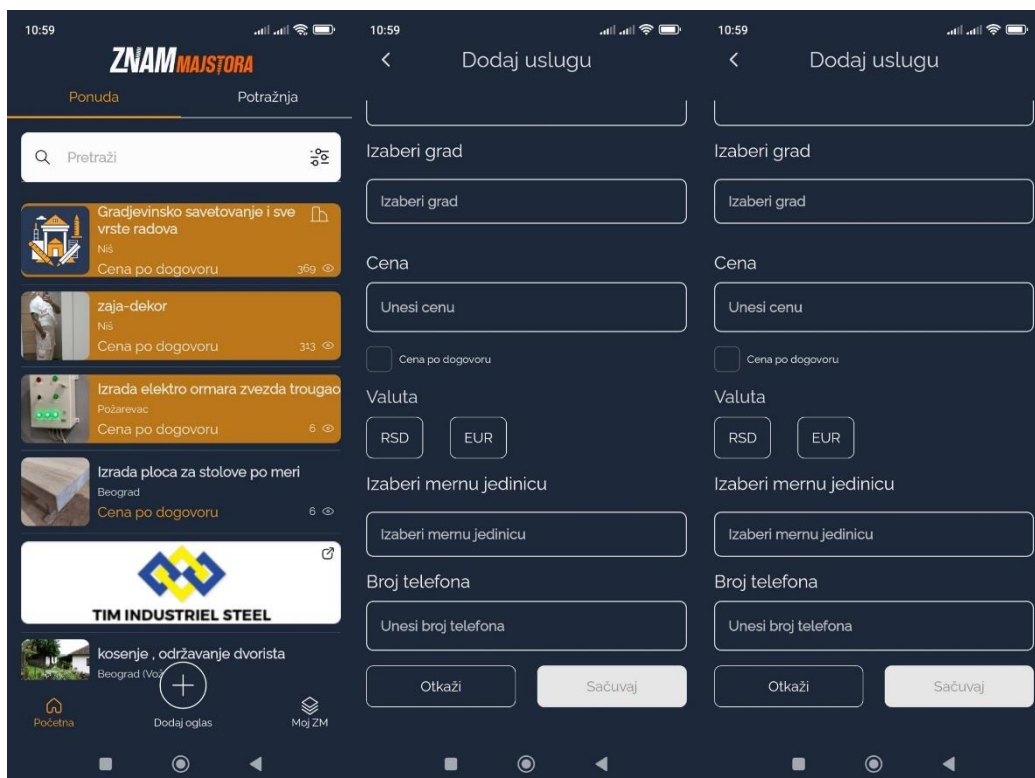
Slika 4 Ocene i komentari majstora

Znam majstora

Aplikacija „Znam Majstora“ omogućava korisnicima da postavljaju oglase kako za nudjenje, tako i za traženje usluga, čime se postiže visoka fleksibilnost u korišćenju platforme. Aplikacija je veoma pregledna, olakšavajući korisnicima navigaciju i korišćenje funkcionalnosti, iako trenutno nema svoju web verziju, što može biti nedostatak za one koji preferiraju rad sa računara.

Jedna od ključnih prednosti aplikacije je mogućnost dodavanja slika uz oglase, što značajno pomaže klijentima da steknu jasniju predstavu o uslugama koje majstori nude. Ova vizuelna komponenta je veoma važna jer omogućava klijentima da vide šta mogu očekivati od majstora pre nego što ih angažuju.

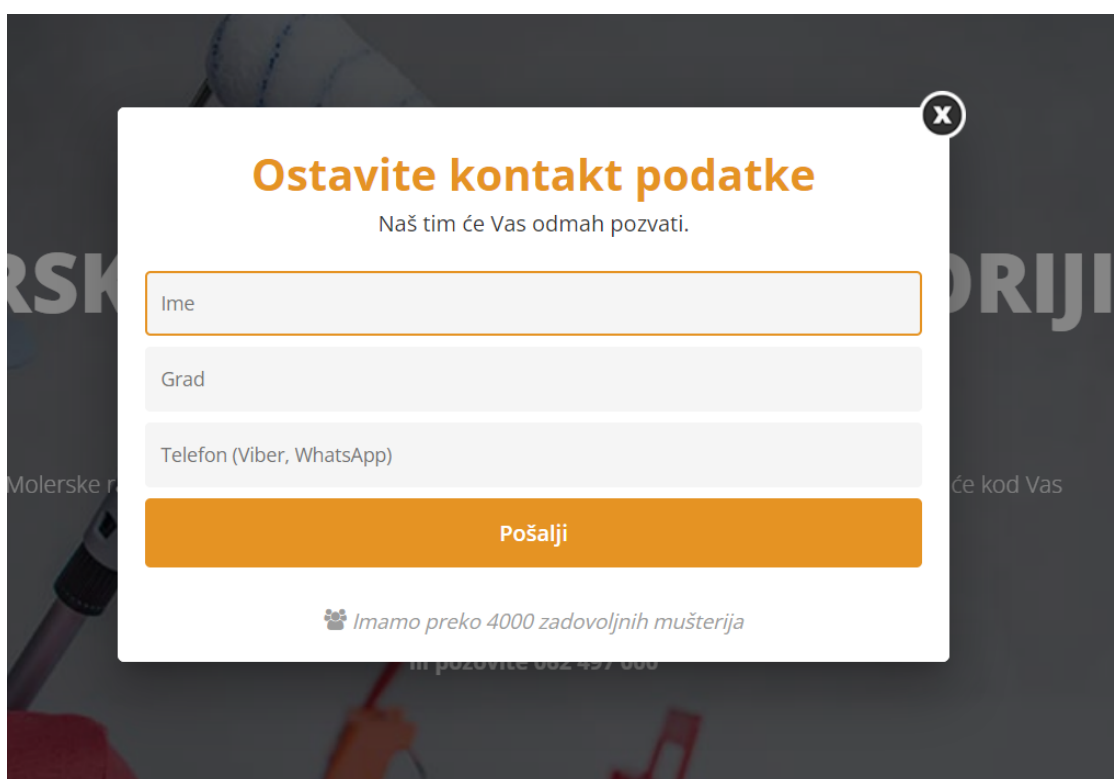
Dodatno, aplikacija koristi model zarade gde se plaćeni oglasi pozicioniraju na vrh liste, čime se povećava njihova vidljivost i šansa za brzo pronalaženje majstora ili usluge. Proces dodavanja oglasa je izuzetno detaljno osmišljen, olakšavajući korisnicima postavljanje tačnih i preciznih informacija. Ovaj proces će biti ilustrovan sa dve slike koje ću priložiti uz tekst (Slika 5).



Slika 5 Znam majstora

E- majstori

Sajt „e-majstori“ ima nekoliko značajnih mana koje treba istaći. Prvo, ne poseduje mobilnu aplikaciju, što može ograničiti pristup i funkcionalnost za korisnike koji preferiraju korišćenje telefona. Drugi nedostatak je nedostatak direktnog kontakta sa majstorima—komunikacija se odvija isključivo preko operatera na sajtu (Slika 6). Ovaj pristup može biti koristan ukoliko je baza majstora mala, jer tada administratori sajta preuzimaju odgovornost da pronađu odgovarajućeg majstora za određeni posao na odabranoj lokaciji. Međutim, ovakva organizacija ima i svoje slabosti. Kada ne postoje profili majstora, korisnici nemaju uvid u slike njihovih prethodnih radova, što može izazvati neizvesnost u pogledu kvaliteta usluge koju će dobiti. Nedostatak vizuelnih dokaza o prethodnim radovima otežava procenu i donošenje odluke o angažovanju. S druge strane, dobra stvar kod sajta „e-majstori“ je postojanje blogova, gde korisnici mogu pronaći korisne savete i informacije, što može biti od velike pomoći pri planiranju i izvršenju radova.



Ostavite kontakt podatke

Naš tim će Vas odmah pozvati.

Ime

Grad

Telefon (Viber, WhatsApp)

Pošalji

Imamo preko 4000 zadovoljnih mušterija

Slika 6 Forma za stupanje u kontak sa majstorima, E- majstori

2.2 Predlog novog rešenja

Na osnovu analize trenutnih rešenja, identifikovane su sledeće mogućnosti za unapređenje aplikacije za pronalazak majstora. Prvo, detaljni profili majstora mogu biti dodatno unapređeni kroz uvođenje biografija, specijalizacija, detaljnih opisa usluga i galerija radova, što bi omogućilo korisnicima da steknu bolji uvid u stručnost i kvalitet rada majstora.

Drugo, vizualizacija usluga može biti poboljšana kroz dodavanje slika za svaku uslugu i projekat, čime bi korisnici dobili jasniji prikaz onoga što majstori nude. Omogućavanje majstorima da dele savete ili pišu blogove, dodatno bi obogatile korisničko iskustvo.

Treće, sistem recenzija i ocena može biti unapređen kroz uvođenje detaljnijih komentara i ocenjivanja usluga, što bi povećalo transparentnost i olakšalo korisnicima donošenje odluka.

Predložene promene značajno će poboljšati korisničko iskustvo, omogućavajući korisnicima da lakše pronađu majstore koji odgovaraju njihovim potrebama. Majstori će imati priliku da se bolje predstavljaju potencijalnim klijentima kroz detaljnije profile i bogatiju vizualizaciju svojih usluga. Na ovaj način, aplikacija će postati konkurentnija na tržištu i pružiti veću vrednost kako korisnicima, tako i majstorima (Tabela 1).

Tabela 1 Funkcionalnosti postojećih rešenja

Sajt \ Funkcija	Slike usluga	Blog	Komentari o majstorima	Jednostavnost u kontaktu sa majstorima
„ProfiMajstor“	✗	✗	✗	✗
„Postaj“	✓	✓	✓	✗
„Pozovi majstora“	✓	✗	✓	✓
„Nadji majstora“	✗	✗	✓	✓
„Znam majstora“	✓	✗	✓	✓
„E-majstori“	✗	✓	✗	✗

3 Tehnologije korišćene za izradu veb aplikacije

U okviru ove glave biće predstavljene tehnologije koje su korišćene za izradu veb aplikacije. Prvo će biti opisan *JavaScript* kao zajednički programski jezik klijentske i serverske strane, nakon togai njihovi radni okviri *Angular* i *SpringBoot* respektivno, i na kraju baza podataka – *MySQL*.

3.1 JavaScript

JavaScript je postao nezaobilazan alat u svetu modernog web razvoja. Od svojih skromnih početaka kao jednostavan skriptni jezik namenjen za dodavanje osnovne interaktivnosti na veb stranice, *JavaScript* je evoluirao u jedan od najmoćnijih i najpopularnijih programskih jezika današnjice. Njegova početna svrha bila je omogućiti početnicima lako dodavanje animacija i interakcija na svoje stranice, ali s vremenom, *JavaScript* je postao ključan za izradu složenih i robusnih veb aplikacija.

Kako su aplikacije postajale sve kompleksnije, tako je postala očigledna potreba za dobro strukturiranim kodom koji je lako održavati. U tom smislu, *JavaScript* je prošao kroz brojne faze razvoja, uključujući uvođenje modernih programskih paradigmi kao što su objektno orijentisano programiranje, asinhrono programiranje, i modularnost. Ove promene su omogućile programerima da pišu kod koji je ne samo efikasan već i lak za održavanje, što je ključno kada se radi na velikim projektima.

Jedan od ključnih elemenata koji su doprineli rastu popularnosti *JavaScript* je bogat ekosistem biblioteka i radnih okruženja koji su razvijeni oko njega. Biblioteke poput *React*, *Angular* i *Vue.js* omogućavaju programerima da brzo i efikasno grade složene korisničke interfejse, dok *Node.js* i *Express* omogućavaju izradu serverskih aplikacija koristeći isti jezik. Ovaj ekosistem pruža programerima alate za rešavanje širokog spektra problema, od jednostavnih veb stranica do složenih aplikacija i sistema.

JavaScript je objektno orijentisan skriptni jezik, što znači da omogućava rad sa objektima - strukturama podataka koje kombinuju podatke i funkcije koje na tim podacima operišu. Ovaj pristup je od suštinske važnosti za stvaranje složenih aplikacija, jer omogućava programerima da organizuju svoj kod na način koji je intuitivan i lak za razumevanje. Kroz upotrebu objekata, programeri mogu precizno definisati kako se određeni elementi na veb stranici ponašaju i interaguju jedni sa drugima.

Jedna od ključnih karakteristika *JavaScript* je njegova integracija sa *HTML* i *CSS*, čime se kreira dinamički *HTML* (*DHTML*). Ova kombinacija omogućava kreiranje bogatih korisničkih interfejsa koji se mogu menjati u realnom vremenu, u zavisnosti od korisničkih akcija. Na primer, putem *JavaScript* možete jednostavno dodavati ili uklanjati elemente sa stranice, menjati njihov izgled, ili čak kreirati složene animacije i interakcije koje čine iskustvo korisnika bogatijim i intuitivnijim.

Pored toga, *JavaScript* je doživeo značajnu ekspanziju sa pojavom *Node.js* platforme, koja omogućava izvršavanje *JavaScript* na serverskoj strani. To znači da sada možete koristiti *JavaScript* za izradu kompletnih veb aplikacija koje pokrivaju sve aspekte razvoja - od korisničkog interfejsa do serverskog dela. Ova fleksibilnost je dovela do eksplozije popularnosti *JavaScript*, čineći ga neophodnim alatom za svakog savremenog programera.

U svetu u kojem sve više biznisa prelazi na online poslovanje, *JavaScript* nudi izuzetne poslovne mogućnosti. Njegova univerzalnost i široka primena omogućavaju programerima da rade na raznovrsnim projektima, uključujući razvoj mobilnih aplikacija, video igara, pa čak i softvera za pametne uređaje poput satova i televizora. *JavaScript* je podržan od strane svih popularnih veb pretraživača kao što su *Mozilla Firefox*, *Google Chrome*, *Safari* i *Opera*, kao i na različitim operativnim sistemima, uključujući *Windows*, *macOS*, *Linux*, *Android* i *iOS*.

Nije iznenađujuće da je *JavaScript* trenutno najpopularniji programski jezik na svetu, a njegova popularnost nastavlja da raste. Sa sve većim brojem biznisa koji se odlučuju za digitalno prisustvo, interesovanje za *JavaScript*, njegova upotreba i relevantnost će samo nastaviti da raste u narednim godinama. Zahvaljujući svojoj prilagodljivosti, lakoći učenja i širokom spektru primena, *JavaScript* ostaje ključni alat za svakog ko želi da izgradi uspešnu karijeru u svetu programiranja (Slika 7) [1] [2].



Slika 7 JavaScript

3.2 Angular JS

Angular je *JavaScript* radni okvir za razvoj klijentskog dela koji se koristi za razvoj dinamičkih veb aplikacija za desktop i mobilne uređaje. Prva verzija ovog radnog okvira objavljena je 2012. godine od strane kompanije *Google*.

2016. godine pojavljuje se druga verzija *Angular* okvira. *Angular* je iz temelja redizajniran na osnovu saveta iz prakse, ali i tako da bude u koraku s modernim tehnologijama koje su se pojavile nakon prve verzije.

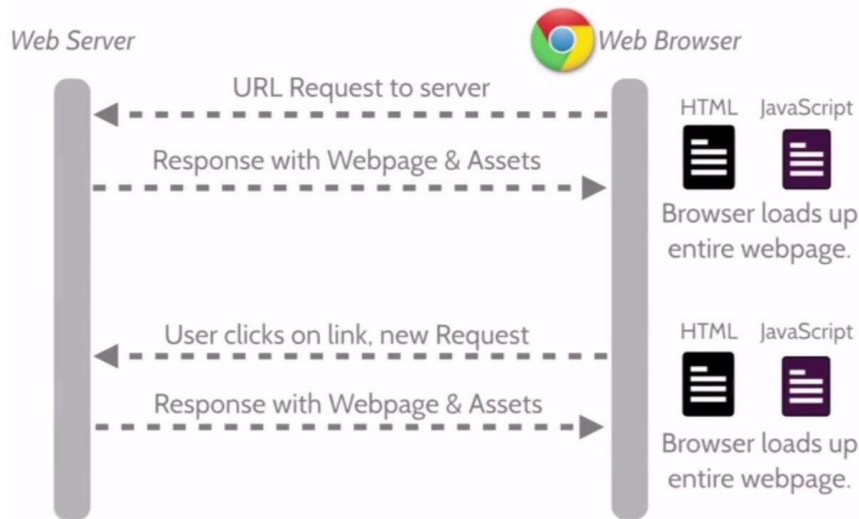
Angular okvir napisan je programskom jeziku *TypeScript*, razvijenom od strane kompanije *Microsoft*. Njegova svrha je nadomestiti nedostatke *JavaScript* jezika, zato se često u literaturi *TypeScript* naziva nadskup *JavaScript* jezika. *TypeScript* nije jezik kojeg pretraživači mogu interpretirati, stoga se *TypeScript* kod prevodi u *JavaScript* kod. *TypeScript* omogućava strogo tipiziranje nad *JavaScript* kodom, to jest možemo deklarirati tipove podataka. Time je omogućena provera nad tipovima podataka prilikom prevođenja koda kako ne bi došlo do greške prilikom izvršavanja koda. Uz strogo tipiziranje i otkrivanje grešaka prilikom prevođenja, najvrednija dopuna *JavaScript* kodu jeste uvođenje mehanizama objektno-orijentiranog programiranja [3].

Dobra svojstva prve verzije ovog okvira, poput direktiva i zavisnosti, nasleđena su i poboljšana unutar druge verzije. Mnogi mehanizmi prve verzije su odbačeni radi lakšeg snalaženja prilikom rada s *Angular* okvirom, ali i radi poboljšanja performansi same aplikacije. Oslanjajući se na *TypeScript* sintaksu, svaki se entitet unutar *Angular* okvira definiše pomoću *TypeScript* klase čime je omogućeno jednostavnije pisanje i lakše snalaženje unutar koda. Uz to, *Angular* okvir dolazi s komandnim alatom *Angular-CLI* koji vodi računa o stvaranju kostura same aplikacije (Slika 8).



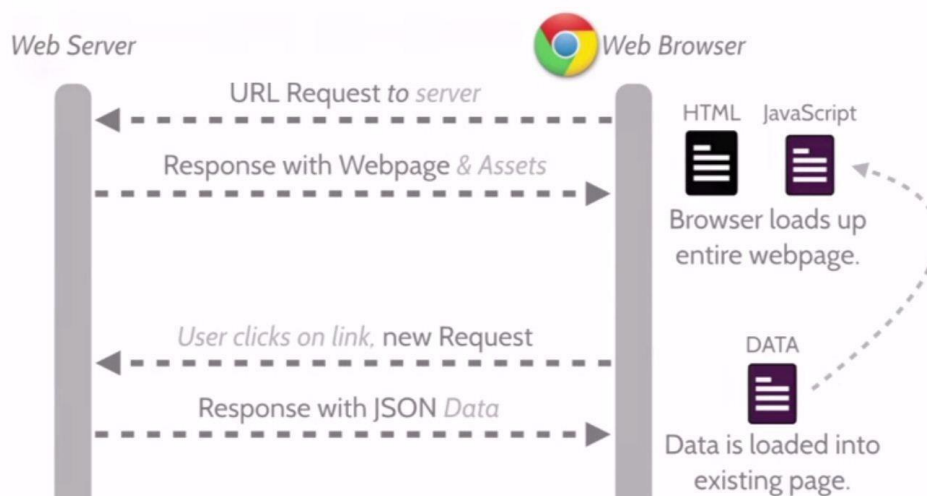
Slika 8 Angular

Osnova namena *Angular* okvira je razvoj jednostraničnih veb aplikacija. Njihova glavna karakteristika jeste prikazivanje sadržaja na jednoj glavnoj stranici što znači da prilikom navigacije korisnika kroz aplikaciju nema potrebe za slanje zahteva serveru za dohvaćanje nove *HTML* stranice. *SPA (Single Page Applications)* aplikacije se oslanjaju na router. Ruter se sastoji od rutak koje opisuju mesto na kojem bi se trebalo preusmeriti. To mogu biti statične ili dinamičke rute.



Slika 9 Klijent server komunikacija

Kod klasične klijent server komunikacije sa jedne strane imamo server dok je sa druge klijent. Klijent predstavlja veb pretraživač u našem slučaju (Slika 9) *Google Chrome*. Kada korisnik u veb pretraživač unese *URL*, na server se šalje zahtev. Server tada odgovara tako što pošalje veb stranicu, nakon toga korisnik dobije učitanu stranicu. Zatim, kada korisnik klikne na neki link, veb pretraživač na server šalje novi zahtev. Server tada ponovo šalje celu veb stranicu sa svim dodacima, te se sve ponovo učitava u veb pretraživač.



Slika 10 Angular komunikacija sa serverom

Pri korišćenju *Angular* proces je malo drugačiji. Ipak, početak je isti, korisnik unese *URL* u vebpretraživač pa se zahtev šalje na server. Server tada šalje celu veb stranicu i korisnik dobije učitanu veb stranicu. Kada korisnik klikne na link i želi da pređe na neku drugu stanicu, veb pretraživač šalje novi zahtev na server. Međutim, server sada ne šalje celu stranicu kao pre, nego samo podatke u *JSON* formatu. *JavaScript* kod stranice tada obrađuje dobijene podatke, te na taj način odlučuje šta će korisnik videti na ekranu (Slika 10).

Angular je dizajniran sa ciljem da razdvoji *DOM* manipulacije od aplikacione logike. *Angular* takođe razdvaja klijentsku stranu aplikacije od serverske strane aplikacije. Ovo omogućava rad na razvoju obe strane paralelno, i omogućava ponovnu upotrebu obe strane. On takođe obezbeđuje i strukturu za redosled razvoja aplikacije – od projektovanja korisničkog interfejsa, preko pisanja poslovne logike do testiranja.

Angular implementira *MVC* arhitekturu radi razdvajanja komponenti za prezentaciju, podatke i logičkih komponenti. Koristeći zavisnost paketa, *Angular* donosi tradicionalne serverske servise, kao što su kontroleri koji zavise od prezentacionog sloja, na veb aplikacije na klijentskoj strani. Shodno tome, veliki deo tereta na serveru se može smanjiti.

Angular biblioteka radi tako što prvo pročita *HTML* strane, koja ima ugrađene dodatne nestandardne tag attribute. *Angular* te attribute interpretira kao direktive da bi vezao ulazne ili izlazne delove stranice za model koji je predstavljen standardnim *JavaScript* promenljivama. Vrednosti tih *JavaScript* promenljivih se mogu ručno podesiti u kodu, ili mogu biti preuzete od statičnih i dinamičnih *JSON* izvora. U izradi projekta je korišćen i *Angular* materijal koji predstavlja materijalni dizajn za *Angular* komponente. Omogućuje brz razvoj i dizajn korisničkog interfejsa uz moderne komponente [4].

Mnogi frejmvorci su napravljeni od strane ljudi iz *Open source* zajednice, *Angular* je napravljen i održavan od strane inženjera kompanije *Google*. Samim tim možete biti sigurni da radite sa pouzdanim i testiranim kodom na vašem projektu. *McDonald's*, *AT&T*, *HBO*, *Apple*, *Forbes*, *Adobe*, *Nike*, a takođe i *Microsoft*, *PayPal* samo su neke od kompanija koje koriste *Angular* i koje mu daju na kredibilitetu.

Izbor grafičkog korisničkog interfejsa za *Angular* je veliki. Razvoj aplikacije u nekoj od popularnih razvojnih okolina (*WebStorm*, *Intellij IDEA*, *Visual Studio* ili *Angular IDE*) donosi prednosti kao stosu predlozi programskog koda, trenutni prikaz greški i ostale povratne informacije.

Za *Angular* programere postoji ogroman broj paketa otvorenog koda. Glavni paketi uključuju *NgBootstrap*, *Angular Google Maps*, *NgRx*, *NgTranslate*, *AngularFire*, *NgxTextEditor*, *Angular Material*, *Ng2 Pdf Viewer* i grafikone. (Npm packages n.d.) Nekim od ovih paketa upravlja službeni *Angular* tim, dok zajednica programera upravlja drugim. Ako želite stvoriti složenu funkcionalnost ili komponentu za svoju aplikaciju, postoji velika verovatnoća da paket već postoji, pa možete iskoristiti postojeće rešenje [5].

3.3 Java

Java je jedan od najpopularnijih programskih jezika i platformi na svetu, poznat po svojoj prenosivosti, sigurnosti i skalabilnosti. Razvijen od strane *Sun Microsystems* 1995. godine, *Java* je dizajnirana da bude objektno orijentisana, što omogućava programerima da pišu kod koji je modularan, lak za održavanje i ponovnu upotrebu. Ova karakteristika, zajedno sa filozofijom "piši jednom, izvršavaj svuda", čini *Javu* jednim od najpoželjnijih jezika u razvoju softverskih aplikacija (Slika 11) [6].

Jedan od ključnih faktora za uspeh *Jave* je *Java Virtual Machine (JVM)*, koja omogućava da se *Java* aplikacije izvršavaju na bilo kojoj platformi koja ima instaliranu *JVM*. Ova nezavisnost od platforme daje *Javi* prednost u odnosu na druge programske jezike, jer omogućava programerima da razvijaju aplikacije koje mogu raditi na različitim uređajima bez potrebe za promenama u kodu.

Java se koristi u različitim domenima, uključujući razvoj veb aplikacija, mobilnih aplikacija, enterprise sistema, igara i ugrađenih sistema. Na primer, *Java* je osnovni jezik za razvoj *Android* aplikacija, što joj daje značajnu ulogu u mobilnom razvoju. Pored toga, *Java* je omiljena u velikim poslovnim sistemima zbog svoje sposobnosti da rukuje velikim količinama podataka i korisnika.

Njene ključne karakteristike su:

- **Objektno-orijentisan pristup:** *Java* je zasnovana na principima objektno-orijentisanog programiranja (OOP), što omogućava modularnost, enkapsulaciju, nasleđivanje i polimorfizam. OOP omogućava programerima da organizuju kod na način koji je intuitivan i olakšava održavanje i proširenje aplikacija.
- **Prenosivost:** Zahvaljujući *JVM*, *Java* aplikacije su prenosive i mogu se izvršavati na bilo kojoj platformi koja podržava *JVM*, bez obzira na operativni sistem ili hardversku arhitekturu.
- **Sigurnost:** *Java* je dizajnirana sa sigurnošću u vidu, sa ugrađenim mehanizmima za zaštitu od različitih pretnji. *JVM* pruža izolaciju između aplikacija, dok *Java API* omogućavaju programerima da implementiraju sigurnosne protokole i metode za zaštitu podataka.
- **Multithreading:** *Java* omogućava razvoj aplikacija koje mogu izvršavati više niti istovremeno, što je korisno za aplikacije koje zahtevaju visoke performanse i efikasno korišćenje resursa.
- **Robustnost:** *Java* je poznata po svojoj stabilnosti i pouzdanosti, sa ugrađenim mehanizmima za detekciju i rukovanje greškama, kao i za upravljanje memorijom, što smanjuje rizik od padova i problema sa performansama.

Java ima bogat ekosistem alata i biblioteka koje omogućavaju programerima da razvijaju aplikacije brže i efikasnije. *Java* je postala ključni jezik za razvoj mikroservis arhitekture, zahvaljujući radnim okvirima kao što su *Spring Boot* i *Micronaut*. Ovi alati omogućavaju jednostavnu izradu i implementaciju mikroservisa, što je postalo ključno u modernom razvoju aplikacija koje zahtevaju skalabilnost i fleksibilnost., dok alatke kao što su *Maven* i *Gradle* pomažu u automatizaciji izgradnje i upravljanju zavisnostima. *IntelliJ IDEA*, *Eclipse* i *NetBeans* su neki od najpopularnijih integrisanih razvojnih okruženja za *Javu*, koja pružaju bogate funkcionalnosti za ubrzanje razvoja.

Java se koristi i u oblastima kao što su obrada velikih podataka i mašinsko učenje. Alati kao što su *Apache Hadoop* i *Apache Spark*, koji su napisani u *Javi*, omogućavaju efikasno rukovanje velikim količinama podataka. Takođe, *Java* biblioteke poput *Weka* i *Deeplearning4j* pružaju alate za mašinsko učenje i duboko učenje.

Java je često korišćena za razvoj aplikacija koje se izvode u oblaku. Platforme kao što su *Amazon Web Services (AWS)*, *Google Cloud Platform (GCP)*, i *Microsoft Azure* pružaju podršku za *Java* aplikacije, omogućavajući im da se lako integrišu u cloud ekosisteme.

Takođe *Java* ima jednu od najvećih i najaktivnijih programerskih zajednica na svetu. Postoji obilje resursa, tutorijala, foruma i konferencija posvećenih *Javi*, što pomaže programerima da brzo rešavaju probleme i uče nove tehnike. Sve ovo su razlozi zašto sam se odlučio za *Javu* u svom projektu [7] [8].



Slika 11 Java

3.4 SpringBoot

Spring Boot je postao sinonim za brz i efikasan razvoj *Java* aplikacija, omogućavajući programerima da brzo kreiraju robusne, samostalne aplikacije koje su spremne za produkciju. Kao deo šireg *Spring* ekosistema, *Spring Boot* je dizajniran da ukloni složenost koja je tradicionalno povezana sa *Spring Framework*, olakšavajući postavljanje aplikacija bez potrebe za opsežnim konfiguracijama (Slika 12).

Jedna od najvažnijih prednosti *Spring Boot* je automatska konfiguracija, koja omogućava aplikaciji da se automatski prilagodi na osnovu zavisnosti prisutnih u *classpath*. Ovaj mehanizam automatske konfiguracije omogućava programerima da se fokusiraju na poslovnu logiku umesto na tehničke detalje podešavanja. Na primer, ako je u projektu uključena baza podataka, *Spring Boot* automatski konfiguriše povezanost sa bazom, bez potrebe za ručnim podešavanjem.

Spring Boot takođe uključuje ugrađene servere kao što su *Tomcat*, *Jetty* i *Undertow*, što omogućava jednostavnu implementaciju aplikacija bez potrebe za eksternim kontejnerima. Ovo značajno smanjuje vreme potrebno za postavljanje aplikacije i olakšava testiranje tokom razvoja. *Starter* zavisnosti koje dolaze uz *Spring Boot* dalje pojednostavljaju proces upravljanja zavisnostima grupisanjem često korišćenih biblioteka u gotove pakete.

Spring Boot je posebno koristan za izradu mikroservisnih arhitektura, gde se veliki sistemi razbijaju na manje, autonomne komponente koje mogu nezavisno da se razvijaju, testiraju i postavljaju. Ova fleksibilnost omogućava kompanijama da brže reaguju na promene u poslovnim zahtevima i da poboljšaju skalabilnost svojih sistema. *Netflix*, na primer, koristi *Spring Boot* kao osnovu svoje mikroservisne arhitekture, omogućavajući efikasnu izgradnju i održavanje kompleksne infrastrukture u oblaku.

Pored mikroservisa, *Spring Boot* se koristi za razvoj *RESTful API*, veb aplikacija, i *enterprise* sistema, integrisanih sa *Spring Security*, *Spring Data*, i *Spring Cloud*. Ova fleksibilnost omogućava programerima da izgrade širok spektar aplikacija koje se lako mogu proširiti i prilagoditi različitim potrebama. *Amazon* koristi *Spring Boot* za izgradnju svoje platforme za elektronsku trgovinu, dok *Google* koristi ovaj okvir za razvoj i implementaciju svoje infrastrukture u oblaku.



Slika 12 SpringBoot

U poređenju sa tradicionalnim *Spring* radnim okvirom, *Spring Boot* značajno smanjuje složenost postavljanja i konfigurisanja aplikacija. Dok su u prošlosti programeri morali ručno da konfigurišu svaki aspekt aplikacije, *Spring Boot* automatski prepoznaje i konfiguriše potrebne komponente, čime omogućava brže pokretanje i razvoj. Takođe, u poređenju sa drugim mikroslužbenim okvirima kao što su *Dropwizard* i *Play*, *Spring Boot* pruža dublju integraciju sa *Spring* ekosistemom, čineći ga preferiranim izborom za mnoge organizacije.

Spring Boot je takođe dizajniran da bude kompatibilan sa različitim alatima za kontinualnu integraciju i isporuku, što omogućava jednostavno postavljanje i ažuriranje aplikacija u produkciji. Njegova sposobnost da se integriše sa alatima kao što su *Docker* i *Kubernetes* dodatno pojednostavljuje proces upravljanja aplikacijama u složenim okruženjima.

Spring Boot je neophodan alat za svakog *Java* programera koji želi da ubrza razvoj aplikacija bez žrtvovanja kvaliteta i performansi. Njegova jednostavnost, kombinovana sa moćnim funkcijama, omogućava razvoj aplikacija svih veličina, od malih veb projekata do kompleksnih *enterprise* sistema. Bez obzira na to da li radite na jednostavnom *RESTful API* ili na složenoj mikroservisnoj arhitekturi, *Spring Boot* pruža alate i resurse koji su vam potrebni da obavite posao brzo i efikasno [9] [10].

3.5 MySQL

MySQL je sistem za upravljanje relacionim bazama podataka (*RDBMS*) koji se široko koristi za veb aplikacije i drugi softver. Relacioni model podrazumeva da su podaci organizovani u formi tabela sa kolonama i redovima, a grupisani su relacijama. Prvi put je razvijen 1994. godine od strane švedske kompanije *MySQL AB* i kasnije je kupljena od strane *Sun Microsystems* 2008. godine, koja je zatim kupljena od strane *Oracle Corporation* 2010. godine (Slika 13).

Prednosti *MySQL*:

- *MySQL* je softver otvorenog koda, što znači da svako može da ga koristi i modifikuje. To ga je učinilo popularnim izborom za mnoge programere i organizacije.
- *MySQL* je poznat po svojoj jednostavnosti instaliranja i upotrebi. Ima jednostavan proces podešavanja i korisnički prijateljski interfejs, što ga čini dobrom opcijom za početnike.
- *MySQL* je poznat po svojim dobrim performansama, optimizovan je da brzo radi što ga čini idealnim za aplikacije koje zahtevaju brzi pristup podacima.
- Skalabilnost, vertikalna – dodavanje više resursa na postojeći servis, horizontalna – dodavanje više servisa u klaster.
- Pouzdanost, podržava *ACID* (*Atomicity, Consistency, Isolation, Durability*) svojstva, što osigurava da su transakcije uvek dosledne i da ne gube podatke čak i u slučaju grešaka u sistemu. *MySQL* takođe pruža opcije za automatski oporavak podataka i replikaciju, što dodatno povećava pouzdanost i dostupnost sistema.
- *MySQL* je dostupan za različite operativne sisteme, uključujući *Windows, Linux* i *macOS*. To ga čini fleksibilnim izborom za programere i organizacije [11] [12].



Slika 13 MySQL

3.6 Relacione baze podataka

Relacione baze podataka (*RDBMS*) su vrsta baza podataka koje organizuju podatke u tabelama koje su međusobno povezane putem odnosa. Svaka tabela sadrži redove i kolone (atribute), pri čemu su redovi jedinstveni i identifikovani ključevima, kao što su primarni ključevi, koji osiguravaju integritet podataka. Ovi odnosi omogućavaju efikasno pretraživanje, upit i manipulaciju podacima.

Jedna od ključnih prednosti relacionih baza podataka je njihova sposobnost da održavaju integritet podataka kroz mehanizme kao što su primarni i strani ključevi, koji definišu odnose između tabela. Normalizacija, proces organizovanja podataka kako bi se smanjila redundancija, takođe igra ključnu ulogu u očuvanju doslednosti podataka.

Relacione baze podataka koriste *Structured Query Language (SQL)* za interakciju sa podacima. *SQL* omogućava korisnicima da definišu, manipulišu i pristupaju podacima kroz različite operacije, uključujući selektovanje, umetanje, ažuriranje i brisanje podataka.

U primeru na Slika 14 vidimo jednu entitetsku klasu koja ima id kao primarni ključ, nekoliko kolona, koje moraju imati vrednost i kolonu user_id koja je strani ključ i koja se mapira na tabelu UserEntity i njenu kolonu id.

```
@Table(name = "service")
@Entity
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
public class ServiceEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false)
    private String l1Category;
    @Column(nullable = false)
    private String l2Category;
    @Column(nullable = false)
    private String description;
    @ManyToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private UserEntity user;
}
```

Slika 14 Tabela u bazi

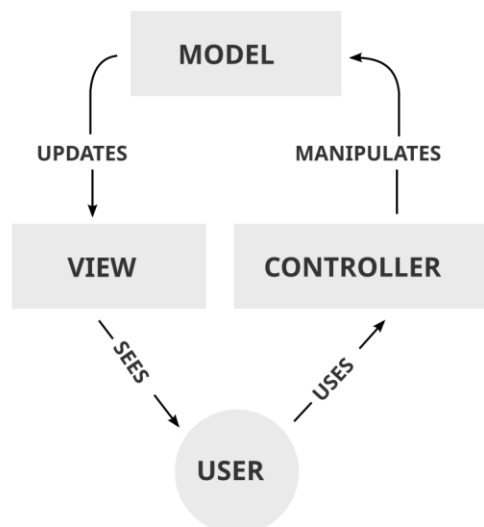
4 Pregled detalja rešenja

U prvom zaglavlju biće opisana arhitektura sistema, nakon toga imajući u vidu obim aplikacije u ovoj glavi će biti prikazani i objašnjeni samo najinteresantniji delovi koda na serverskom i klijentskom delu aplikacije respektvno.

4.1 Arhitektura sistema

Za aplikaciju će biti korišćen *Model-View-Controller* projektni uzorak (Slika 15). Projektni uzorak koji se obično koristi za razvoj korisničkih interfejsa. Počiva na ideji o ponovnoj upotrebi već postojećeg softverskog koda, olakšavanju razvoja i kasnijem održavanju aplikacionog softvera metodom razdvajanja na posebne komponente: model, prikaz podataka i kontrolor, pri čemu je komponenta za prikaz informacija odvojena od interakcije korisnika sa tim informacijama [13].

MVC arhitektura se sastoji od određenih komponenti u kojima je svaka zadužena za obavljanje specifičnih funkcija. Takvo rešenje je mnogo bolje u odnosu na ranije rešenja u kojima se sve nalazilo na jednom mestu, što je dovodilo do otežanog čitanja izvornog koda, otežanog pronalaženja i ispravljanja grešaka, kao i do znatno manje funkcionalnosti i fleksibilnosti. MVC ja kao okvir softverske arhitekture nastao pre nego što je izmišljen veb-pregledač, i u početku je korišćen samo za kreiranje grafičkog korisničkog okruženja.



Slika 15 Model-View-Controller

MVC projektni šablon se sastoji od tri zasebne, ali međusobno povezane komponente:

- *Model* – je centralni deo aplikacije, koja obuhvata promenljivu (dinamičku) strukturu podataka, direktno upravljanje podacima, logikom i pravilima aplikacije
- *View* - bilo koji izlazni prikaz podataka u korisničkom okruženju (na primer grafički pomoću dijagrama), pri čemu se isti podaci mogu prikazati na više načina (na primer stubični dijagrami za menadžment i tabelarni prikaz za računovodstvo)
- *Controller* - ulazne podatke pretvara u komande koje upravljaju modelom ili prikazom podataka u korisničkom okruženju.

Model sadrži glavne programske podatke, kao što su informacija o objektima iz baze podataka, sastoji se od skupa klasa koje modeliraju i podržavaju rešavanje problema kojim se aplikacija bavi. Obično je stabilna komponenta, koja traje koliko i sam problem. Sva poslovna logika aplikacije sadržana je u modelu. Postoji nekoliko načina konstrukcije kostura modela. Naime, programer može najpre da konstruiše model, definisanjem klasa i atributa unutar klasa, te kasnije, na osnovu klasa da formira bazu podataka.

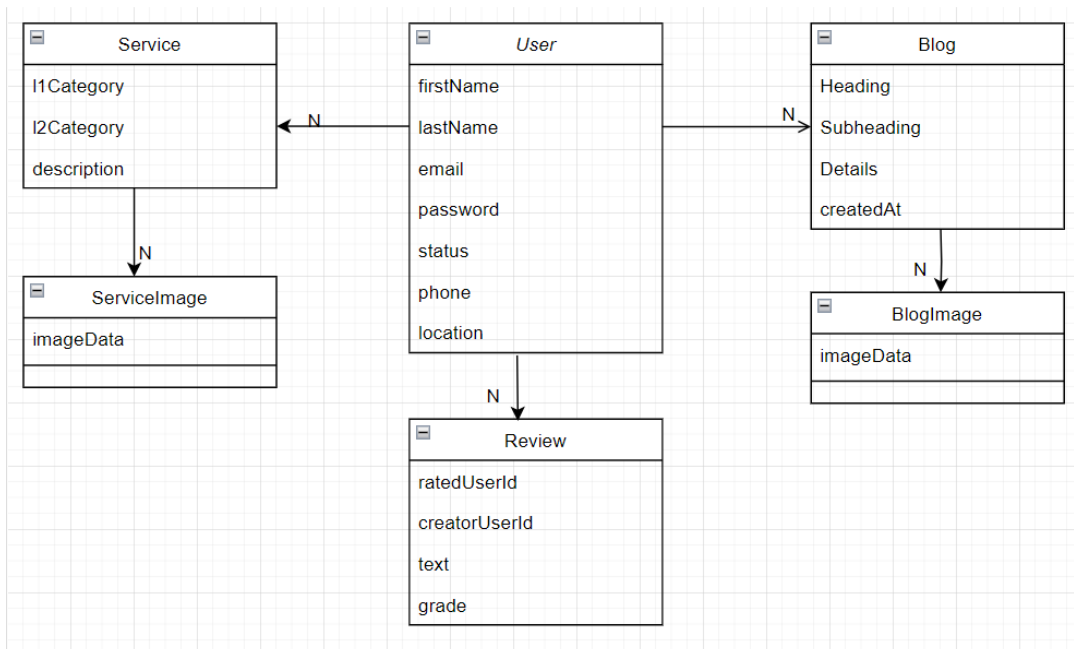
View je komponenta za prikaz podataka je poslednji sloj *MVC* arhitekture, koji sadrži okruženje aplikacije, odnosno obezbeđuje različite načine za prezentovanje podataka dobijenih od modela, preko kontrolora. Korisnik može videti samo ono što se vidi u ovoj komponenti, dok su model i kontrolor obično skriveni od korisnika. Najvažnija osobina ove komponente je njena jednostavnost, jer predstavlja vizuelni prikaz modela koji sadrži metode za prikazivanje i omogućava korisniku da menja podatke. Ona svakako ne treba da bude nadležna za skladištenje podataka, osim kada se koristi keširanje kao mehanizam za poboljšanje karakteristika.

Controller predstavlja posrednika između prikaza podataka i modela. Sadrži glavne mehanizme za kontrolu toka programa, odnosno ponašanje same aplikacije i upravlja korisničkim zahtevima. On je programski najzahtevniji deo aplikacije. Kontrolor interpretira ulazne podatke korisnika i prosleđuje ih do modela ili ih prikazuje korisniku. On sadrži deo aplikacijske logike i ima sposobnost da utiče na stanje modela, odnosno odlučuje kako model treba da se izmeni, kao rezultat korisničkih zahteva, kao i način na koji će se podaci prikazati.

MVC arhitektura omogućava visoku modularnost, što olakšava promene i proširenja u softverskoj aplikaciji. Kada je potrebno unaprediti određenu funkcionalnost ili dodati novu, promene se najčešće mogu izvršiti samo u jednoj od tri komponente, bez potrebe za značajnim promenama u ostatku koda. Ovo doprinosi lakšem održavanju aplikacije tokom njenog životnog ciklusa. Takođe, zbog jasne podele odgovornosti, timovi programera mogu paralelno raditi na različitim komponentama, čime se ubrzava proces razvoja. *MVC* je postao standard u industriji zbog svoje fleksibilnosti i sposobnosti da se lako prilagodi različitim vrstama aplikacija, uključujući i moderne web aplikacije.

4.2 Klasni dijagram sistema

Na Slika 16 Dijagram klasa je prikazan klasni dijagram sistema. Klasni dijagram je tip dijagrama statičke strukture koji opisuje strukturu sistema prikazivanjem klasa sistema, njihovih atributa i odnosa između objekata. Na dijagramu je prikazana centralna klasa *User* koja ima osim svoji atributa (*firstName*, *lastName*, *email*, *password*, *status*, *phone*, *location*) ima i odnos ka tabelama *Service* i *Blog* 1 na više, što u praksi znaci da će domenksi objekat klase *User* imati kolekciju blogova i servisa, date kolekcije će predstavljati njegove lične blogove i servise koje je sam kreirao. I klasa *Blog* i klasa *Service* imaju odnos 1 na N sa klasama *BlogImage* i *ServiceImage* u kojima se u blob formatu čuvaju vrednosti njihovih slika.



Slika 16 Dijagram klasa

4.3 JSON Web Token (JWT)

Autentifikacija je proces utvrđivanja identiteta korisnika, kojim se proverava da li korisnik ima pravo pristupa sistemu. Jedan od popularnih metoda autentifikacije je *JWT* token autentifikacija. *JWT* token autentifikacija se oslanja na *JSON Web Token (JWT)* format za generisanje tokena. Ovi tokeni sadrže informacije o korisniku ili sesiji i potpisani su kriptografski. Kada korisnik obavi uspešnu prijavu, server generiše *JWT* token i vraća ga korisniku. Ovaj token se zatim šalje sa svakim zahtevom ka serveru kako bi se autentifikovao korisnik. Prednost *JWT* tokena je u tome što su samoodrživi – sadrže sve potrebne informacije za autentifikaciju, što znači da server ne mora čuvati stanje sesije.

Implementacija *JWT* token autentifikacije je relativno jednostavna, postoje biblioteke poput *jwt.io* koje omogućavaju lako generisanje i verifikaciju tokena u različitim programskim jezicima. U većini slučajeva, potrebno je samo nekoliko linija koda da se integriše *JWT* autentifikacija u vašu aplikaciju (Slika 17). Ovo čini *JWT* token autentifikaciju popularnim izborom za razvoj aplikacija, posebno za one koje koriste *RESTful API*. Međutim, važno je napomenuti da *JWT* tokeni treba da se koriste uz odgovarajuće mere bezbednosti, uključujući enkripciju i upravljanje ključevima, kako bi se sprečila zloupotreba.

```
2 usages  krsnik
public String createToken(User user) {
    Date now = new Date();
    Date validity = new Date(now.getTime() + 3600000);

    return JWT.create()
        .withIssuer(user.getEmail())
        .withIssuedAt(now)
        .withExpiresAt(validity)
        .withClaim( name: "userId", user.getId())
        .withClaim( name: "firstName", user.getFirstName())
        .withClaim( name: "lastName", user.getLastName())
        .withClaim( name: "status", user.getStatus().name())
        .sign(Algorithm.HMAC256(secretKey));
}
```

Slika 17 Generisanje JWT tokena

Sam token se sastoji iz tri dela: zaglavlja, glavnog sadržaja i potpisa koji su spojeni tačkom(.) (Slika 18). Zaglavlje i glavni sadržaj sastoje se iz parova ključ/vrednost u *JSON* formatu i nose informacije o samom tokenu. Potpis se formira od zaglavlja i glavnog sadržaja primenom algoritma za šifrovanje. Jednom kada se token formira on predstavlja tekstualni podatak koji je nečitljiv/nerazumljiv za samog korisnika. Podaci iz tokena se mogu pročitati kada se token dešifruje. Primer JWT tokena i njegovih segmenata dat je u nastavku: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c [14].

Zaglavlje (Header):

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Glavni sadržaj (Payload):

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

Slika 18 Zaglavlje i glavni sadržaj tokena

Validacija se radi na obrnuti način, prvo je potrebno podeliti token na tri dela, nakon čega se obavlja dekodiranje prva dva dela; na kraju, ukoliko od dekodiranih delov' (uz korišćenje postupka koji smo ranije prikazali), nastane potpis koji je isti kao i potpis koji je došao uz token, sledi da je token validan. Kada je token verifikovan možemo iz njega izvući ono sto nas zanima (Slika 19).

```
public Authentication validateToken(String token) {
    Algorithm algorithm = Algorithm.HMAC256(secretKey);
    JWTVerifier verifier = JWT.require(algorithm).build();
    DecodedJWT decodeJWT = verifier.verify(token);
    User user = User.builder()
        .email(decodeJWT.getIssuer())
        .id(decodeJWT.getClaim(name: "id").asLong())
        .firstName(decodeJWT.getClaim(name: "firstName").asString())
        .lastName(decodeJWT.getClaim(name: "lastName").asString())
        .status(UserStatus.valueOf(decodeJWT.getClaim(name: "status").asString()))
        .build();
    return new UsernamePasswordAuthenticationToken(user, credentials: null, Collections.emptyList());
}
```

Slika 19 Dekodiranje JWT tokena

Klijentska strana aplikacije prilikom pozivanja serverskog dela aplikacije preko *HTTP* poziva šalje u *Header* zaglavlju, u *Authorization* delu, token. Serverski deo proverava da li korisnik čiji je token ima pravo pristupa tom zahtevu (Slika 20). U primeru sa slike *POST* metoda na putanji “api/v1/blog” zahteva autentikaciju, sve ostale pitanje su “otvorene”.

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http.csrf(AbstractHttpConfigurer::disable)
        .addFilterBefore(new JwtAuthFilter(userAuthProvider), BasicAuthenticationFilter.class)
        .sessionManagement(
            customizer -> customizer.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .authorizeHttpRequests(
            (request) ->
                request
                    .requestMatchers(HttpMethod.POST, "/api/v1/blog")
                    .authenticated()
                    .anyRequest().permitAll());
    return http.build();
}
```

Slika 20 Autorizacija poziva

Token klijentskoj strani osim što služi da ga šalje i time autorizuje svoj poziv ka serverskoj strani služi i da iz njega dobije potrebne informacije od serverskog dela aplikacije. Informacije koje su mojoj aplikaciji potrebne su id korisnika i njegova rola, da li je admin ili običan korisnik, ugrađenom metodom *atob* dekriptujemo token i izvlačimo iz njega potrebne informacije (Slika 21).

Slika 21 Dekripcija tokena na klijentskoj strani

```
getToken() {
    const tokenDecoded :string = atob(this.getAuthToken().split(' ')[1]);
    return JSON.parse(tokenDecoded);
}

2 usages  🔍 krsnik *
isAdmin(): boolean {
    if (this.getAuthToken() == null) return false;
    let token = this.getToken();
    return token["role"] === "ADMIN";
}

3 usages  🔍 krsnik
getUserId(): number {
    let token = this.getToken();
    return token["id"];
}
```

Slika 21 Dekripcija tokena na klijentskoj strani

Flyway migracije

Flyway migracije je alat koji se koristi za verzioniranje i upravljanje bazama podataka u aplikacijama, posebno u okruženjima koja koriste *SQL* baze podataka. *Flyway* omogućava automatsko primenjivanje skripti za migraciju baze podataka, čime se osigurava da struktura baze podataka (šeme) bude u skladu sa verzijom aplikacije. Osnovni koncept *Flyway* migracija uključuje:

- Skripte za migraciju: Skripte su *SQL* datoteke ili *Java* klase koje definišu promene u bazi podataka, kao što su kreiranje, menjanje ili brisanje tabela, indeksa, pogleda, procedura itd.
- Verzioniranje: Svaka migracija ima jedinstveni identifikator verzije, što omogućava *Flyway* da prati koje su migracije već primenjene, a koje još nisu.
- Automatizacija: *Flyway* automatski prepoznaje i primenjuje nove migracije koje se nalaze u projektu, što olakšava rad sa više verzija aplikacije u različitim okruženjima (npr. razvoj, testiranje, produkcija).
- Reverzibilnost: *Flyway* podržava i mogućnost vraćanja na prethodnu verziju baze podataka ako je to potrebno, iako to zahteva dodatne skripte koje definišu kako se promena može poništiti.

U primeru sa Slika 22, kreirao sam tabelu “review” sa primarnim ključem id koji se samostalno generise i uvećava za 1, sa stranim ključevima na tabelu “user”, jedan predstavlja onog koji je napisao komentar, a drugi onog koji je komentarisao. Pored datih kolona postoje i kolone ocena, tekst komentara i kada je komentar kreiran, sve kolone su označene da moraju da imaju vrednost.

```
DROP TABLE IF EXISTS review;

CREATE TABLE review
(
    id                BIGINT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    rated_user_id     BIGINT NOT NULL,
    creator_user_id    BIGINT NOT NULL,
    rating            INT NOT NULL,
    text              VARCHAR(1000),
    created_at        TIMESTAMP NOT NULL DEFAULT NOW(),

    CONSTRAINT fk_review_rated_user_id
        FOREIGN KEY (rated_user_id) REFERENCES users(id),

    CONSTRAINT fk_review_creator_user_id
        FOREIGN KEY (creator_user_id) REFERENCES users(id)
);
```

Slika 22 Flyway migracija

4.4 Axios servis

Axios je popularna *JavaScript* biblioteka koja se koristi za slanje *HTTP* zahteva iz pregledača. Posebno je cenjena u *Angular* projektima zbog svoje jednostavnosti, efikasnosti i lakoće korišćenja prilikom komunikacije sa serverskim delom. Za razliku od ugrađenog *Angular HttpClient*, *Axios* nudi jednostavniji *API* koji programeri često smatraju lakšim za rad.

Ključne karakteristike *Axios*:

- *Promise-based Axios* koristi *promise* za rukovanje asinhronim operacijama, što olakšava pisanje i upravljanje kodom koji zavisi od *HTTP* zahteva.
- *Interceptors*: *Axios* omogućava programerima da presretnu zahteve ili odgovore pre nego što budu obrađeni, što omogućava dodavanje zaglavlja, logovanje ili rukovanje greškama.
- Automatska obrada *JSON*: *Axios* automatski parsira *JSON* odgovore i pretvara podatke u *JavaScript* objekte, smanjujući potrebu za ručnim parsiranjem.
- Transformacija zahteva i odgovora: *Axios* može transformisati podatke zahteva pre slanja ili promeniti podatke odgovora pre nego što ih prosledi aplikaciji.
- Otkazivanje: Uz *Axios* je moguće otkazati zahtev korišćenjem *CancelToken* funkcije, što je posebno korisno u situacijama kada korisnik napusti stranicu pre nego što je zahtev završen.
- Jednostavnost: *Axios* nudi jednostavniji i intuitivniji *API* u poređenju sa *Angular HttpClient*, što ga čini lakšim za razumevanje i korišćenje, posebno za nove programere.
- Kompatibilnost sa pregledačima: *Axios* radi besprekorno u modernim pregledačima i rešava probleme kompatibilnosti između različitih pregledača koji se mogu pojaviti pri korišćenju *native fetch* ili drugih *HTTP* biblioteka.

Iako *Angular* pruža svoj *HttpClient* za rukovanje *HTTP* zahtevima, *Axios* ostaje moćna alternativa koju mnogi programeri preferiraju zbog lakoće korišćenja i naprednih funkcija poput presretna i otkazivanja zahteva. Bilo da radite na malom projektu ili na aplikaciji velikih razmera, integracija *Axiosa* u vaš *Angular* projekat može značajno unaprediti vaše mogućnosti za upravljanje *HTTP* komunikacijom sa serverskim delom aplikacije [15].

Na Sliku 23 u konstruktoru-u prikazano je podešavanje *base-url*, koji će klijentski deo aplikacije da gađa i tip sadržaja koji će se nalaziti u njihovoj komunikaciji. U metodi *requestWithToken* prikazano je generisanje *header* zaglavlja odnosno *Authorization* polja unutar njega, ono se formira sa prefiksom “*Bearer* ” a u nastavku je token ulogovanog korisnika. Nakon toga se poziva metoda *request* *axios* objekta koja za prvi parametar prihvata metod koji se poziva (“*POST*”, “*GET*”, “*DELETE*”, “*UPDATE*”), *uri* sufiks koji se dodaje na *base-url*, podatke koji se šalju uz dati poziv i *header*.

```
import axios from "axios";

5+ usages  ⓘ krsnik *
@Injectable({
  providedIn: 'root'
})
export class AxiosService {

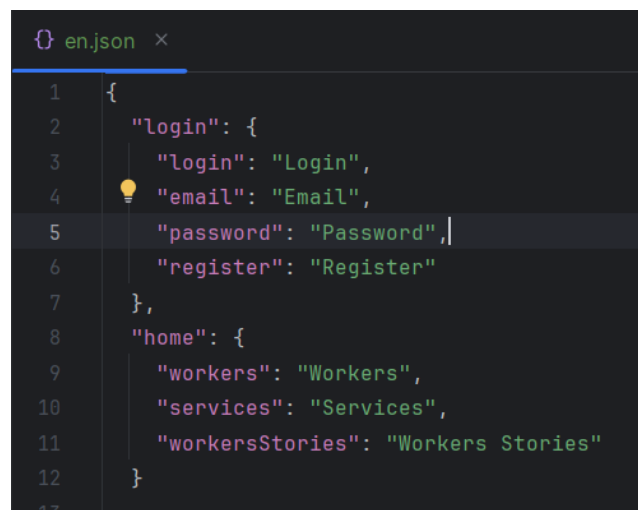
  no usages  ⓘ krsnik
  constructor() {
    axios.defaults.baseURL = 'http://localhost:8080';
    axios.defaults.headers.post['Content-Type'] = 'application/json';
  }

  5+ usages  new *
  requestWithToken(method: string, uri: string, data: any): Promise<any> {
    let headers :{} = {};
    if (this.getAuthToken() !== null) {
      headers = {
        Authorization: "Bearer " + this.getAuthToken()
      };
    }
    return axios.request( config: {
      method: method,
      url: uri,
      data: data,
      headers: headers
    });
  }
}
```

Slika 23 Axios servis

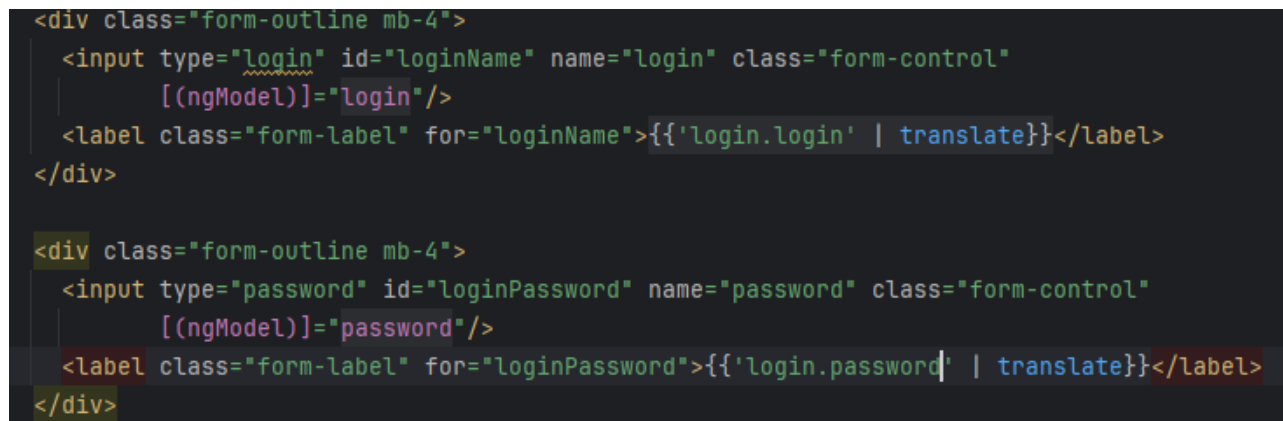
4.5 Dvojezičnost

Dvojezičnost je ostvarena uz pomoc biblioteke *ngx-transalte*. Potrebno je imati za svaki od podržanih jezika po jedan *json* file u kome će se nalaziti prevodi za taj jezik (Slika 25). Zatim je dovoljno unutar *HTML* fajla samo se referencirati na želejenu poruku u *json* fajlu (Slika 24Slika 24).



```
en.json x
1 {
2   "login": {
3     "login": "Login",
4     "email": "Email",
5     "password": "Password",
6     "register": "Register"
7   },
8   "home": {
9     "workers": "Workers",
10    "services": "Services",
11    "workersStories": "Workers Stories"
12  }
13 }
```

Slika 25 JSON sa prevodom



```
<div class="form-outline mb-4">
  <input type="login" id="loginName" name="login" class="form-control"
    [(ngModel)]="login"/>
  <label class="form-label" for="loginName">{{'login.login' | translate}}</label>
</div>

<div class="form-outline mb-4">
  <input type="password" id="loginPassword" name="password" class="form-control"
    [(ngModel)]="password"/>
  <label class="form-label" for="loginPassword">{{'login.password' | translate}}</label>
</div>
```

Slika 24 Dvojezičnost na klijentsom delu aplikacije

4.6 Učitavanje slike

Jedan od interesantnijih delova koda na serverskoj i klijentskoj strani je učitavanje slike.

Klijentska strana

Na Sliku 26 je prikazana *HTML* stranica na kojoj se nalazi *input* polje tipa *file*, prilikom unosa fajla/fajlova, okida se događaj koji okida metodu u komponenti prikazanu na Sliku 27. Na ulazu u metodu prikazana je provera gde proveravamo da li je korisnik stvarno odabrao datoteke i da li prva datoteka nije *undefined*. Iteriranjem kroz sve unete fajlove kreiramo *FileReader* objekat koji omogućava čitanje sadržaja datoteke. *Onload* funkcija se izvršava kada *FileReader* završi učitavanje datoteke, učitani sadržaj, koji je u formatu *Base64*, se dodaje u *this.images* niz. *ReadAsDataURL* metoda čita sadržaj trenutne datoteke u formatu *Base64*, što je pogodna za prikazivanje slike direktno u *HTML*.

```
<input class="form-control" id="fileId" type="file" placeholder="Slike" required name="file"
      (change)="onFileSelected($event)" accept=".jpg, .png" multiple />
label>
```

Slika 26 HTML input za učitavanje slike

```
onFileSelected(event) :void {
  if (event.target.files && event.target.files[0]) {
    this.imagesOk = true;
    for (let i : number = 0; i < event.target.files.length; i++) {
      const reader : FileReader = new FileReader();

      reader.onload = (event: any) :void => {
        this.images.push(event.target.result);
      };

      reader.readAsDataURL(event.target.files[i]);
    }
  }
}
```

Slika 27 Obrada event-a učitavanja

Serverska strana

Na serverskoj strani za mapiranje domenskih objekata u entitetske je prikazano korišćenje *mapstruct* mapera (Slika 28). Anotacija *@Mapping* specificira kako se pojedinačna polja iz jednog objekta mapiraju na polja u drugom objektu. *Source* polje predstavlja polja izvornog objekta dok *target* polje ciljnog objekta, *qualifiedByName* označava specifičnu metodu koja se koristi tokom mapiranja, u ovom slučaju je to *decodeImage* odnosno *encodeImage*. Domenski objekat sadrži atribut tipa *String* koji dobijemo sa klijentske strane prilikom dodavanja slika, a entitetski objekat sadrži atribut tipa *byte[]*, koje odgovara polju tipa *mediumblob* u bazi i zbog toga je neophodno uraditi gore pomenuto mapiranje.

```
@Mapper(componentModel = "spring")
public interface BlogImageMapper {

    no usages
    ServiceImageMapper INSTANCE = Mappers.getMapper(ServiceImageMapper.class);

    1 implementation  ↕ krsnik
    @Mapping(source = "imageData", target = "imageData", qualifiedByName = "decodeImage")
    BlogImageEntity toEntity(BlogImage domain);

    1 implementation  ↕ krsnik
    @Mapping(source = "imageData", target = "imageData", qualifiedByName = "encodeImage")
    BlogImage toDomain(BlogImageEntity entity);

    1 usage  ↕ krsnik
    @Named("decodeImage")
    default byte[] decodeImage(String base64Image) {
        return Base64.getDecoder().decode(base64Image.split(regex: "[,]"[1]);
    }

    1 usage  ↕ krsnik
    @Named("encodeImage")
    default String encodeImage(byte[] image) {
        return "data:image/png;base64," + Base64.getEncoder().encodeToString(image);
    }
}
```

Slika 28 Mapper na serverskoj strani

5 Korisničko uputstvo

U ovom delu će najpre biti opisane kategorije korisnika a zatim i funkcije koje sistem pruža različitim kategorijama korisnika. Funkcije će biti detaljno opisane uz priložene slike.

5.1 Neregistrovani korisnici

Neregistrovani korisnici predstavljaju većinsku grupu koja će posećivati ovaj sajt. To su klijenti koji žele da pronađu kvalitetne majstore sa proverenim referencama koji će ih učiniti zadovoljnim. Oni će moći da pretražuju majstore po kategorijama koje ih interesuju. Postoji dva nivoa kategorija, L1 i L2 kategorije, koje su podeljene radi lakšeg snalaženja. Na primer: L1 kategorija, građevinski radovi će imati svoje podkategorije L2 : moler, parketar, fasader, zidar, keramičar itd.

Neregistrovani korisnik će moći i da čita blogove, priče majstora, kako bi mogao što bolje da se informiše na temu radova koji ga zanimaju. Detaljnije o svim funkcionalnostima neregistrovanog korisnika u nastavku.

5.2 Registrovani korisnici

Registrovani korisnici su nadskup neregistrovanih korisnika, sve što su mogli neregistrovani korisnici mogu i registrovani korisnici. Registrovani korisnici mogu biti majstori koji su kreirali nalog kako bi mogli da objave svoje usluge, ali takođe i klijenti koji žele da ocene svog majstora. Registrovani korisnici mogu da pišu blogove o temama u kojima su stručni, time će postati vidljiviji zajednici i njihove usluge će biti traženije. Registrovani korisnik osim što može dodavati svoje blogove i usluge, može ih i brisati.

Administratori sistema su nadskup registrovanih korisnika, sve što su mogli registrovani korisnici mogu i administratori, pored toga oni imaju ulogu da vode računa o sajtu i mogu da brišu sve usluge i blogove.

5.3 Prijavljivanje postojećih korisnika

Ova opcija se odnosi na korisnike koji su već registrovani imali oni običnu ili administratorsku ulogu u sistemu. Unošenjem korisničkog imena i lozinke, koji se moraju poklapati sa postojećim podacima o korisnicima u bazi podataka, pristupa se nalogu (Slika 29).

Početna

Majstori Majstorske priče Kontakt Prijavi se

Login Register

nikola@gmail.com
Email

.....
Password

Sign in

Majstor f X i

Početna Majstori Majstorske priče Kontakt

Copyright 2024. Sva prava zadržana. Andrej & Nikola

Slika 29 Prijavljivanje

5.4 Registracija korisnika

Neregistrovani korisnik ima mogućnost dodavanja novog korisnika unošenjem svojih podataka (ime, prezime, mejl, telefon, grad, šifra). Potreba za registrovanje korisnika se javlja kada neko želi da postane majstor i okači svoje usluge, napiše blog ili ostavi komentar na usluge majstora (Slika 30).

Login Register

Nikola
First name

Krstic
Last name

nikola@gmail.com
Email

0631897714
Telefon

Beograd
Grad

.....
Password

Sign up

Slika 30 Registracija

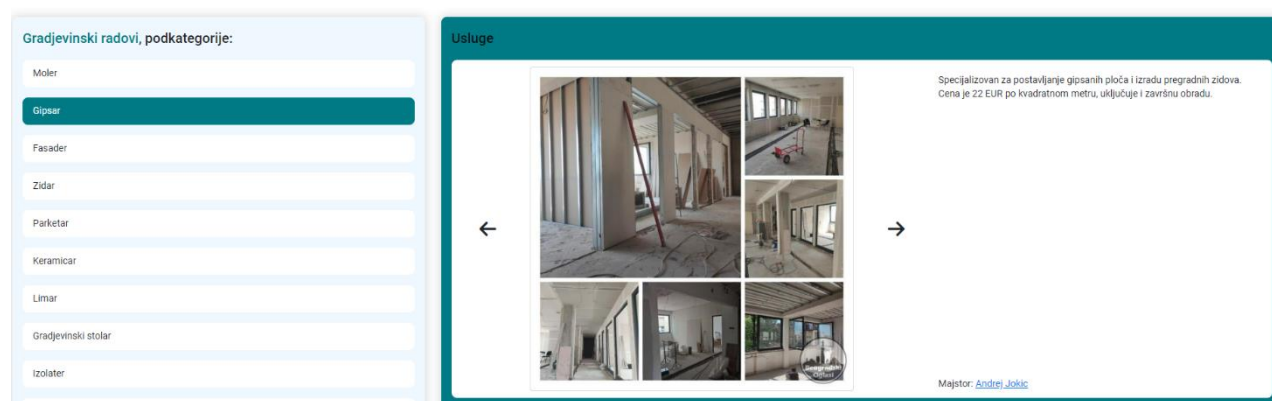
5.5 Pretraga majstora

Pretraživanje majstora je dostupno svim korisnicima, vrši se sa početnog ekrana, gde se odabira L1 kategorija (Slika 31) .(Građevinski radovi, Elektronika, Održavanje, Vodovod i sanitarije, Obrada materijala , Održavanje vozila)



Slika 31 Odabir L1 kategorije

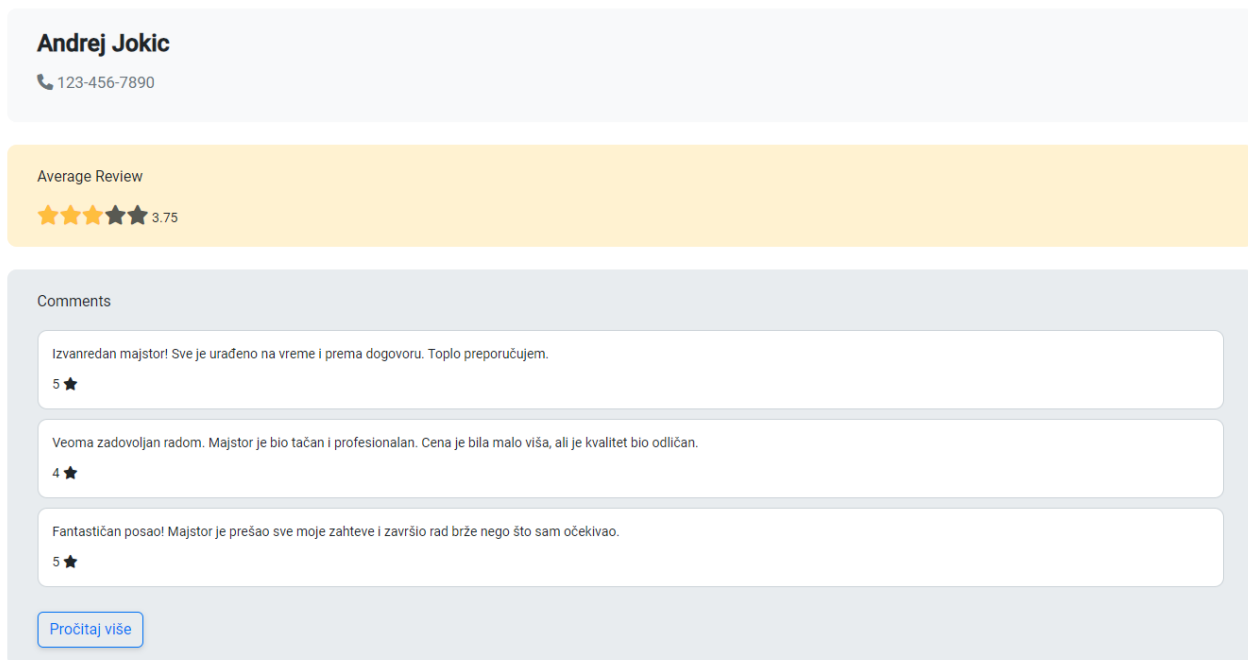
Kada se kline na L1 kategorija otvara nam se nova stranica(Slika 32) na kojoj imamo izlistane podkategorije izabrane L1 kategorije sa leve strane. Klikom na željeni L2 kategoriju u desnom deslu stranice dobićemo izlistane usluge te podkategorije. Usluge u svom levom delu sadrže galeriju slika prethodnih radova iz te podkategorije tog majstora, detaljan opis same usluge i pri samom dnu ime majstora, klikom na majstora ulazi se na profil majstora.



Slika 32 L2 kategorija i usluge

5.6 Profil korisnika/majstora

Na profilnoj stranici korisnika možemo videti njegov kontakt, prosečnu ocene sa kojom su ga klijenti ocenili, komentare njegovih usluga (Slika 33).



Andrej Jokic
📞 123-456-7890

Average Review
★★★★★ 3.75

Comments

Izvanredan majstor! Sve je urađeno na vreme i prema dogovoru. Toplo preporučujem.
5 ★

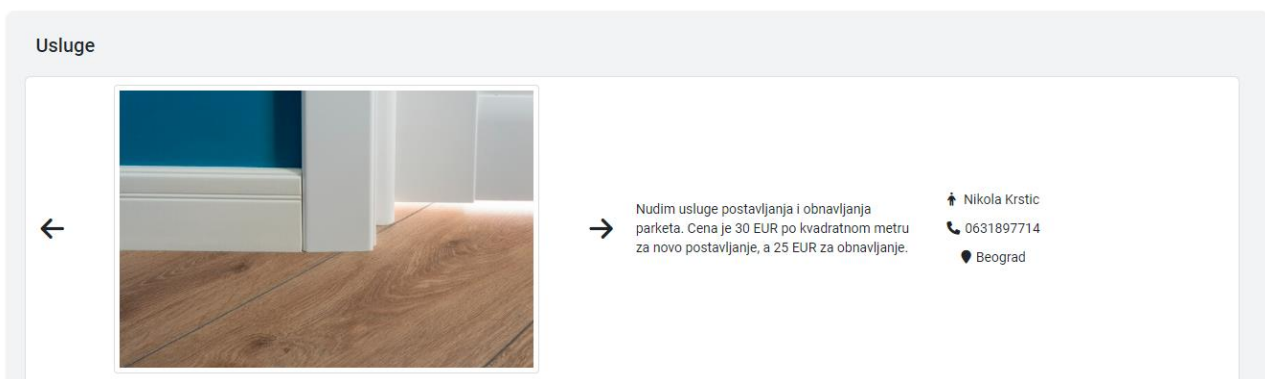
Veoma zadovoljan radom. Majstor je bio tačan i profesionalan. Cena je bila malo viša, ali je kvalitet bio odličan.
4 ★

Fantastičan posao! Majstor je prešao sve moje zahteve i završio rad brže nego što sam očekivao.
5 ★

[Pročitaj više](#)

Slika 33 Profil korisnika 1.deo

Ispod ovog dela možemo videti i sve njegove usluge izlistane, slično kao i na početnom ekranu kad smo tražili po podkategorijama, možemo videti galeriju slika njegove usluge, detaljan opis i njegove kontakt podatke (Slika 34).



Usluge

←  →

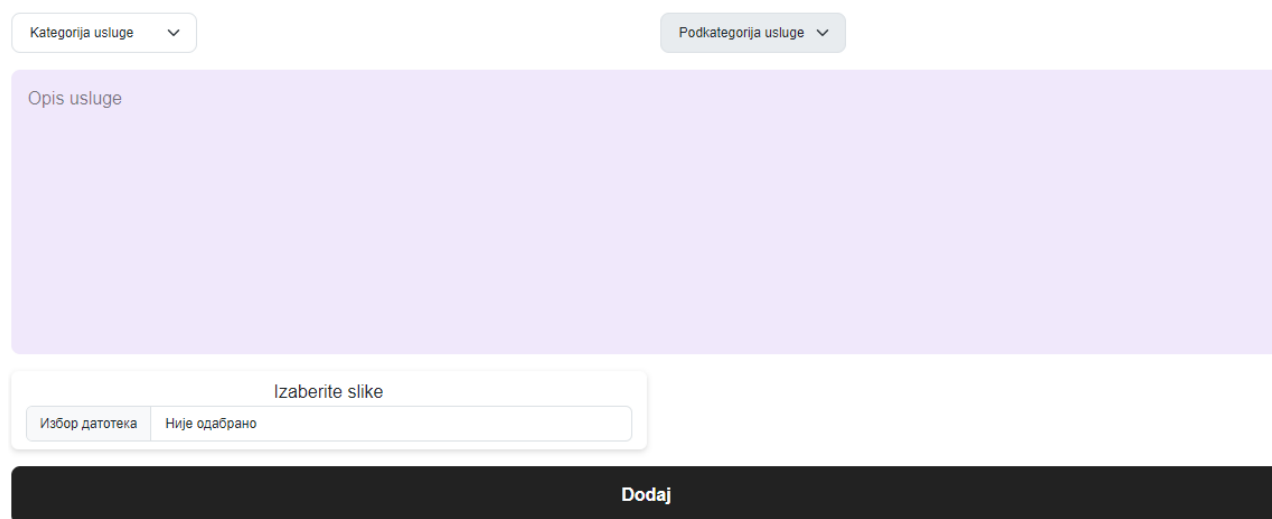
Nudim usluge postavljanja i obnavljanja parketa. Cena je 30 EUR po kvadratnom metru za novo postavljanje, a 25 EUR za obnavljanje.

👤 Nikola Krstic
📞 0631897714
📍 Beograd

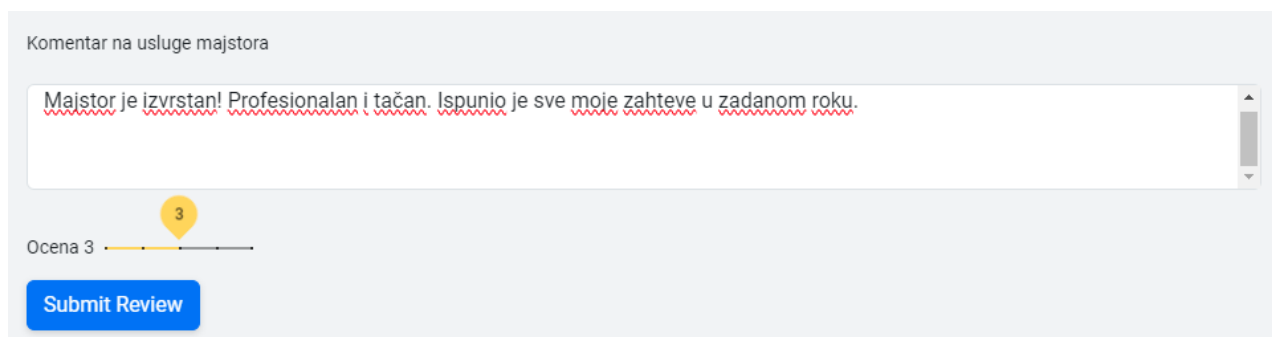
Slika 34 Profil korisnika 2.deo

5.7 Dovanje usluge i recenzije

Ukoliko se nalazite na svom profilu možete da dodate svoju novu uslugu, odabirmo L1 i L2 kategorije, unosom opisa usluge i dodavanjem slika vaših prethodnih radova željene usluge. (Slika 35) Ukoliko se nalazite na tuđem profilu, možete ostaviti recenziju za usluge majstora dodavanjem komentara i odabirom ocene od 1 do 5 (Slika 36).



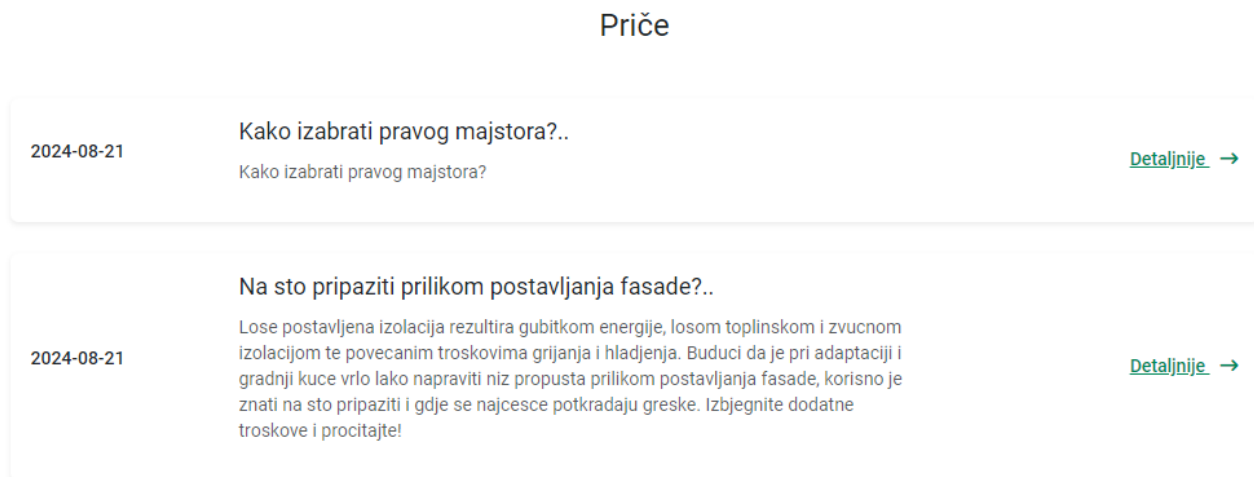
Slika 35 Dodavanje usluge



Slika 36 Dodavanje komentara

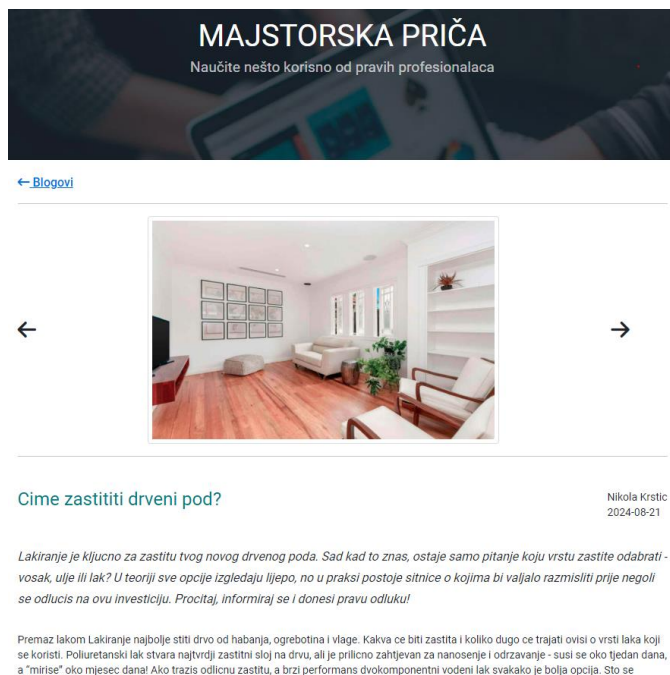
5.8 Blogovi

Svi korisnici mogu da čitaju blogove (Slika 37). Blogovi omogućavaju majstorima da dele svoja iskustva i savete, što je veoma korisno za ljude koji su u procesu radova u svom domu. Blogovi mogu biti odličan način da klijenti dođu do majstora čiji blog ih je zainteresovao.



Slika 37 Blogovi

Kad se uđe u blog korisnici na vrhu imaju galeriju slika tog bloga, datum kreiranja bloga, autora odnosno majstora koji je kreirao blog i tekst bloga. Klikom na ime majstora korisnik će otići na njegovu profilnu stranicu (Slika 38).



Slika 38 Blog

6 Zaključak

Tema ovog master rada je izrada aplikacije za pronalaženje majstora. Potreba za ovom aplikacijom se javila jer je u velikim gradovima sve teže doći do kvalitetnih i proverenih majstora. Aplikacija za pronalaženje majstora omogućuje korisniku pretragu majstora po kategorijama, pregled njegovih radova, utiske njegovih prethodnih mušterija, pregled blogova koje su pisali majstori i koji su informativnog karaktera za ljude kojima su neophodni radovi u domu.

Korišćenje *Spring Boot* i *Angular* radnog okvira se pokazalo kao odlično, jer je pružilo jednostavnost i intuitivnost prilikom programiranja. *Spring Boot* je omogućio brzo i lako postavljanje serverskog dela, s minimalnom konfiguracijom. *Angular* je, s druge strane, pružio strukturiran pristup izradi klijentog dela, što je olakšalo razvoj i održavanje korisničkog dela. Zajedno, ove tehnologije su omogućile efikasnu realizaciju projektnih zadataka uz visoku produktivnost i ugodno iskustvo programiranja. Posebno se istaklo korišćenje *Flyway* migracija u *Spring Boot* koje je olakšalo zajednički rad na serverskom delu sa kolegom Andrejem Jokićem, koji je radio na mobilnoj verziji aplikacije. *Flyway* je omogućio jednostavno upravljanje verzijama baze podataka, što je bilo ključno za koordinaciju tima i osiguranje konzistentnosti podataka kroz različite faze razvoja. Ova kombinacija tehnologija je omogućila fokus na funkcionalnosti aplikacije, bez brige o problemima sa bazom podataka ili kompatibilnošću kodnog okruženja, što je dodatno doprinelo uspešnom završetku projekta.

Drugi deo rada je testiranje aplikacije, testiranje omogućuje proveru da li aplikacija ispravno radi. Aplikacija za pronalazak majstora je testirana pomoću dva testa: testiranje jedinice (*Unit test*), celovito testiranje (*End-to-End test*). Testiranje jedinice omogućuje da se testira samo jedan deo aplikacije, dok se celovitim testiranjem aplikacija testira od početka do kraja, odnosno kod celovitog testiranja je potrebno simulirati stvarnog korisnika u pretraživaču.

Testiranje jedinice je teklo paralelno sa razvojem projekta funkcionalnost po funkcionalnost ali i kad je projekat završen svaka stavka je detaljno testirana. Jedinično testiranje je otkrilo dosta propusta u aplikaciji koji su blagovremeno otklonjeni.

Celovito testiranje je sprovedeno zajedno sa kolegom Andrejem Jokićem koji je radio na mobilnoj verziji aplikacije. U ovoj fazi su uočene greške na sva tri dela aplikacije serverskom, veb i mobilnom delu i one su otklonjene u najkraćem roku. Unutar tima su razmenjene uočene greške koje su ispravljene i mane u dizajnu od kojih su neke prihvaćene.

U toku same izrade projekta pojavljivali su se problemi koji su rešavani u toku izrade i koji nisu oduzeli previše vremena. Nakon odrađene aplikacije kao stavka koja je mogla biti drugačije urađena je instanciranje administratora u sistem, jedini način da se on doda je ručno slanje *API* zahteva, pa nekome ko nije stručan to bi moglo da predstavlja problem. Rešenje za to bi bila mogućnost dodavanja administratora iz same aplikacije preko korisničkog grafičkog interfejsa.

Od starta je postojala želja za čuvanjem slika na *S3 bucket*, ali to nije bilo moguće jer je aplikacija razvijana u lokalnu. *S3* predstavlja servis za čuvanje velike količine podataka. Podaci se šalju preko internog *API*, konzole ili nekim od dostupnih alata. *S3* se može zamisliti kao ogroman *hard-disk*, gde su podaci organizovane u tzv. kofe (*buckets*), i u okviru kofe u direktorijume. U pozadini, međutim, direktorijumi ne postoje, svi fajlovi se nalaze na istom nivou, i dostupni su po svom imenu. *S3* obezbeđuje neke specifične opcije za baratanje fajlovima, npr. brisanje fajlova nakon određenog vremena i arhiviranje na *storage* dubljeg nivoa. Takođe, garantuje se i bezbednost podataka, budući da se svi podaci smeštaju kao dve istovetne replike. [16] Kao zamensko rešenje odabrano je čuvanje slika u lokalnu u blob formatu što zna da bude veoma limitirajuće.

Eventualna poboljšanja aplikacije mogu biti ta da korisnik zakaže majstora preko aplikacije. Tu opciju je potrebno bolje razmotriti jer se dovodi u pitanje da li je lakša u odnosu na direktan kontakt majstora sa klijentom preko telefona.

Reon koji majstor pokriva se može prikazati pomoću mape, međutim tačno označavanje samog reona može biti komplikovana funkcionalnost za majstora koji bi morao pinovima da označava reon unutar kog je operativan, zato je i ostavljena opcija za slobodan unos za opis usluge, gde bi majstor to rečima definisao. Moguće je i uvesti opciju plaćenog oglasa gde bi se usluge sponzorisanih majstora prikazivale na vrhu usluga iz selektovane kategorije.

Prostor za unapređenje postoji i u segmentu čuvanja poverljivih informacija korisnika (mobilni broj telefon, email, šifra), pošto je aplikacija razvijana u lokalu u sledećim koracima bi trebalo napraviti integraciju sa *Auth* provajderom. *Auth* provajder predstavlja komponentu koja upravlja procesom autentikacije korisnika i čuva njihove podatke na način koji osigurava njihovu bezbednost i privatnost.

Literatura

- [1] D. Megida, "What is JavaScript," 8 2024. [Online]. Available: <https://www.freecodecamp.org/news/what-is-javascript-definition-of-js/>.
- [2] "Šta je JavaScript," 8 2024. [Online]. Available: <https://www.oxfordwebstudio.com/dali-znate/sta-je-javascript.html>.
- [3] 8 2024. [Online]. Available: <https://sr.m.wikipedia.org/sr/AngularJS>.
- [4] 8 2024. [Online]. Available: <https://material.angular.io/>.
- [5] M. A. Ayaz, Angular kuvar, Beograd: Vulkan, 2021.
- [6] "Java Introduction," 8 2024. [Online]. Available: https://www.w3schools.com/java/java_intro.asp.
- [7] "Java (programming language)," 8 2024. [Online]. Available: [https://en.wikipedia.org/wiki/Java_\(programming_language\)](https://en.wikipedia.org/wiki/Java_(programming_language)).
- [8] "Introduction to Java," 8 2024. [Online]. Available: <https://www.geeksforgeeks.org/introduction-to-java/>.
- [9] "Spring Tutorial," 8 2024. [Online]. Available: <https://www.baeldung.com/spring-tutorial>.
- [10] "Introduction to Spring Framework," 8 2024. [Online]. Available: <https://www.geeksforgeeks.org/introduction-to-spring-framework/>.
- [11] "What is MySQL?," 8 2024. [Online]. Available: <https://www.talend.com/resources/what-is-mysql/>.
- [12] "MySQL Tutorial," 8 2024. [Online]. Available: <https://www.javatpoint.com/mysql-tutorial>.
- [13] "MVC arhitektura," 8 2024. [Online]. Available: https://sr.wikipedia.org/wiki/MVC_arhitektura.
- [14] A. Bošnjak, "Zaštita vaše aplikacije na jednostavan način," 8 2024. [Online]. Available: <https://ftninformatika.com/zastita-vase-aplikacije-na-jednostavan-nacin/>.
- [15] "Axios Service," 8 2024. [Online]. Available: <https://axios-http.com/docs/intro>.
- [16] M. Savic, "Servisi AWS-a," 8 2024. [Online]. Available: <https://startit.rs/aws-amazon-web-services/>.

Spisak skraćenica

MySQL	My Structured Query Language
AWS	Amazon Web Services
GCP	Google Cloud Platform
JVM	Java Virtual Machine
OOP	Objektno orijentisano programiranje
REST	Representational state transfer
RDBMS	Relational database management system
MVC	Model View Control
JWT	JSON Web Token
HTTP	HyperText Transfer Protocol
API	Application Programming Interface
SQL	Structured Query Language
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
DHTML	Dynamic HTML

Spisak slika

Slika 1 Podkategorija vodoinstalaterske usluge	3
Slika 2 Blogovi.....	4
Slika 3 Profil majstora i galerija njegovih radova	5
Slika 4 Ocene i komentari majstora.....	6
Slika 5 Znam majstora	7
Slika 6 Forma za stupanje u kontak sa majstorima, E- majstori.....	8
Slika 7 JavaScript.....	11
Slika 8 Angular.....	12
Slika 9 Klijent server komunikacija	13
Slika 10 Angular komunikacija sa serverom.....	13
Slika 11 Java	16
Slika 12 SpringBoot.....	17
Slika 13 MySQL.....	18
Slika 14 Tabela u bazi.....	19
Slika 15 Model-View-Controller.....	20
Slika 16 Dijagram klasa	22
Slika 17 Generisanje JWT tokena	23
Slika 18 Zaglavlje i glavni sadržaj tokena	24
Slika 19 Dekodiranje JWT tokena	24
Slika 20 Autorizacija poziva	25
Slika 21 Dekriptacija tokena na klijentskoj strani	25
Slika 22 Flyway migracija	26
Slika 23 Axios servis	28
Slika 24 Dvojezičnost na klijentsom delu aplikacije	29
Slika 25 JSON sa prevodom.....	29
Slika 26 HTML input za učitavanje slika	30
Slika 27 Obrada event-a učitavanja	30
Slika 28 Mapper na serverskoj strani.....	31
Slika 29 Prijavljivanje.....	33
Slika 30 Registracija	33
Slika 31 Odabir L1 kategorije	34
Slika 32 L2 kategorija i usluge.....	34
Slika 33 Profil korisnika 1.deo	35
	42

Slika 34 Profil korisnika 2.deo	35
Slika 35 Dodavanje usluge.....	36
Slika 36 Dodavanje komentara.....	36
Slika 37 Blogovi.....	37
Slika 38 Blog.....	37

Spisak tabela

Tabela 1 Funkcionalnosti postojećih rešenja.....	9
--	---