IZRADA PROGRAMSKOG RJEŠENJA ZA PRUŽANJE USLUGA AUTOMEHANIČARA

Ćosić, Krešimir

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

IZRADA PROGRAMSKOG RJEŠENJA ZA PRUŽANJE USLUGA AUTOMEHANIČARA

Krešimir Ćosić

Zagreb, veljača 2023.

Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada.

U Zagrebu, 29.03.2023.

Predgovor

Želim se zahvaliti svim profesorima i asistentima na Visokom učilištu Algebra za njihov profesionalizam i posvećenost učenju svojih studenata. Posebno želim zahvaliti mentoru Danielu Beleu. Njegovo strastveno razmišljanje o programiranju i motiviranju studenata me motiviralo i ohrabrilo od prvog nastavnog sata iz programiranja. Također se zahvaljujem sestri Mireli koja me podržala, te ženi Ivani koja me pratila tijekom cijelog studija.

Sažetak

U doba gdje se ponuda automobila sve više širi, potraga za automehaničarskim uslugama i dalje ostaje nepromijenjena. Kako bi klijentima omogućili brz, jednostavan i relevantan pristup informacijama, napisana je Android aplikacija Car Medic. Aplikacija pomaže klijentima da odaberu automehaničarsku radionicu, a istovremeno poboljšava poslovanje automehaničara kroz pristup i praćenje zahtjeva. Car Medic je razvijen u Kotlin programskom jeziku, temelji se na Microsoft SQL Server bazi podataka. Poslužiteljski razvojni okvir ASP.NET Web API 2 napisan je u C# jeziku. Ovaj rad detaljnije objašnjava ključne funkcije, tehnologije, jezike i implementaciju Car Medic aplikacije.

Ključne riječi: Android aplikacija, Kotlin, programsko rješenje

Abstract

The market for cars is increasing, but the search for car repair workshops is still carried out in the same way as it was a long time ago. In order to provide clients with fast, transparent, and important information about car repair services, an Android mobile application called Car Medic has been developed. The application makes it easier for the client to choose a car repair workshop, but at the same time it optimizes the business and enables them to track requests and market themselves. The application is written in the Kotlin programming language, the database is Microsoft SQL Server, and the server development framework is ASP.NET Web API 2, which is written in the C# programming language. This work describes in detail the key functionalities, technologies, languages, and implementation of the Car Medic software solution.

Keywords: Android application, Kotlin, software solution

Sadržaj

1. Uvod

Svi vlasnici automobila, kao i oni koji ga samo povremeno koriste, moraju katkad posjetiti radionicu za popravak automobila. Iako broj automobila na tržištu stalno raste, broj radionica ne prati taj trend, što znači da je za mnoge teško pronaći radionicu u trenutku kada je to potrebno.

Cilj aplikacije Car Medic je olakšati pronalaženje najbliže automehaničarske radionice pomoću pametnog telefona prema dva kriterija- najbliže i od korisnika najbolje ocijenjena. Ova Android aplikacija ima funkciju geolokacije koja omogućuje korisniku da pronađe najbližu opciju. Klijenti također mogu provjeravati termine, pregledavati recenzije i opisivati svoj problem, što omogućuje lakše donošenje odluke s manjim rizikom. Aplikacija također pomaže automehaničarima, omogućujući im da koriste inovativno sredstvo za promociju svojih usluga i povećanje vidljivosti na tržištu, kao i da filtriraju zahtjeve, dogovaraju pregled problema i šalju ponude za plaćanje.

U drugom dijelu ovog rada opisana je trenutna situacija na tržištu automehaničara, kao i provedena anketa o zadovoljstvu Hrvata automehaničarskim uslugama.

U trećem dijelu rada govori se o troslojnoj arhitekturi sustava te se detaljnije objašnjavaju korištene tehnologije, programski jezici i razvojna okruženja koja su korištena za izradu Car Medic programskog rješenja. Kroz klijentski sloj spominje se Android Studio te objašnjava programski jezik Kotlin. Poslužiteljski sloj objašnjava svrhu ASP.NET Web API-ja u aplikaciji, i opisuje ulogu Entity okvira objektno relacijskog mapiranja. U podatkovnom sloju govori se o Microsoft SQL Serveru.

Četvrti dio rada opisuje samu implementaciju sustava kroz navedene tehnologije, daje primjere kodova i objašnjenja zašto se u kojem slučaju nešto koristilo.

Peti dio opisuje gotove funkcionalnosti s primjerima i slikama korisničkog sučelja aplikacije, dok posljednji dio uključuje testiranje programskog rješenja.

2. Opis problema pronalaska automehaničara

Prema istraživanju Eurostata iz 2020. godine, u Hrvatskoj na 1000 ljudi imamo više od 400 automobila (Jadran 466 automobila, kontinentalna Hrvatska 413). Drugim riječima, gotovo svaki drugi Hrvat posjeduje osobno vozilo.[1] Svaki automobil potrebno je redovito održavati i servisirati. Čak i tada često se događaju neplanirani i nenadani popravci. To znači da gotovo pola stanovnika Hrvatske treba automehaničara.

Centar za vozila Hrvatske objavio je statistiku za 2021. godinu prema kojoj je vidljivo kako je prosječna starost automobila 14,34 godine. Zabrinjavajući podatak je i da iste godine skoro 20% automobila nije prošlo tehnički pregled iz prvog pokušaja.[2] Iako istraživanje nije obuhvatilo održavanje automobila, na temelju ovih brojeva može se zaključiti da Hrvati ne posjećuju automehaničare dovoljno često.

Iako je broj automobila prilično velik, broj automehaničara se smanjuje. Potražnja za automehaničarima daleko je veća od ponude. Mlade se potiče da upisuju obrtnička zanimanja, no taj je broj i dalje premali da bi se zadovoljile potrebe vozača. Osim toga, platforma za zapošljavanje Adorio navodi kako je prosječna plaća automehaničara u Hrvatskoj 6412 kn [3], pa se mnogi odlučuju na rad u inozemstvu gdje su za isti posao puno više plaćeni. Stoga ne treba čuditi što je u prosincu 2022. godine na istom portalu prikazano 305 oglasa za posao automehaničara. [4]

Navedena istraživanja i podaci ukazuju na problem - automehaničara nije lako pronaći. U običnoj društvenoj komunikaciji o toj temi kao problem nameće se manjak raspoloživih automehaničara u trenutku kada je osobi usluga potrebna odmah. Veliki servisi automobila se dogovore unaprijed, planirani su te se na njih moguće unaprijed pripremiti - pronaći alternativu za vrijeme bez automobila te osigurati novčana sredstva. Što kada se kvar na automobilu dogodi iznenada te je nužan hitan popravak? Tad je bitan faktor hoće popravak kod automehaničara biti za sat, dva dana, tjedan ili mjesec dana. Vozač je prepušten traženju alternativnog prijevoza te kalkuliranju financija zbog kvara automobila

Kao jedno od mogućih rješenja nameće se mobilna aplikacija koja pokazuje pouzdane automehaničarske radionice u blizini te omogućuje transparentne i jasne informacije na temelju kojih je moguće donijeti optimalan odabir.

2.1. Psihologija odabira usluge

Odabir optimalne automehaničarske radionice može se usporediti s odabirom proizvoda odnosno usluge. Sve navedeno uključuje ispunjavanje neke naše potrebe u zamjenu za novac.

Iako postoji mnogo teorija o načinima i motivaciji kako ljudi dolaze do odluke kod odabira usluge ili proizvoda, svi se psiholozi slažu u jednome - što je veći izbor opcija, to je teže donijeti odluku. Veliki broj mogućnosti povećava strah od krivog odabira i dovodi do osjećaja preplavljenosti i frustracije. U današnjem svijetu su informacije udaljenje jedan klik. S druge strane, iako su lako dostupne, velike količine informacija iziskuju i veliku količinu vremena koje je potrebno za pronalaženje najkorisnije informacije. Upravo zato pomaže filtriranje i isticanje samo najvažnijih informacija o usluzi/proizvodu. [5]

Ljudi donose odluke na temelju različitih motivacija. Jedne od poznatijih teorija ekonomije su teorija ukupne i granične korisnosti koja kaže da svaku odluku donosimo na temelju najvećeg potencijalnog benefita. Osim racionalnog i proračunatog odabira, na odluke utječu i suptilnije marketinške poruke, uokvirivanje pitanja, pa čak i medijska manipulacija. Također, postoje teorije poput *status quo* koje ističu da se, zbog averzije prema riskiranju i promjeni, ljudi odlučuju za poznate, stare opcije, radije nego da prihvate nove, bez obzira na to što su možda bolje. [6]

Sve ove teorije i faktori donošenja odluka mogu se povezati s odabirom automehaničarske radionice. Upisivanjem riječ ‘automehaničar’ u Google proizlazi veliki izbor automehaničarskih radionica, u blizini ili nešto dalje, prema Google parametrima koji osobi koja pretražuje u tom trenutku možda uopće nisu važni. Osim toga, za pronalazak bitne informacije poput kontakta, radnog vremena, lokacije i slično, svi se dobiveni linkovi moraju otvoriti i pregledati. U ubrzanom svijetu, za to najčešće nema vremena. S obzirom na preplavljenost informacijama i zbog uštede vremena, ljudi se nekad odlučuju impulzivno, a ne promišljeno i racionalno. Također, zbog straha od promjene, često ostaju kod istog mehaničara, bez obzira na to što takvo rješenje možda nije optimalno.

2.2. Digitalna rješenja

Godinama se priča o digitalnom dobu i digitalnoj tranziciji, no u nekim djelatnostima i društvenim područjima i dalje postoji otpor prema digitalnim rješenjima. Automehaničarska djelatnost je jedna od njih. Vozači mehaničare najčešće zovu telefonom, a samo rijetki mehaničari komuniciraju e-poštom. Gotovo pa im i ne prođe kroz glavu da bi automehaničarska radionica uopće mogla imati e-poštu. Najčešće je to jer 'automehaničar nema vremena tipkati'. S druge strane, kako ima vremena javljati se na brojne dolazne pozive? Ne bi li bilo jednostavnije da upite može odgoditi za njemu prikladno vrijeme, kada se može posvetiti nekom problemu?

Na tržištu trenutno ne postoje prikladna softverska rješenja koja bi omogućavala ljudima jednostavniju pretragu za automehaničarskom radionicom, ali istovremeno pomogla mehaničarima da pronađu klijente, reagiraju na upit i lakše organiziraju radno vrijeme.

U Indiji postoji sustav GoMechanic koji je mreža različitih automehaničarskih radionica koji se mogu kontaktirati preko jedne aplikacije. GoMechanic nude brojne mogućnosti: preuzimanje vozila sa željene lokacije, popravak i povrat vozila na željenu lokaciju, nabavku dijelova za vozila…

GoMechanic sustav može se usporediti s hrvatskim sustavom HAK - Hrvatski autoklub. HAK također ima mobilnu aplikaciju Croatia Traffic Info-HAK, no iako ona ima opciju povezivanja pomoći na cesti uz slanje točne lokacije kontaktnom centru HAK-a, aplikacija više služi kao sredstvo bržeg informiranja o stanju na cestama, popisu autoklubova, informiranju o interesnim točkama poput restorana, hotela, ljekarni i dr., cijenama goriva, za pomoć kod plaćanja parkinga ili pronalaska parkiranog automobila. Aplikacija HAK-a je zasigurno korisnija od portala HAK-a i bolja opcija za vozače, no ne pruža jasan pregled automehaničarskih radionica u blizini, ne nudi dodatne informacije o njima niti omogućava izbor te je orijentirana isključivo na povezivanje s HAK brzom i hitnom pomoći na cesti.

Kako bi neko digitalno rješenje, odnosno mobilna aplikacija na tržištu opstala i dobila zadovoljne korisnike, ona mora biti korisna i mora biti *user-friendly*. To bi značilo:

- orijentirana je na potrebe svoje primarne publike,
- sigurnost je uvijek na prvom mjestu,
- ima izvrstan dizajn,
- prazan prostor mora biti dobro iskorišten,

- fontovi i grafika moraju biti optimalne veličine,
- omogućuje jednostavno snalaženje,
- *push* notifikacije se koriste mudro,
- aplikacija mora biti intuitivna,
- u aplikaciji treba smanjiti vrijeme učitavanja (eng. *loading time*),
- izbjegavati nepotrebna preusmjeravanja,
- razmotriti korištenje različitih platformi,
- pri izradi se ne smije zaboraviti na offline iskustvo,
- ne pretvoriti aplikaciju u znanstvenu fantastiku. [7]

2.3. Anketa o zadovoljstvu automehaničarskim radionicama

U istraživanju teme primijećeno je kako ne postoje kvalitetna istraživanja o zadovoljstvu odnosno stavu korisnika prema automehaničarima i automehaničarskim radionicama. Jedno od rijetkih istraživanja koje se može pronaći je istraživanje American Automobile Association (skraćeno AAA) iz 2016. godine koje je pokazalo da je većina Amerikanaca sumnjičava prema automehaničarskim radionicama te da u njih nemaju povjerenja. Također, jedna trećina vozača (što je u slučaju Amerike 75 milijuna ljudi) tek treba pronaći svojeg stalnog mehaničara. [8]

Za provjeru situacije u Hrvatskoj, napravljeno je malo istraživanje koje može pokazati generalan stav o automehaničarskim radionicama kod hrvatskih vozača.

2.3.1. Metodologija istraživanja

Cilj istraživanja bio je prikupiti podatke o stavu hrvatskih vozača prema automehaničarima te njihovim potrebama kada traže automehaničarske radionice.

S obzirom na to da je anketa "standardizirani postupak s pomoću kojeg se potiču, prikupljaju i analiziraju izjave odabranih ispitanika s namjerom da se dobije uvod u njihove stavove, mišljenja, preferencije, motive i oblike ponašanja." [9], odabrana je kao metoda ovog istraživanja. Anketa je napravljena i poslana ispitanicima preko alata Google Forme.

Ispitanici, odnosno uzorak istraživanja bili su vozači, točnije 49 osoba iznad 18 godina koje posjeduju vozačku dozvolu. S obzirom na to da je potreba za automehaničarima nešto što u mnogim slučajevima dođe nepredvidljivo i tiče se svih bez obzira na spol ili godine, nije napravljena dodatna segmentacija. Anketa je bila anonimna.

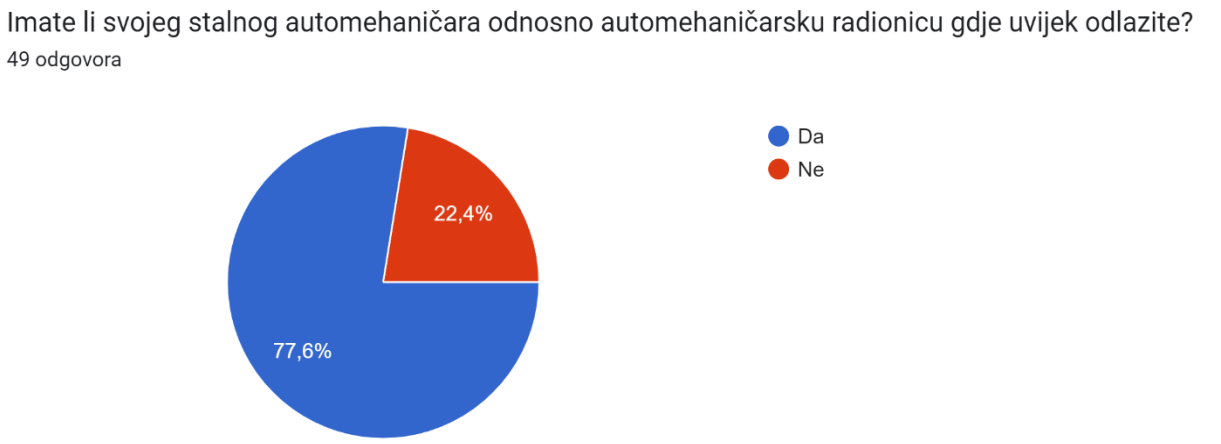
2.3.2. Istraživačka pitanja

Anketa se sastojala od 8 pitanja. 2 pitanja su bila pitanja u obliku linearne skale, a 6 pitanja su bila pitanja višestrukog odabira.

2.3.3. Analiza istraživanja

S obzirom na to da je anketa kvantitativna metoda, prikupljenim podacima dobiven je prikaz što se događa (npr. gdje osobe traže automehaničarske radionice), no prikupljeni su i brojevi koji pokazuju postotak, odnosno koliko ljudi ima određeni stav.

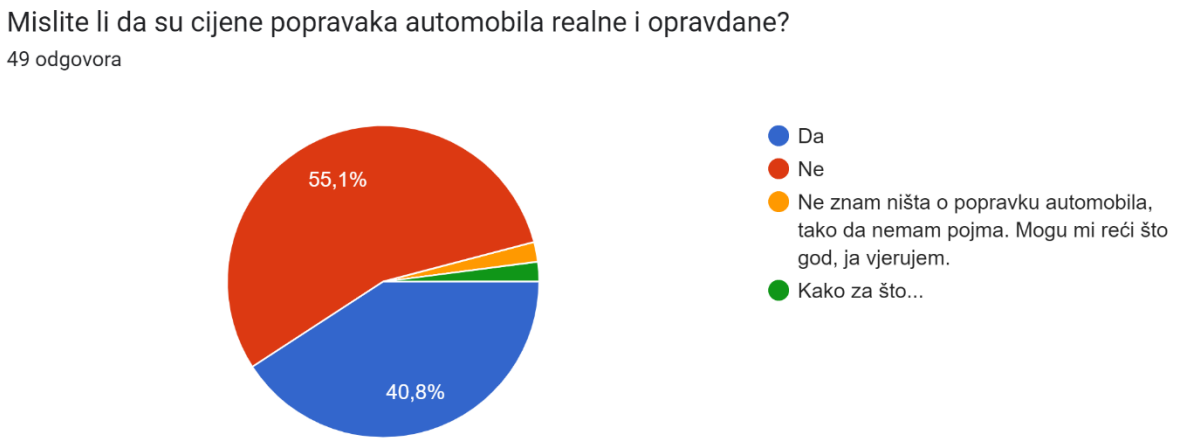
Prvo pitanje ankete glasilo je "Imate li svojeg stalnog automehaničara odnosno automehaničarsku radionicu gdje uvijek odlazite?". Više od tri četvrtine ispitanika potvrdno je odgovorilo, dok za jednu četvrtinu to znači kako svojeg automehaničara tek trebaju pronaći. To pokazuje kako je u Hrvatskoj situacija nešto malo bolja od Amerike i američkog, prethodno spomenutog, AAA istraživanja.



Slika 2.3.3.1 - Prvo pitanje ankete

Drugo pitanje ankete glasilo je: “Mislite li da su cijene popravaka automobila realne i opravdane?” Više od polovice ispitanika (55,1%) odgovorilo je ‘Ne’, 40,8% odgovorilo je ‘Da’ dok su dva odgovora bila proizvoljna:

1) Ne znam ništa o popravku automobila, tako da nemam pojma. Mogu mi reći što god, ja vjerujem.
2) Kako za što…



Slika 1.3.3.2 - Drugo pitanje ankete

Iz ovog pitanja vidljivo je da većina smatra kako cijene nisu realne, no s obzirom na to da se anketa provodi u vrijeme inflacije u Hrvatskoj kada cijene i energenata i tehnike, pa samim time i cijena rada automehaničara rastu, ove odgovore treba uzeti u obzir u širem kontekstu.

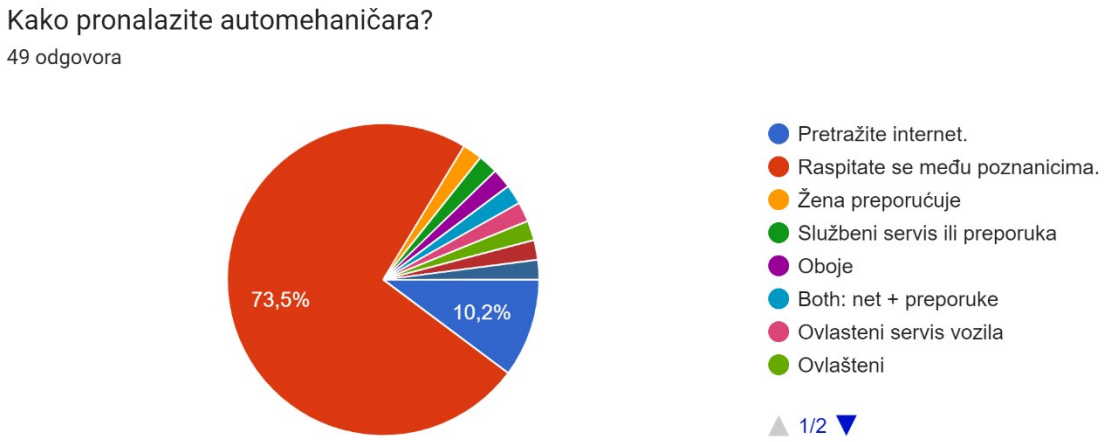
Treće pitanje ankete glasilo je: “Jeste li se ikada osjećali prevarenima nakon popravka automobila?”

Slika 2.3.3.2 - Treće pitanje ankete

Kroz odgovore je vidljivo kako je više od pola ispitanih imalo negativno iskustvo s automehaničarem. S obzirom na to da većina ispitanika cijene smatra nerealnim, možemo pretpostaviti da je to ujedno i jedan od razloga za ovakav postotak.

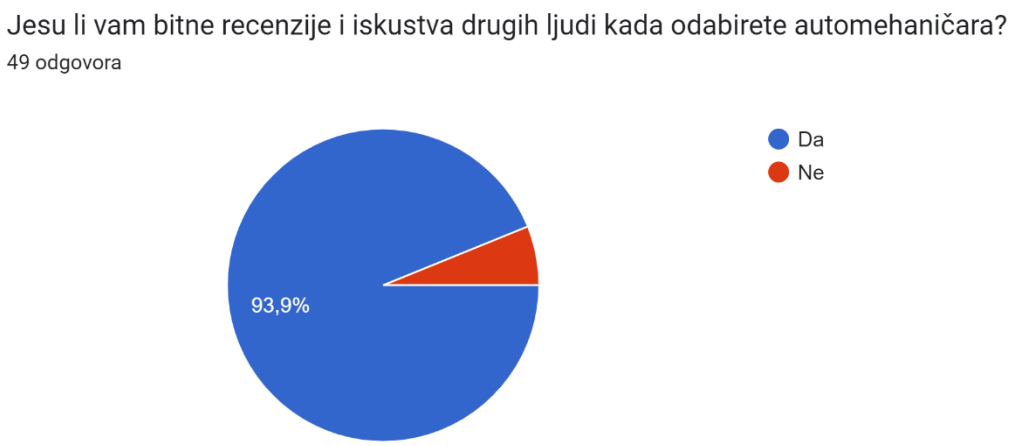
Četvrto pitanje ankete bilo je: “Kako pronalazite automehaničara?” Čak je 36 ispitanika odgovorilo da automehaničara pronalazi preko preporuke poznanika, što je potvrdilo tezu da ljudi još uvijek automehaničare pronalaze usmenom predajom i preporučivanjem. Sljedeći odgovor po zastupljenosti bio je ‘Pretražujete internet’, na što je odgovorilo 5 osoba. Troje ispitanika rekli su da automehaničara pronalaze u ovlaštenom servisu, dvoje se odlučilo za kombinaciju preporuke i Interneta. Ostatak ispitanika su ponudili specifične odgovore poput:

- “Žena preporučuje.”
- „Prijatelj nam je automehaničar“
- „Obiteljski majstor“



Slika 2.3.3.3 - Četvrto pitanje ankete

Peto pitanje bilo je vezano za recenzije. Glasilo je: “Jesu li vam bitne recenzije i iskustva drugih ljudi kada odabirete automehaničara?”. Čak je 93.9% ispitanih potvrdno odgovorilo. To je ujedno bio najveći postotak slaganja u cijeloj anketi.

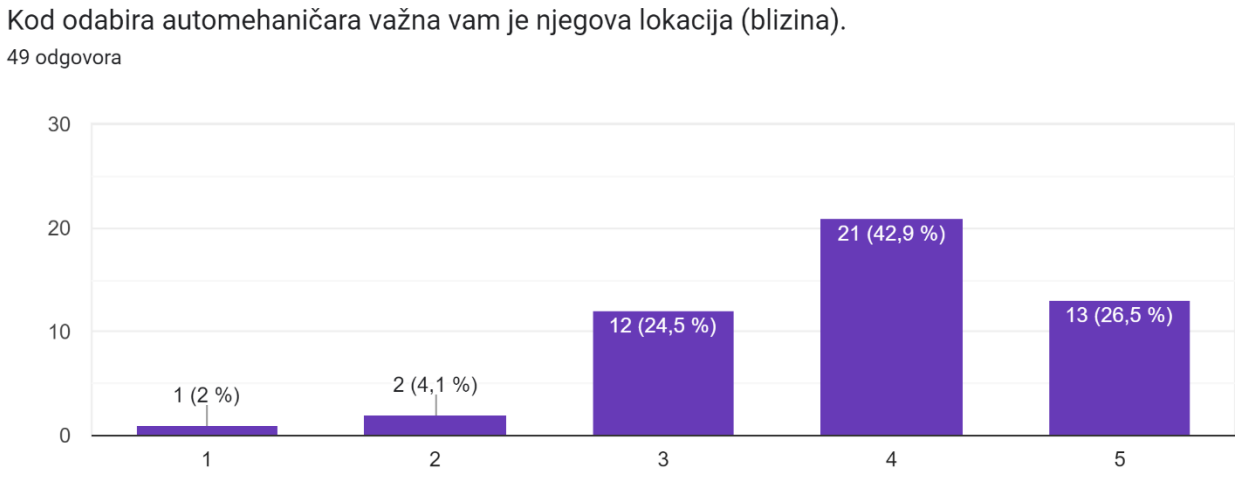


Slika 2.3.3.4 - Peto pitanje ankete

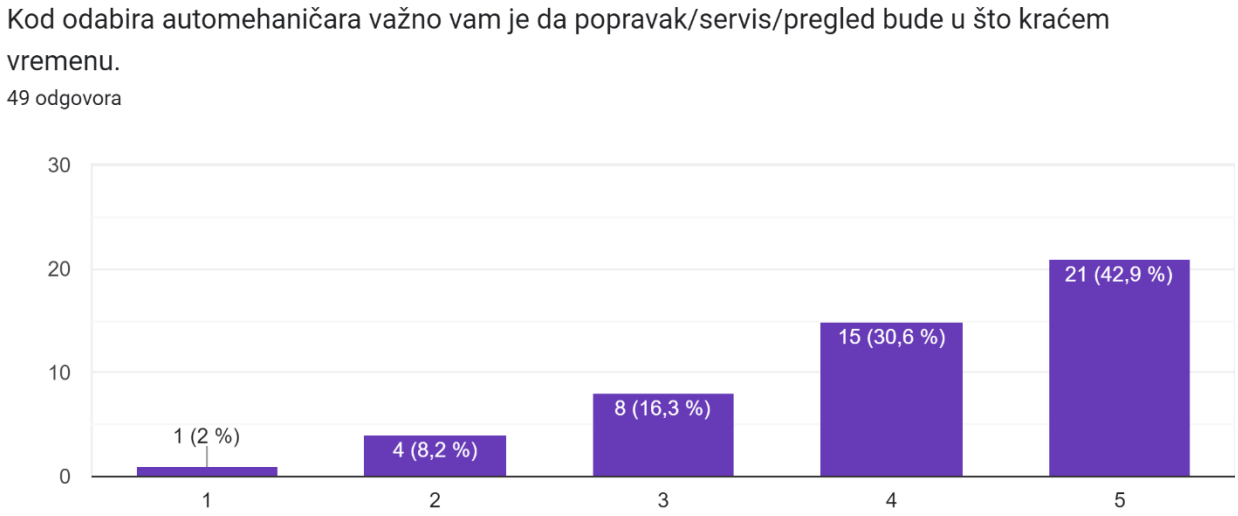
Šesto i sedmo pitanje sadržavalo je linearno mjerilo gdje je 1 značilo - Nije mi bitno, a 5 - Jako mi je bitno.

Šesto pitanje glasilo je: “Kod odabira automehaničara važna vam je njegova lokacija (blizina)”. Odgovori su pokazali kako većina smatra blizinu automehaničarske radionice bitnom, no sedmo pitanje: “Kod odabira automehaničara važno vam je da popravak/servis/pregled bude u što kraćem vremenu.” pokazalo je da je vrijeme popravka ispitanicima još važniji faktor od same lokacije.

Točni postoci odgovora šestog i sedmog pitanja prikazani su grafički na slikama.

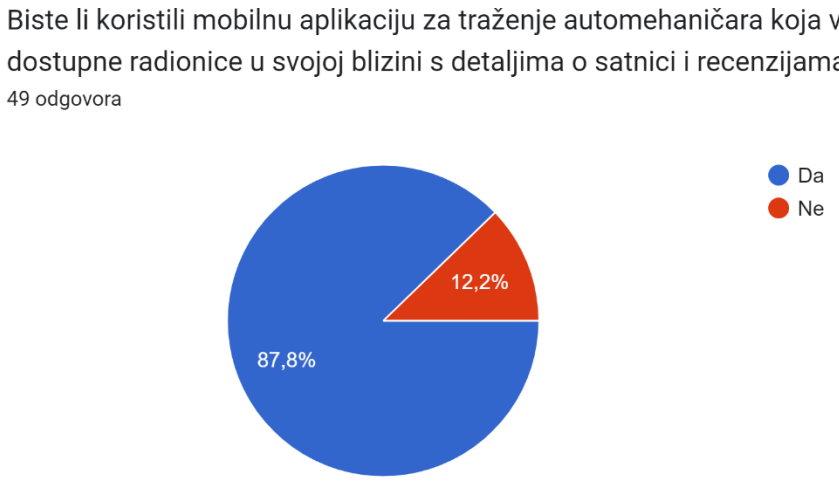


Slika 2.3.3.5 - Šesto pitanje ankete



Slika 2.3.3.6 - Sedmo pitanje ankete

Posljednje, osmo pitanje ankete bilo je direktno povezano s Car Medic aplikacijom. Glasilo je: “Biste li koristili mobilnu aplikaciju za traženje automehaničara koja vam omogućava da vidite sve dostupne radionice u svojoj blizini s detaljima o satnici i recenzijama?” i tako je provjerena potencijalna korisnost planiranog softverskog rješenja za pružanje usluga automehaničara. Čak je 43 ispitanih reklo kako bi aplikaciju koristilo, dok je 6 ispitanika odgovorilo s ‘Ne.’



Slika 2.3.3.7 - Osmo pitanje ankete

S obzirom na to da se rezultati ankete na temelju uzorka mogu generalizirati na širu društvenu skupinu, iako se radi o manjem uzorku, moguće je zaključiti da naših 49 ispitanika pokazuju stavove vozača Hrvatske.

Zaključno - iako većina ima stalnog mehaničara, povjerenje u struku je upitno. Najviše se cijeni brzina rada i popravka, no bitna je i lokacija automehaničarske radionice. Ispitanici cijene ne smatraju realnima, no zato kod odabira automehaničarske radionice gotovo svi paze na recenzije. Također, velika većina ispitanih vozača koristila bi Car Medic aplikaciju.

3. Arhitektura sustava

Polazište kod razvoja bilo koje aplikacije je dizajn arhitekture sustava i o njemu treba dobro razmisliti. Arhitektura se može shvatiti kao organizacija sustava te prilikom planiranja treba uzeti u obzir sve komponente sustava te promisliti kako će se one povezivati i u kojem će okruženju međusobno komunicirati.

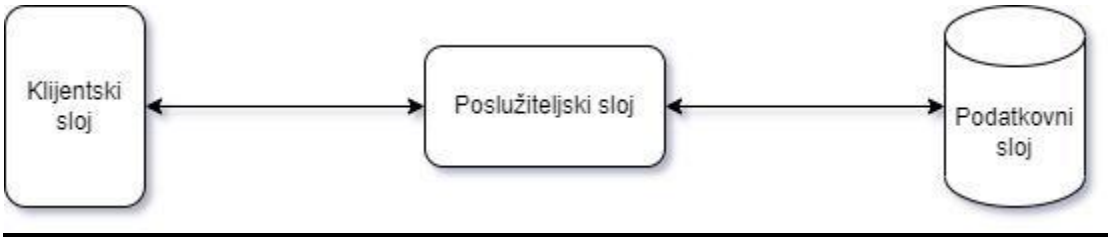
Jedna od najučestalijih i korištenih arhitektura kod izrade mobilnih aplikacija danas je troslojna arhitektura koja je zbog prednosti navedenih u nastavku odabrana i kao arhitektura za aplikaciju Car Medic.

Slojeve arhitekture sustava čine:

- prezentacijski odnosno klijentski sloj, shvaćen kao korisničko sučelje koje može biti statičko ili dinamičko,
- sloj poslovne logike čini jezgru aplikacije, upravlja funkcionalnostima aplikacije i dio je poslužiteljskog sloja,
- podatkovni sloj koji čini bazu podataka gdje se podaci pohranjuju i preuzimaju.

3.1. Skica i opis sustava

U troslojnoj arhitekturi podatkovni sloj pohranjuje podatke i odgovoran je za njihovu prezentaciju i postojanost, poslužiteljski odnosno logički sloj je srednji sloj koji koristi i priprema podatke, a prezentacijski sloj je glavni produkt, odnosno sučelje koje korisnik koristi.



Slika 3.1.1 - Skica arhitekture sustava

Najveća prednost troslojne arhitekture sustava je međusobna logička i fizička odvojenost funkcionalnosti. Tri različita sloja su u potpunosti neovisna i mogu raditi na zasebnim operativnim sustavima i poslužiteljima, ovisno o potrebama i funkcionalnostima, a možemo ih i zasebno optimizirati i mijenjati. Primjerice, možemo prilagoditi prezentacijski sloj, točnije korisničko sučelje ovisno o funkcionalnostima bez da moramo mijenjati poslužiteljski dio.

U suštini, najvažnije prednosti korištenja troslojne arhitekture sustava su:

1. međusobna neovisnost slojeva,
2. brži razvoj koji omogućuje neovisnost - zasebni timovi programera rade na pojedinačnim slojevima brže dolazeći do rješenja,
3. bolja skalabilnost - može se skalirati individualno prema sloju što omogućava potencijalne uštede,
4. veća pouzdanost - ako jedna komponenta, odnosno sloj ima grešku, manja je vjerojatnost da će to utjecati na sve druge performanse,
5. više sigurnosti - poslužiteljski sloj odnosno sloj poslovne logike funkcionira kao membrana između korisničkog sučelja i baze podataka, sprječavajući zloupotrebu. [10]

3.2. Korištene tehnologije, jezici i razvojna okruženja

U ovom poglavlju su detaljnije opisana tri sloja arhitekture sustava, a unutar njih opisane su korištene tehnologije, programski jezici i razvojna okruženja koja se koriste za izradu programskog rješenja za pružanje usluga automehaničara Car Medic.

3.2.1. Klijentski sloj

Klijentski, odnosno prezentacijski sloj arhitekture čini Android aplikacija pisana u programskom jeziku Kotlin. Iako istu aplikaciju koriste i automehaničari i klijenti, korisničko sučelje preko kojeg koriste funkcionalnosti aplikacije izgleda drugačije za obje skupine.

Android je izrazito moćan i najpopularniji operacijski sustav na svijetu. U studenom 2022. godine Android operacijski sustav zauzimao je 71,96% tržišta, dok je drugi na tržištu iOS sa 27,48%. [11] Gotovo kao da nemaju konkurencije, broj Android aplikacija se svakodnevno povećava te se Android trenutno nalazi unutar 2,5 milijardi aktivnih uređaja. Android je sveprisutna i besplatna platforma za korištenje, a pokriva širok spektar uređaja od mobilnih telefona preko tableta do televizora.

Operacijski sustav Android je 2003. godine za potrebe kamera krenula razvijati američka tvrtka Android Inc. koju je 2005. godine kupio Google Inc. Googleov tim programskih inženjera odlučuje bazirati Android na Linux jezgri (eng. *kernel*) i namjenjuje ga za mobilne telefone. Prva verzija Androida za komercijalnu upotrebu izašla je 2008. godine, a 2012. godine postaje najzastupljeniji operativni sustav za mobilne telefone na tržištu. [12]

Za razvoj mobilnog sučelja aplikacije Car Medic, korišteno je razvojno okruženje **Android Studio.** To je integrirano razvojno okruženje (eng. *Integrated Development Environment* – skraćeno IDE) namijenjeno za produktivniji razvoj Android aplikacija. Prva stabilna verzija Android Studia 1.0. izašla je 2014. godine. Odlična značajka mu je što se aplikacija, nakon što je razvijena kroz Android Studio, može objaviti u Google Play Storu.

Od 7. svibnja 2019. godine, Kotlin je zamijenio Javu i postao Googleov preferirani programski jezik za razvoj mobilnih aplikacija. [13] Car Medic razvijen je u Kotlinu.

Kotlin je programski jezik koji je razvila tvrtka JetBrains, poznata po stvaranju IntelliJ IDEA (razvojnog okruženja za razvoj Java aplikacija, na kojem se temelji i Android Studio). JetBrains programeri htjeli su napraviti statički tipizirani programski jezik, kompatibilan s Java kodom i koji je kvalitetan, ali istovremeno ga je jednostavno naučiti. Ime Kotlin dolazi od ruskog otoka pored Sankt Peterburga gdje je smještena većina razvojnog tima. [14]

Kotlin je besplatan programski jezik otvorenog koda koji je 100% interoperabilan s Java kodom, a od 2019. godine prvi izbor Googlea za razvoj Android mobilnih aplikacija. Kotlin ima brojne značajke koje omogućuju jednostavno korištenje, a Jemerov i Isakova ističu neke od njih:

- tipovi funkcija (eng. *function types*) - omogućavanje funkcijama da prime ili vraćaju druge funkcije kao parametre,
- Lambda izrazi (eng. *lambda expressions*) - omogućavanje pisanje izraza s manje šablonskog kôda (eng. *boilerplate code*),
- klase podataka (eng. *data classes*) - omogućuju sažetu sintaksu za stvaranje objekata nepromjenjive vrijednosti,
- bogat skup API-ja za rad s objektima.

Dodatne karakteristike Kotlina su pragmatičnost odnosno rješavanje stvarnih problema, konciznost koja omogućava jednostavnije snalaženje unutar koda bez potrebe za pisanjem ponavljajućih kodova i sigurnost.

Danas mnogi odabiru pisanje u Kotlinu zbog njegovih prednosti nad Javom. U svojem radu Šimec i Filipec ističu kako: “Aplikacija programirana u Kotlinu može u nekim slučajevima biti i do 40% kraća nego da je programirana u Javi.”[15]. Također ističu kako je Kotlinova sintaksa jednostavnija što dovodi do kraćeg koda i bržeg pisanja. Primjerice, nije potrebno stavljati točku-zarez na kraju reda, a *get*-eri i *set*-eri su automatski napravljeni i nisu vidljivi na kodu.

3.2.2. Poslužiteljski sloj

Poslužiteljski sloj zvan još i slojem poslovne logike čini web serverski dio koji spaja korisničko sučelje, točnije klijentski sloj, s bazom podataka u podatkovnom sloju sustava.

Kod stvaranja Car Medic programskog rješenja, korišten je razvojni okvir (eng. *framework*) **ASP.NET Web API 2** i on je pisan u C# programskom jeziku.

Aplikacijsko programsko sučelje (eng. *Application Programming Interface*, skraćeno API) je set definicija i protokola za izgradnju i integraciju aplikacijskog softvera. API dozvoljava proizvodu ili servisu da komunicira s drugim proizvodima i servisima bez znanja o tome kako su implementirani. [16] API je pojednostavljeni način kako spojiti infrastrukturu kroz razvoj aplikacije, ali također omogućava podjelu podataka s vanjskim korisnicima. Javni API predstavlja jedinstvenu poslovnu vrijednost jer može pojednostaviti i proširiti poslovanje kroz spajanje s partnerima, kao i potencijalno donijeti zaradu kroz naplatu dijeljenja podatka (Google Karte API je dobar primjer).

ASP.NET (eng. *Active Server Pages Network Enabled Technologies*) *Web* API je skraćenica za aktivne poslužiteljske stranice mrežno omogućene tehnologije web aplikacijsko programsko sučelje i to je jednostavan, no vrlo moćan Microsoftov razvojni okvir napravljen za samo HTTP, JSON web servise. [17]

ASP.NET se temelji na otvorenom kodu te se koristi za brzu izgradnju dinamičkih aplikacija na sloju poslužitelja.

Većina web poslužitelja, odnosno web API-ja temelji se na REST stilu arhitekture (eng. *Representational State Transfer*), a to je slučaj i s Car Medic web API-jem. REST možemo opisati kao set principa koji definiraju i uspostavljaju komunikaciju između klijenta i poslužitelja pomoću jednostavnih HTTP protokola. Sustavi koji poštuju REST ograničenja zovu se RESTful sistemi, a oni komuniciraju putem HTTP metoda.

Najpoznatije HTTP metode ili akcije su:

- POST - metoda koja stvara resurs,
- GET - metoda koja dohvaća resurs,
- PUT - metoda koja mijenja resurs,
- DELETE - metoda koja briše resurs.

Ove metode odgovaraju CRUD operacijama CREATE, RETRIEVE, UPDATE i DELETE. [18]

U poslužiteljskom sloju korišteno je objektno relacijsko mapiranje (eng. *Object Relational Mapping,* skraćeno ORM) što je tehnika koja stvara poveznicu između objektno orijentiranog programa i relacijske baze podataka. ORM služi kao sloj koji omogućava manipulaciju podacima u bazi koristeći objekte, a ne SQL upite. Za ORM upotrebljava se Entity okvir (eng. *framework*) s Database first pristupom. Entity Framework [19] pod potporom je Microsofta, a koristi se zbog jednostavnijeg rada s bazom podataka.

```
namespace WebAPI.Models.ORM
{
    using System;
    using System.Collections.Generic;

    public partial class Car
    {
        public Car()
        {
            this.Requests = new HashSet<Request>();
        }
        public int Id { get; set; }
        public System.DateTime DateCreated { get; set; }
        public System.DateTime DateModified { get; set; }
        public Nullable<System.DateTime> DateDeleted { get; set; }
        public bool Deleted { get; set; }
        public int UserId { get; set; }
        public string Manufacturer { get; set; }
        public string Model { get; set; }
        public int Year { get; set; }
        public decimal Odometer { get; set; }
        public string LicensePlate { get; set; }
        public virtual User User { get; set; }
        public virtual ICollection<Request> Requests { get; set; }
    }
}
```

Kôd 3.2.2.1 - Primjer generirane klase od Entity Frameworka

Postoje različiti načini mapiranja unutar Entity Frameworka, no za razvoj Car Medic aplikacije odabran je *Database first* pristup koji omogućuje kreiranje modela i klasa korištenjem postojeće baze podataka. [20]

3.2.3. Podatkovni sloj

Podatkovni sloj ponekad se naziva i razinom pristupa podacima. Među najkorištenijim sustavom za upravljanje bazom podataka (eng. *database management system*) je Microsoft SQL Server koji se koristi i u izradi ovog programskog rješenja.

Microsoft SQL Server koriste brojne tvrtke kojima je cilj pohrana te obrada i analiza podataka. Kao što samo ime govori, razvila ga je tvrtka Microsoft 1989. godine, a od tada gotovo na godišnje izlaze nove verzije. Posljednja verzija je izašla je 2022. godine, no kako se radi o novoj verziji, većina i dalje koristi verziju iz 2019. godine zbog produktivnosti koju nude njezine funkcionalnosti. Petrovečki i Kovačević u analizi SQL Servera 2019 zaključuju: "Rad s velikim podatkovnim klasterima unutar SQL Servera 2019 znatno olakšavaju rad s velikim količinama podataka te rad s visokovrijednim relacijskim bazama podataka na jednoj vrlo stabilnoj platformi." [21]

Microsoft SQL Servera koristi T-SQL (*Transact-SQL*) upitni jezik, koji se sastoji od tri primarne komponente:

- sloj protokola (eng. *protocol layer*) - omogućuje da klijent i poslužitelj koriste zajedničku memoriju iako su instalirani na odvojivim uređajima, prevodi upite u oblik koji relacijska baza može obraditi,
- procesor upita ili relacijski mehanizam (eng. *query processor*, poznat još kao eng. *relational engine*) - određuje izvršavanje upita i odlučuje što s njima napraviti, odnosno traži podatke iz mehanizma za pohranu i obrađuje podatke koji se vraćaju korisniku. Procesor upita omogućuje obradu upita te sadrži brojne funkcije i komponente, no tri glavne komponente su: parser upita (eng. *query parser*), optimizator upita (eng. *query optimizer*) i izvršitelj upita (eng. *query executor*). Parser je prva komponenta koji prima upit, stoga se prvenstveno bavi uklanjanjem sintaktičkih grešaka i generiranjem stabla upita (eng. *query tree*). Optimizator upita se pobrine da upit bude što učinkovitiji, dok izvršitelj upita izvrši upit i vrati rezultat.
- mehanizam za pohranu (eng. *storage engine*) - upravlja pohranom i dohvaćanjem stvarnih podataka kada korisnik to zatraži. [22]

Prednosti SQL Servera su brojne, no među najbitnijima se spominju:

- jednostavnost instalacije i korištenja uz dobre upute čine pozitivno korisničko iskustvo,
- zbog mogućnosti kompresije podataka i enkripcije nudi sigurno upravljanje poslovnim podacima,
- mogućnost odabira izdanja SQL Servera koje zadovoljava potrebe tvrtke odnosno klijenta (*enterprise* za korporacije, standardni, *express* za niske potrebe i *developer* za razvojne potrebe),
- visoka razina sigurnosti, mogućnost vraćanja izgubljenih ili oštećenih podataka.

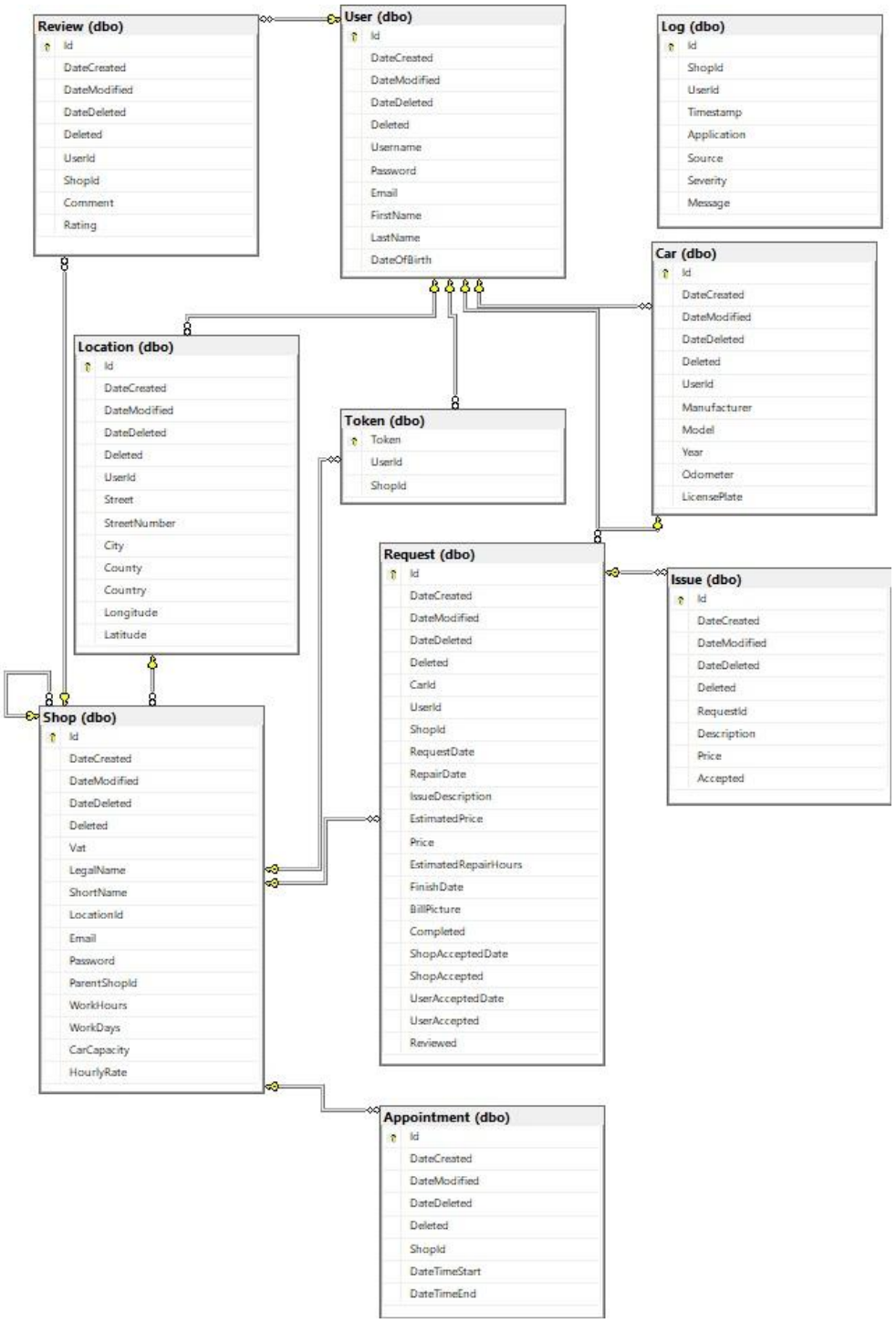
4. Implementacija sustava

U ovom poglavlju opisan je razvoj i implementacija komponenata sustava koje su spomenute u prethodnom dijelu. Prvo je napravljena baza podataka, potom Web API te nakon toga Android aplikacija sa zasebnim korisničkim sučeljem za automehaničare i klijente.

4.1. Razvoj komponenata

Kako bi neko programsko rješenje bilo cjelovito i uspješno napravljeno, prvo je potrebno osmisliti i isplanirati arhitekturu sustava, a zatim prema njoj implementirali komponente. S obzirom na to da je arhitektura Car Medic programskog rješenja troslojna, ovo potpoglavlje podijeljeno je na tri dijela. Prvi dio opisuje izradu i konfiguraciju baze podataka kroz Microsoft SQL Server. Drugi dio prikazuje izradu serverske aplikacije kroz ASP.NET Web API razvojni okvir i treći dio prikazuje izradu korisničkog sučelja Android aplikacije kroz Kotlin programski jezik.

4.1.1. Izrada i konfiguracija baze podataka



Slika 4.1.1.1 - Dijagram baze podataka

Na slici 10 je prikazan dijagram relacije tablica u bazi podataka i sadrži deset entiteta: *Shop, User, Token, Location, Review, Appointment, Request, Issue, Car* i *Log.*

Svaka tablica ima pet istih stupaca koji služe kao identifikatori i meta podaci za taj red koji su detaljnije objašnjeni u tablici 1.

Naziv stupca	Značenje stupca	Vrsta podatka
Id	Jedinstveni identifikator reda, automatski generiran od baze podataka	int
DateCreated	Vremenska oznaka kreiranja(stvaranja) resursa	datetime2
DateModified	Vremenska oznaka zadnje promjene resursa	datetime2
DateDeleted	Vremenska oznaka brisanja resursa	datetime2
Deleted	Zastavica koja označava je li resurs izbrisan	bit

Tablica 4.1.1.1 - Opis meta podataka svake tablice

Tablica *User* predstavlja klijenta u sustavu koji traži automehaničara. U ovu tablicu se spremaju podaci o korisniku koju su potrebni za rad sustava: e-mail, korisničko ime, ime i prezime osobe i datuma rođenja.

```
CREATE TABLE [dbo].[User](
      [Id] [int] PRIMARY KEY IDENTITY NOT NULL,
      [DateCreated] [datetime2](7) NOT NULL,
      [DateModified] [datetime2](7) NOT NULL,
      [DateDeleted] [datetime2](7) NULL,
      [Deleted] [bit] NOT NULL,
      [Username] [nvarchar](50) NOT NULL,
      [Password] [nvarchar](100) NOT NULL,
      [Email] [nvarchar](50) NOT NULL,
      [FirstName] [nvarchar](50) NOT NULL,
      [LastName] [nvarchar](50) NOT NULL,
      [DateOfBirth] [date] NOT NULL,
 )
```

Kôd 4.1.1.1 - Primjer kôda za izradu tablice klijenta

Tablica *Shop* predstavlja automehaničara i u nju se spremaju podaci o automehaničarskoj radionici koja posluje.

U tablicu *Token* se spremaju token vrijednosti za svakog prijavljenog korisnika koje se provjeravaju na serverskoj aplikaciji kao dio autentikacije i autorizacije. Autentikacija je proces provjere identiteta korisnika ili servisa, a autorizacija je proces utvrđivanja prava pristupa korisnika ili servisa.

Request tablica služi za provjeru zahtjeva o servisu koje klijent stvori te predstavlja glavnu funkcionalnost sustava. Tu se spremaju vrijednosti o datumu kada je zahtjev zaprimljen, kada je servis gotov, cijena, opis problema, datum i vrijeme popravka i slično. Svaki zahtjev ima vezu jedan-više prema tablici *Issue* koji služi za spremanje podataka o pojedinačnom problemu koji je pronađen za vrijeme servisa.

Review tablica sprema recenzije koje su klijenti ostavili nakon odrađenog servisa. Tablica sprema podatke o ocjeni, recenziji, korisniku koji je ostavio recenziju i automehaničaru na koga se odnosi ta recenzija.

Log tablica služi kao pomoćna tablica za spremanje podataka o greškama koje su se dogodile kroz aplikaciju kako bi se greške mogle lakše pronaći i ukloniti u sustavu.

4.1.2. Izrada serverske aplikacije

Serverska aplikacija je izrađena kao sučelje za programiranje aplikacije za prijenos reprezentativnog stanja (eng. *representational state transfer application programming interface*, *REST API*) te koristi kontrolere kako bi obradila podatke za svaki entitet zasebno.

Za autentikaciju korisnika koristi se osnovna autentikacija (eng. *Basic Authentication*) u obliku atributa koji se stavlja na svaki kontroler koji želi zaštititi svoje resurse.

```
public async Task AuthenticateAsync(
       HttpAuthenticationContext context,
       CancellationToken cancellationToken)
        {
            var request = context.Request;
            var authorization =  request.Headers.Authorization;
            if ( context.ActionContext.ActionDescriptor
             .GetCustomAttributes<AllowAnonymousAttribute>().Any()
            )
            { return; }
            if ( authorization == null ||
                 authorization.Scheme != "Basic" ||
                 !request.Headers.Contains("Authorization-For"))
            {
                context.ErrorResult = new
                   AuthenticationFailureResult(
                     new ForbiddenException(), request);
                return;                                    }
            try
            {
                var authFor = request.Headers.First(
                    x => x.Key == "Authorization-For"
                );
                if (authFor.Value.Contains("User"))
                {
                    var identity = new GenericIdentity(
                     ExtractUserFromCredentials(
                         authorization.Parameter
                       ).Username,
                     "User"
                    );
                    context.Principal =
                        new GenericPrincipal(identity, null);
                }
                else if (authFor.Value.Contains("Shop"))
                {
                    var identity = new GenericIdentity(
                       ExtractShopFromCredentials(
                           authorization.Parameter
                       ).Email,
                    "Shop"
                    );
                    context.Principal = new GenericPrincipal(
                        identity, null
                    );
                }
                else
                {
```

```
                context.ErrorResult = new
                   AuthenticationFailureResult(
                       new ForbiddenException(), request
                   );
                return;
            }
        }
        catch (Exception e)
        {
            context.ErrorResult = new
               AuthenticationFailureResult(e, request);
        }
    }
```

Kôd 4.1.2.1 - Autentikacijski atribut

U zaglavlju (eng. *Header*) svakog zahtjeva (eng. *Request*) mora se staviti vrijednost korisničkog imena i lozinke ili vrijednost tokena jer se koristi osnovna autentikacija, Prilikom autentikacije korisnika provjerava se je li poslano korisničko ime i lozinka ili vrijednost „token“ i vrijednost tokena. Ako je poslano korisničko ime i lozinka, sustav prijavljuje korisnika, što znači da briše stari token i generira novi. No ako je poslan token, sustav pretražuje tablicu *Token* u bazi i pokušava tako prijaviti korisnika. Ako sustav pronađe token, dopušta korisniku pristup zaštićenim resursima. Ako ne pronađe token, vraća odgovor s HTTP šifrom 403 i brani pristup zaštićenim resursima.

S obzirom na to da svaki kontroler treba imati pristup autoriziranom korisniku, pristup instanci konteksta baze podataka (eng. *Database Context*) i prilagođenim metodama odgovora (eng. *Response*), napravljena je apstraktna klasa *BaseController* koju svaki kontroler nasljeđuje, čime se rješava problem konzistentnosti, kao što je prikazano u isječku kôda 4.

```
[BasicAuthentication]
 public abstract class BaseController : ApiController
 {
     internal readonly string _controllerName;
     internal readonly DbEntities _db;
     internal User _user;
     internal Shop _shop;
     public User AuthUser
     {
         get
         {
             if (_user != null)
```

```
        {
            return _user;
        }
        if (_user == null &&
          User?.Identity?.AuthenticationType == "User" &&
          User?.Identity?.Name != null
        )
        {
            _user = _db.Users.Where(
               u => u.Username == User.Identity.Name
            ).FirstOrDefault();
            return _user;
        }
        return null;
    }
}

public Shop AuthShop
{
    get
    {
        if (_shop != null)
        {
            return _shop;
        }
        if (_shop == null &&
          User?.Identity?.AuthenticationType == "Shop" &&
          User?.Identity?.Name != null
        )
        {
            _shop = _db.Shops.Where(
                u => u.Email == User.Identity.Name
            ).FirstOrDefault();
            return _shop;
        }
        return null;
    }
}

public DbEntities Db { get { return _db;} }

public BaseController(string controllerName)
{
    _controllerName = controllerName;
    _db = new DbEntities();
}
```

Kôd 4.1.2.2 - Apstraktna klasa *BaseController*

Zbog sigurnosti i izbjegavanja povlačenja previše podataka iz baze, te slanja istih u Android aplikaciju, napravljeni su zasebni objekti za prijenos podataka (eng. *Data Transfer Object,* skraćeno DTO) na koje se mapiraju podaci iz entiteta. Njihova je zadaća samo prijenos podataka (kôd 5).

```
namespace WebAPI.Models.DTOs
{
    [JsonObject]
    public class AppointmentDTO
    {
        [JsonProperty]
        public int Id { get; set; }
        [JsonProperty]
        public DateTime DateCreated { get; set; }
        [JsonProperty]
        public DateTime DateModified { get; set; }
        [JsonProperty]
        public DateTime? DateDeleted { get; set; }
        [JsonProperty]
        public bool Deleted { get; set; }
        [JsonProperty]
        public int ShopId { get; set; }
        [JsonProperty]
        public DateTime DateTimeStart { get; set; }
        [JsonProperty]
        public DateTime DateTimeEnd { get; set; }
        [JsonProperty]
        public ShopDTO Shop { get; set; }
    }
}
```

Kôd 4.1.2.3 - Primjer objekta za prijenos podatka za klasu *Appointment*

Kako bi se olakšalo mapiranje iz entiteta u objekt za prijenos podataka, napravljena je djelomična klasa (eng. *Partial class*) za svaki entitet stvoren od Entity Frameworka koji mu daje mogućnost laganog mapiranja u objekt za prijenos podataka, ovisno koliko se duboko želi ući u entitet i povući reference na druge tablice (primjer u kôdu 6).

Djelomične klase su jedinstvena značajka C# programskog jezika. One omogućavaju dijeljenje funkcionalnosti jedne klase u više datoteka te se prilikom prevođenja koda sve spaja u jednu klasu, u jednoj datoteci.

```
namespace WebAPI.Models.ORM
{
    partial class Appointment
    {
        public static AppointmentDTO ToDTO(
           Appointment item,
           int level
        )
        {
            if (item == null || level <= 0)
            {
                return null;
            }
            return new AppointmentDTO
            {
                DateTimeStart = item.DateTimeStart,
                DateTimeEnd = item.DateTimeEnd,
                Shop = Shop.ToDTO(item.Shop, level - 1),
                DateCreated = item.DateCreated,
                DateDeleted = item.DateDeleted,
                DateModified = item.DateModified,
                Deleted = item.Deleted,
                Id = item.Id,
                ShopId = item.ShopId
            };
        }

        public static ICollection<AppointmentDTO> ToListDTO(
          ICollection<Appointment> list,
          int level
        )
        {
            if (list == null || level <= 0)
            {
                return null;
            }
            return list.Select(x => x.ToDTO(level)).ToList();
        }

        public AppointmentDTO ToDTO(int level)
        {
            return level > 0 ? new AppointmentDTO
            {
                DateTimeStart = DateTimeStart,
                DateTimeEnd = DateTimeEnd,
                Shop = Shop.ToDTO(Shop, level - 1),
                DateCreated = DateCreated,
                DateDeleted = DateDeleted,
```

```
                    DateModified = DateModified,
                    Deleted = Deleted,
                    Id = Id,
                    ShopId = ShopId
                } : null;
            }
        }
    }
```

Kôd 4.1.2.4 - Primjer *Partial* klase za entitet *Appointment*

Ovakav pristup nam omogućava da jednostavno mapiramo sirove podatke iz entiteta unutar jedne linije kôda i šaljemo ih na van. Unutar aplikacije su definirane prilagođene klase za odgovore, kao što je prikazano u tablici 2.

Naziv	**Funkcionalnost**
BaseResponse	Apstraktna klasa koju nasljeđuju svi odgovori. Sadrži poruku, *boolean* vrijednost je li uspješno, statusni kôd odgovora i vremensku oznaku kada je stvoren.
SingleResponse	Implementacija bazne klase koja vraća jedan objekt unutar *data* polja..
ListResponse	Implementacija bazne klase koja vraća više objekata unutar *data* polja.
PaginatedListResponse	Implementacija *ListResponse* klase koja vraća podatke za paginaciju.
ErrorResponse	Implementacija bazne klase koja vraća kôd greške koja se dogodila.

Tablica 4.1.2.1 - Tablica definicije odgovora serverske aplikacije

4.1.3. Izrada Android aplikacije

U ovom poglavlju detaljnije se objašnjavaju ključni dijelovi Car Medic Android aplikacije i zašto je odabran ovakav pristup.

Car Medic Android aplikacija zasniva se na aktivnostima (eng. *Activity*). Aktivnost je instanca klase *Activity* u Androidovom kompletu za razvoj softvera (eng. *Software Development Kit, SDK*) i ona je odgovorna za upravljanje korisničkim sučeljem.[23]

Prvo od čega se kreće je stvaranje apstraktnih baznih klasa. Sve funkcionalnosti koje se ponavljaju unutar određenih dijelova aplikacije dignute na apstraktnu razinu jer nije dobra praksa kopirati ponavljajući kôd (kôd 7).

```
abstract class BaseActivity : AppCompatActivity() {
    val apiService = ApiService.instance()
    val coroutineScope: CoroutineScope = CoroutineScope(Job())
    lateinit var mainHandler:Handler
    abstract fun initializeComponents()
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        appSettings()
        ContextInstance.initialize(this)
        mainHandler = Handler(applicationContext.mainLooper)
        if(!Helper.hasLocationPermissions()){
            Helper.getLocationPermission()
        }
    }
    @SuppressLint("SourceLockedOrientationActivity")
    private fun appSettings(){
        AppCompatDelegate.setDefaultNightMode(
            AppCompatDelegate.MODE_NIGHT_YES
        )
        requestedOrientation=
            ActivityInfo.SCREEN_ORIENTATION_PORTRAIT
    }
    open fun handleApiResponseError(response: ErrorResponse){
        Helper.showLongToast(this,response.Message!!)
    }
    open fun handleApiResponseException(call: Call, e: Exception)
    {
        if(e.message?.contains("Failed to connect to") == true){
            Helper.showLongToast(
               this,
```

```
                    getString(R.string.internet_connection_error))
            }
            else {
                Helper.showLongToast(this, e.message.toString())
            }
            call.cancel()
        }
        protected fun <T : View> showComponent(component: T) {
            component.visibility = View.VISIBLE
        }
        protected fun  <T : View> hideComponent(component: T) {
            component.visibility = View.GONE
        }
    }
```

Kôd 4.1.3.1 - Stvaranje apstraktne klase *BaseActivity*

Takav pristup omogućava spremanje velike količine logike van same aktivnosti, kako bi se smanjila zatrpanost iste. Da bi se dobila modularnost i određena razina generičnosti, napravljena je klasa *ActivityEnum* koja služi kao napredni enumerator kod inicijalizacije aktivnosti koja treba koristiti provjeru je li korisnik prijavljen i koja koristi navigacijsku traku na dnu. Takav pristup omogućava lako definiranje posebnosti svake aktivnosti, poput stila koji se koristi, koje vrste je navigacijska traka i slično.

Sama provjera korisnika se radi kroz zajedničke postavke. U knjizi *Android Programming- The Big Nerd Ranch Guide* daje se sljedeća definicija: „Zajedničke postavke (eng. *SharedPreferences*) su datoteke u sustavu datoteka koji se čitaju i mijenjaju koristeći *SharedPreferences* klasu. Instanca klase *SharedPreferences* ponaša se kao pohrana ključeva i vrijednosti.“ [23] Na tom se mjestu provjerava nalazi li se token vrijednost u postavkama te na koji račun pripada ta vrijednost, klijent ili automehaničar. Ako se token ne nalazi u zajedničkim postavkama, korisnik će biti prebačen na ekran za prijavu i bit će mu zabranjen pristup zaštićenim dijelovima aplikacije. Ovakav pristup je odabran da se aplikaciji može pristupiti čak i kada korisnik nema interneta.

Za komunikaciju Car Medic Android aplikacije i serverske aplikacije koristi se klasa *ApiService* koja je jedinstveni (eng. *Singleton*) objekt. *Singleton* je objekt koji se samo jednom instancira u životnom ciklusu aplikacije te svaki put kada se želi dobiti *singleton* objekt dobiva se referenca na postojeći objekt, ako postoji (kôd 8).

Iako je *Singleton* dostupan na globalnoj razini aplikacije i štiti objekt od vanjskih promjena jer samo *Singleton* objekt može instancirati samog sebe. Njegova upotreba ima i nekoliko mana. Neke od njih su teže jedinično (eng. *unit*) testiranje i to što *Singleton* nije dugoročno rješenje za pohranu jer se uništava kada Android potražuje natrag memoriju.

```
class ApiService private constructor() {
    companion object {
        private var apiService: ApiService? = null

        fun instance(): ApiService {
            if (apiService == null) {
                apiService = ApiService()
            }
            return apiService!!
        }
    }
...
}
```

Kôd 4.1.3.2 - *Singleton* implementacija klase *ApiService*

Unutar *ApiService* klase definirane su četiri metode za dohvat podataka preko HTTP-a, a to su: GET (kôd 9), POST (kôd 10), PUT (kôd 11), DELETE (kôd 12).

```
private fun get(
   url: String, usernamePassword:String?=null):Call
{
    val request: Request = Request.Builder()
       .url(url)
       .addHeader("Authorization",
            if(usernamePassword.isNullOrEmpty())
               getTokenHeader()
            else
              "Basic ${
                  Base64.encodeToString(
                     usernamePassword.encodeToByteArray(),
                     Base64.NO_WRAP
                  )}" )
      .addHeader("Authorization-For", Helper.AUTH_FOR_KEY!!)
      .build()
    return client.newCall(request)
}
```

Kôd 4.1.3.3 - Implementacija *GET* zahtjeva unutar klase *ApiService*

```
private inline fun <reified T> post(
   url: String,
   value: T,
   addAuth: Boolean = true): Call
{
    val json = Helper.serializeData(value)
    val body: RequestBody = json.toRequestBody(headers)
    val request = Request.Builder().url(url).post(body)
    if (addAuth) {
        request.addHeader(
            "Authorization", getTokenHeader())
        .addHeader(
            "Authorization-For", Helper.AUTH_FOR_KEY!!
        )
    }
    return client.newCall(request.build())
}
```

Kôd 4.1.3.4 - Implementacija *POST* zahtjeva unutar klase *ApiService*

```
private inline fun <reified T> put(
   url: String,
   value: T,
   addAuth: Boolean = true): Call
{
    val json = Helper.serializeData(value)
    val body: RequestBody = json.toRequestBody(headers)
    val request = Request.Builder().url(url).put(body)
    if (addAuth) {
        request.addHeader(
            "Authorization", getTokenHeader())
        .addHeader(
            "Authorization-For", Helper.AUTH_FOR_KEY!!
        )
    }
    return client.newCall(request.build())
}
```

Kôd 4.1.3.5 - Implementacija *PUT* zahtjeva unutar klase *ApiService*

```
private fun delete(url: String): Call {
    val request: Request = Request.Builder()
      .url(url)
      .addHeader("Authorization", getTokenHeader())
```

```
            .addHeader(
                "Authorization-For", Helper.AUTH_FOR_KEY!!
            ).delete().build()
        return client.newCall(request)
    }
```

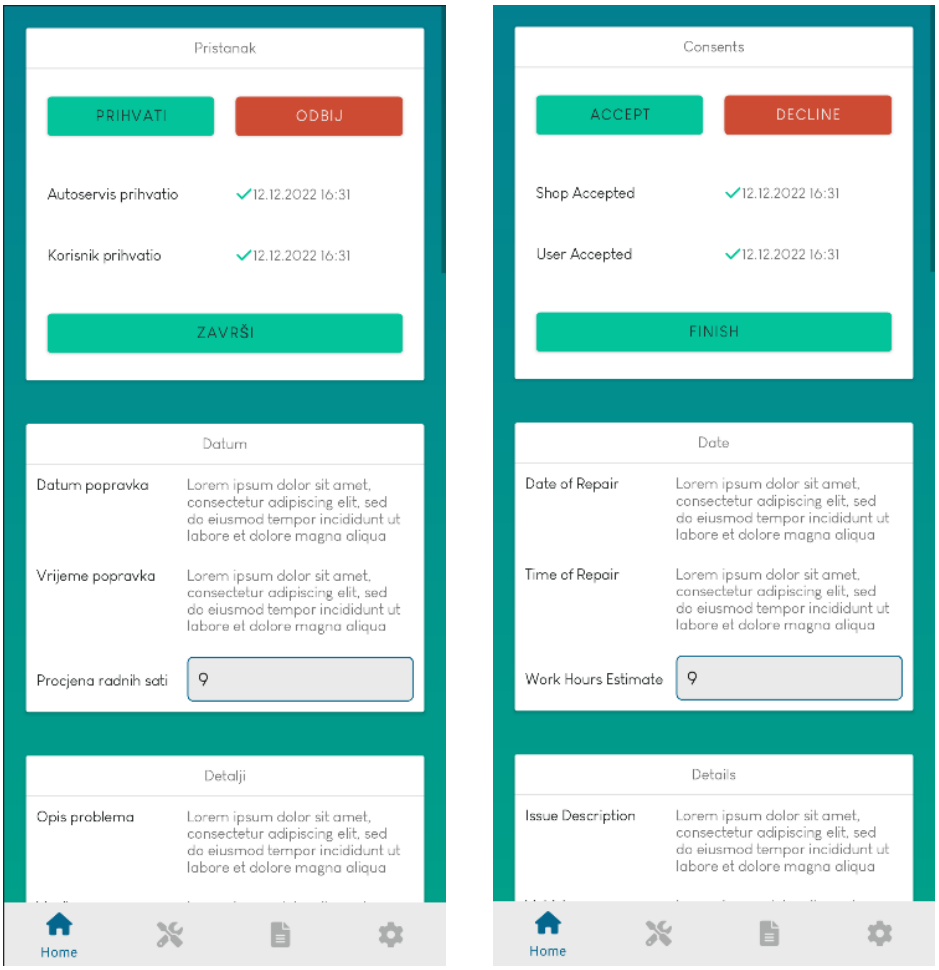
Kôd 4.1.3.6 - Implementacija *DELETE* zahtjeva unutar klase *ApiService*

Ovaj pristup omogućava generičko slanje podataka. Servis ne mora znati koja je vrsta podatka, samo se mora moći serijalizirati. Serijalizacija označava proces pretvaranja objekta u tok bajtova za pohranjivanje objekta ili prijenos u memoriju, bazu podataka ili datoteku.[24] Svaka implementacija zahtjeva, osim *DELETE*, ima mogućnost koristiti autorizaciju ili ne. Kako bi se izbjegle greške kod pisanja putanje, koristi se enumerator *ApiRoutes* koji sadržava sve bazne rute za pojedinačni kontroler na serverskoj aplikaciji. To znači da se potrebni parametri, kao u primjeru kôda 13, mogu lako dodati kako bi se napravio poseban upit na server.

```
fun retrieveCar(carId: Int, expanded: Boolean = false): Call {
    return get("${ApiRoutes.CAR}/${carId}${if (expanded)
"?expanded=true" else ""}")
}

fun retrieveCars(): Call {
    return get(ApiRoutes.CAR)
}

fun deleteCar(carId: Int): Call {
    return delete("${ApiRoutes.CAR}/${carId}")
}

fun createCar(newCar: Car): Call {
    return post(ApiRoutes.CAR, newCar)
}

fun updateCar(updatedCar: Car): Call {
    return put("${ApiRoutes.CAR}/${updatedCar.Id}", updatedCar)
}
```

Kôd 4.1.3.7 - Primjer implementacije *CRUD* funkcionalnosti za vozilo

Svi znakovni nizovi (eng. *strings*) korišteni unutar Car Medic aplikacije su unutar datoteke *strings* koje je pisana proširivim jezikom za označavanje (eng. *extensible markup language*, XML) što omogućava višejezičnost sustava. Sustav trenutačno podržava engleski i hrvatski jezik (slika 11).



Slika 4.1.3.1 - Primjer višejezičnosti aplikacije- lijevo hrvatski, a desno engleski

5. Opis gotovih funkcionalnosti

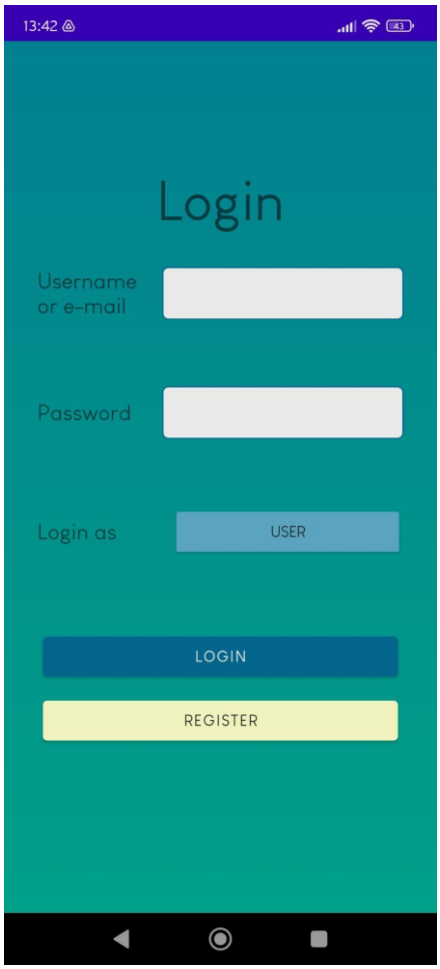
U sljedećim poglavljima opisuju se glavne značajke aplikacije. Za klijentsku aplikaciju to su:

- prijava,
- registracija,
- pretraga automehaničara,
- izrada zahtjeva za popravkom,
- upravljanje zahtjevima,
- praćenje aktivnog popravka,
- recenzija,
- postavke.

A za automehaničarsku aplikaciju su:

- prijava,
- registracija,
- praćenje aktivnog servisa,
- upravljanje servisima,
- postavke računa.

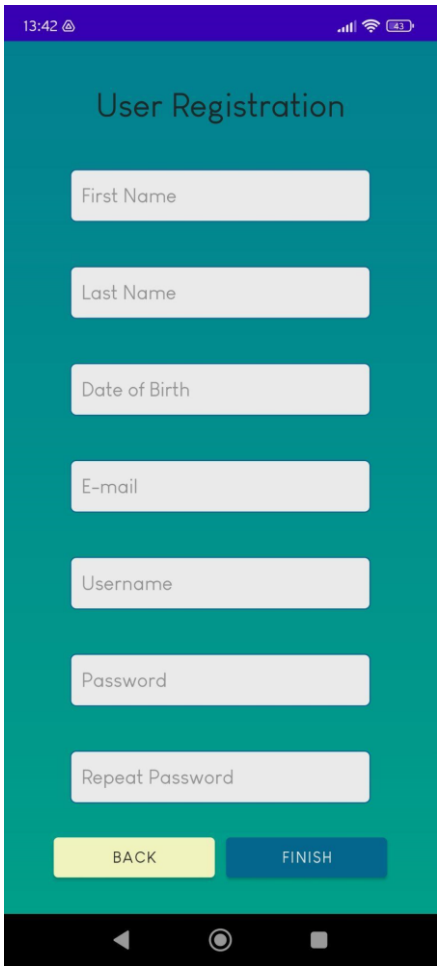
S obzirom na to da je ekran za prijavu automehaničara i klijenta isti, bit će samo jednom opisan. Prilikom navigacije u aplikaciju, ako korisnik nije prethodno prijavljen, bilo kao automehaničar ili klijent, bit će preusmjeren na ekran za prijavu, prikazan na slici 12. Prilikom prijave, korisnik treba unijeti korisničko ime ili e-poštu te lozinku i odabrati vrstu prijave, želi li se prijaviti kao automehaničar ili klijent.



Slika 5.1 - Ekran za prijavu

5.1. Aplikacija za klijenta

Kako bi pristupio aplikaciji, klijent se odmah na početku mora registrirati. To znači da korisnik navigira do ekrana za registraciju i tamo ispunjava sljedeće podatke: ime, prezime, datum rođenja, e-poštu, korisničko ime i lozinku, kako je prikazano na slici 13.



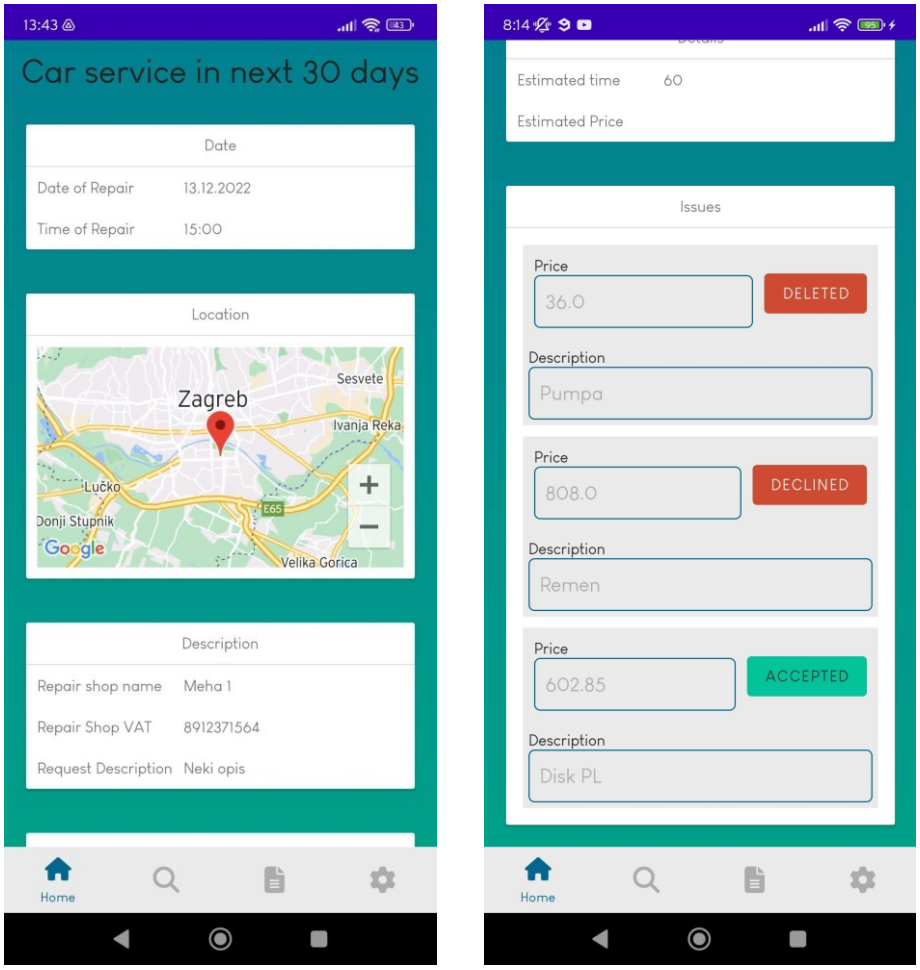
Slika 5.1.1 - Ekran za registraciju klijenta

Po uspješnoj registraciji, korisnik će biti prebačen na ekran za prijavu gdje treba upisati svoje korisničko ime i lozinku kako bi se prijavio u aplikaciju.

Po uspješnoj prijavi, klijent će biti preusmjeren na početni ekran. Na početnom ekranu klijentu će se prikazati, ako ima nadolazeći servis, podatci o (kao što je prikazano na slici 14): datumu popravka,

vremenu popravka, nazivu automehaničara, OIB-u automehaničara, opisu kvara, lokaciji gdje će biti servis.

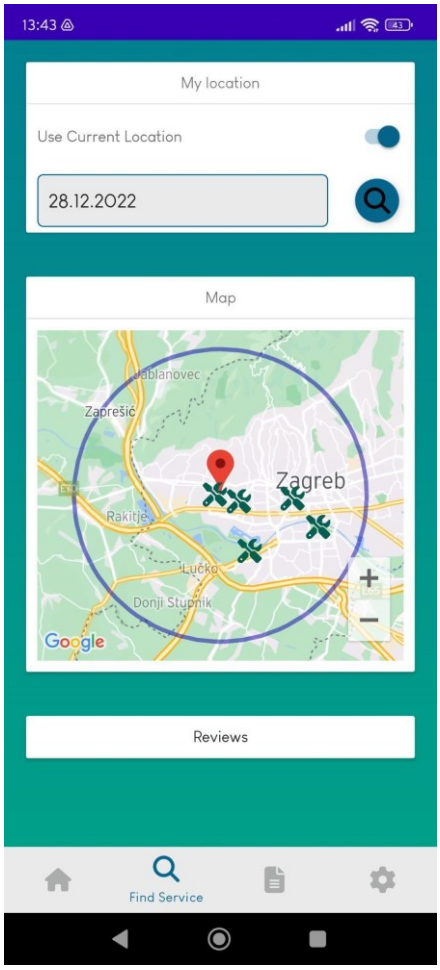
Ako klijent ima aktivan zahtjev za popravak i automehaničar ga je potvrdio, klijentu će se na vrhu ekrana pokazati dvije tipke koje mu omogućavaju da prihvati ili odbije prihvaćeni zahtjev od automehaničara. To služi kao osiguranje klijenta da može odbiti nepovoljne uvjete koje je automehaničar možda postavio, poput predugog roka za popravak.



Slika 5.1.2 - Početni ekran za klijentsku aplikaciju

Također, klijent na početnom ekranu može vidjeti koje je sve probleme naveo automehaničar te ih može prihvatiti ili odbiti.

Ako klijent nema nadolazeći servis, prikazat će se poruka koja ga vodi na pretragu autoservisa. Na ekranu za pretragu automehaničara, klijent ima mogućnost pregleda obližnjih automehaničara, prema svojoj geolokaciji ili prema željenoj lokaciji, kao što je prikazano na slici 15.



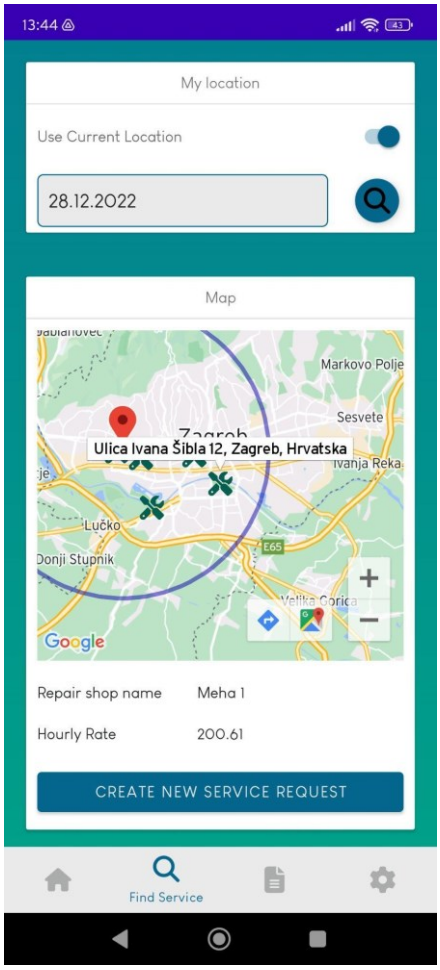
Slika 5.1.3 - Ekran za pretragu automehaničara

Ako klijent želi koristiti svoju trenutnu lokaciju, potrebno je označiti tu opciju. Nakon toga sustav će pretražiti sve automehaničare u krugu od 10 kilometara koji imaju slobodan termin na trenutni datum. Ako se datum želi promijeniti, potrebno je stisnuti na polje s datumom, odabrati drugačiji datum te stisnuti gumb za pretraživanje. Prilikom pretraživanja, sustav će pronaći samo automehaničare koji rade na traženi datum i koji imaju slobodan termin.

Ako klijent ne želi koristiti značajku dohvata trenutne lokacije, opcija se može isključiti. U tom slučaju klijentu će biti ponuđeno polje za unos koje će raditi prijedloge prema klijentovom unosu.

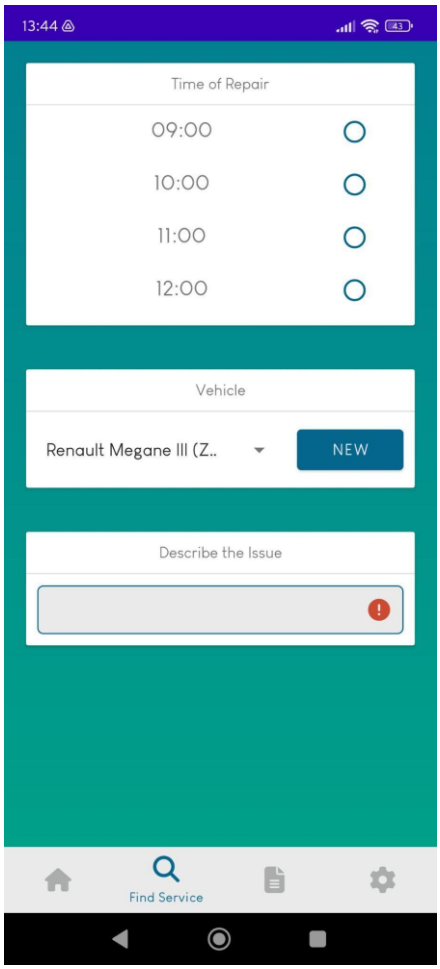
Kada klijent upiše punu lokaciju i označi ju iz ponuđenih prijedloga, oznaka na karti će se pomaknuti na tu adresu i dohvatit će se automehaničari u krugu od 10 kilometara od te adrese.

Nakon označavanja željenog automehaničara na karti, ispod karte će se pojaviti podatak o imenu automehaničara, satnici označenog automehaničara i, ako postoje, recenzije koje su drugi klijenti tog automehaničara ostavili nakon odrađenog servisa, te gumb koji omogućava klijentu da nastavi dalje s izradom zahtjeva za popravkom kod označenog automehaničara (slika 16).



Slika 5.1.4 - Ekran za pretragu automehaničara s označenim automehaničarem

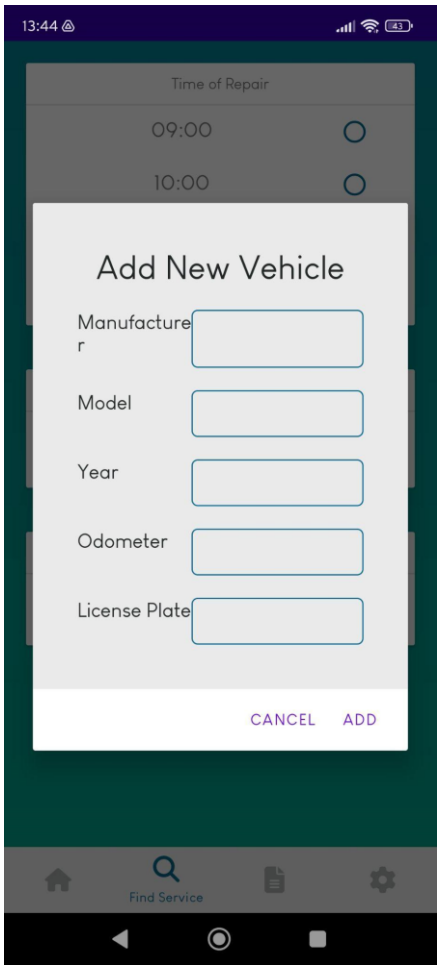
Ako je klijent zadovoljan odabranim automehaničarem i stisne na tipku za stvaranje novog zahtjeva za popravak, klijent je preusmjeren na ekran za stvaranje novog zahtjeva (slika 17). Tamo klijent odabire vrijeme popravka koje mu odgovara i koje je dostupno kod automehaničara. Klijent također mora upisati i opis problema zbog kojeg ide kod automehaničara.



Slika 5.1.5 - Ekran za stvaranje novog zahtjeva za popravak

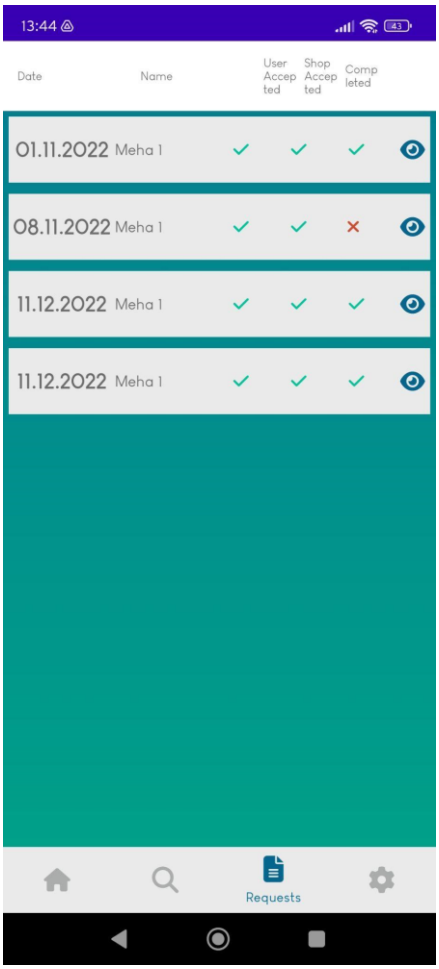
Također je potrebno odabrati i vozilo za koje klijent radi ovaj zahtjev. Ako klijent nema ponuđeno ono vozilo koje želi, može stisnuti na tipku “Novo” te će mu se otvoriti dijaloški prozor s poljima koje treba ispuniti kako bi unio novo vozilo (slika 18). Za novo vozilo potrebni su podaci: marka, model, broj kilometara, godina proizvodnje i registarska oznaka. Klikom na “Dodaj” zatvara se dijaloški prozor te se novo vozilo sprema i prikazuje u padajućem izborniku .

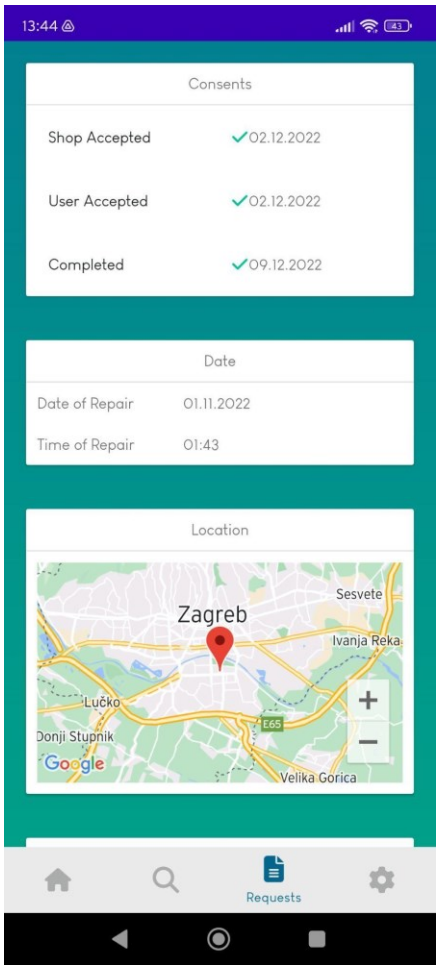
Kada je sve uspješno popunjeno, klijentu se pojavljuje gumb za stvaranje zahtjeva. Klikom na taj gumb, klijent je preusmjeren na početni ekran.



Slika 5.1.6 - Dijaloški prozor za stvaranje novog vozila

Navigiranjem na ekran s popisom zahtjeva, otvara se popis svih stvorenih zahtjeva (slika 19). Zahtjevi su sortirani prema datumu kada servis počinje te se na svakom retku može vidjeti tko je automehaničar za taj zahtjev, koji su pristanci za taj zahtjev i je li servis gotov.





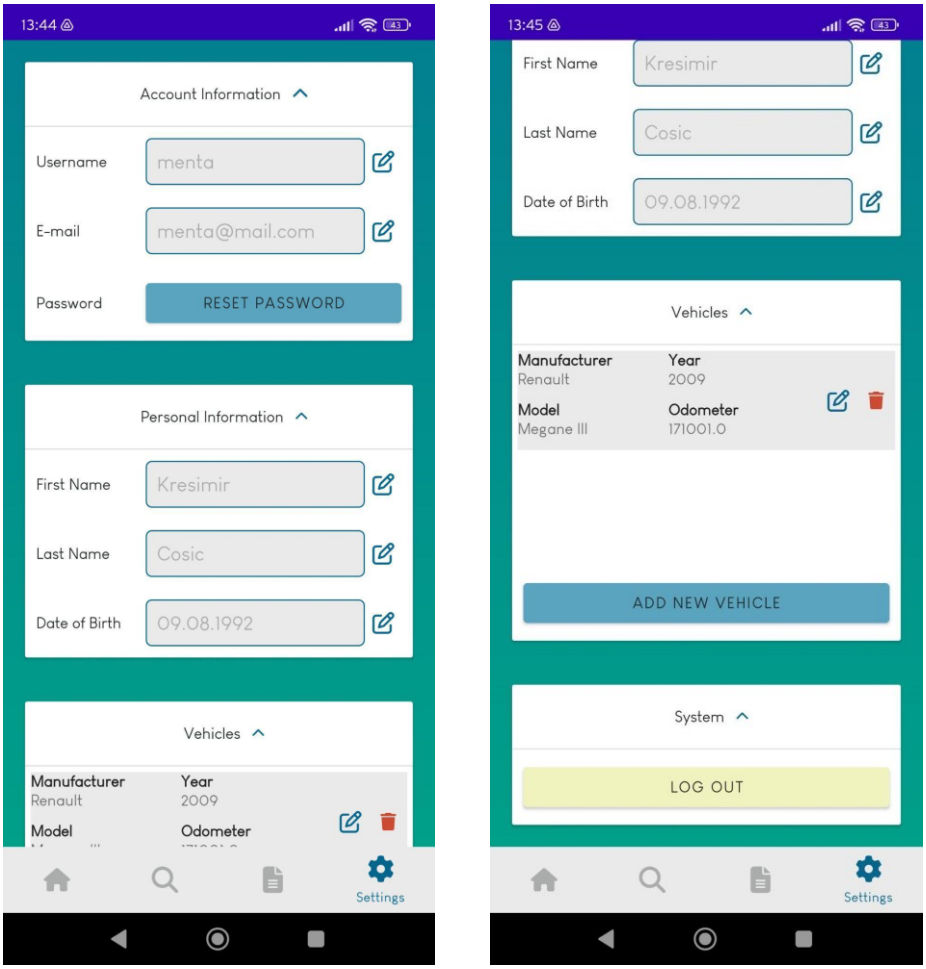
Slika 5.1.7 - Ekran s popisom svih zahtjeva Slika 5.1.8 - Ekran s prikazom jednog zahtjeva

Klikom na oko, klijent je preusmjeren na ekran za pregled jednog zahtjeva (slika 20).

Na tom ekranu klijent vidi sve podatke vezane za zahtjev, kao što su: pristanak automehaničara, pristanak klijenta, je li popravak završen, koji su datum i vrijeme popravka, lokacija popravka, procijenjeno vrijeme trajanja servisa, procijenjena cijena servisa, stvarna cijena servisa, pronađeni problemi i njihova cijena.

Posljednji ekran za klijenta su postavke aplikacije i računa. Na ovom ekranu klijent može promijeniti svoje korisničko ime, adresu e-pošte i lozinku na razini postavke računa. Također se mogu mijenjati ime, prezime i datum rođenja od klijenta u sekciji osobne informacije.

Pod sekcijom “Vozila” klijent vidi popis svih svojih vozila koje ima dodano u sustavu. Svako vozilo može doraditi ili izbrisati klikom na odgovarajuću tipku. Također, ispod liste nalazi se tipka koja omogućava dodavanje novog vozila, kao što je prikazano na slici 18.



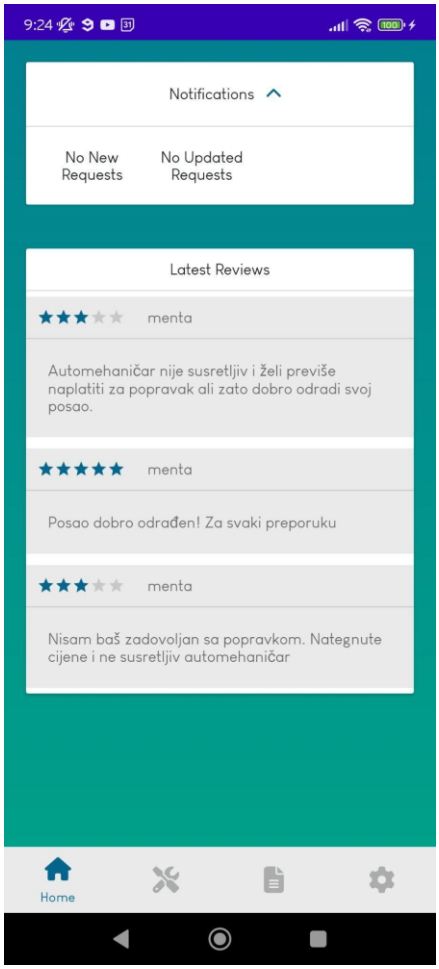
Slika 5.1.9 - Klijentske postavke aplikacije i računa

Klikom na naslov svake sekcije ta se sekcija radi čitljivosti poveća ili smanji. Ako je napravljena promjena, kod naslova sekcije pojavljuje se ikona diskete i to označava da je potrebno stisnuti ikonu kako bi se postavke spremile.

Na dnu ekrana je sekcija “Sustav” gdje se nalazi tipka za odjavu korisnika iz sustava. Klikom na tu tipku, klijent je odjavljen i preusmjeren na ekran za prijavu (slika 12).

5.2. Aplikacija za automehaničara

Kod prijave u aplikaciju, automehaničar na ekranu za prijavu (slika 12) treba unijeti svoju adresu e-pošte, lozinku te označiti opciju da se prijavljuje kao automehaničar. Po uspješnoj prijavi u aplikaciju, automehaničar je preusmjeren na početni ekran. Na početnom ekranu su vidljivi novi zahtjevi, trenutni popravak i posljednje recenzije.

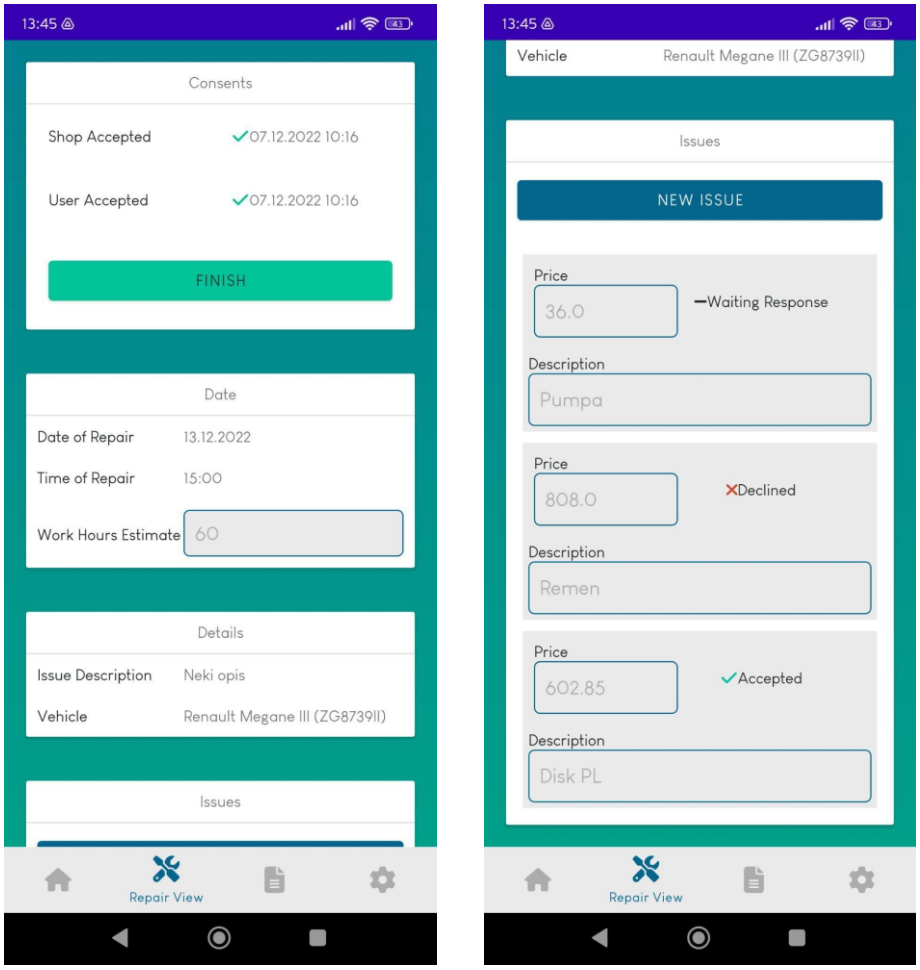


Slika 5.2.1 - Početni ekran za automehaničara

Ako postoje novi zahtjevi za koje automehaničar treba dati pristanak, pojavit će se pod notifikacijama. Klikom na tekst “Novi zahtjevi”, automehaničar je preusmjeren na ekran za pregled pojedinačnog popravka (slika 23).

Na ekranu za pregled pojedinačnog popravka, automehaničar ima pregled svih podataka o označenom servisu. Na vrhu ekrana je sekcija s pristancima. U toj sekciji automehaničar daje ili ne daje svoj pristanak nekom zahtjevu za popravak te može vidjeti ako je klijent dao svoj pristanak ili ne.

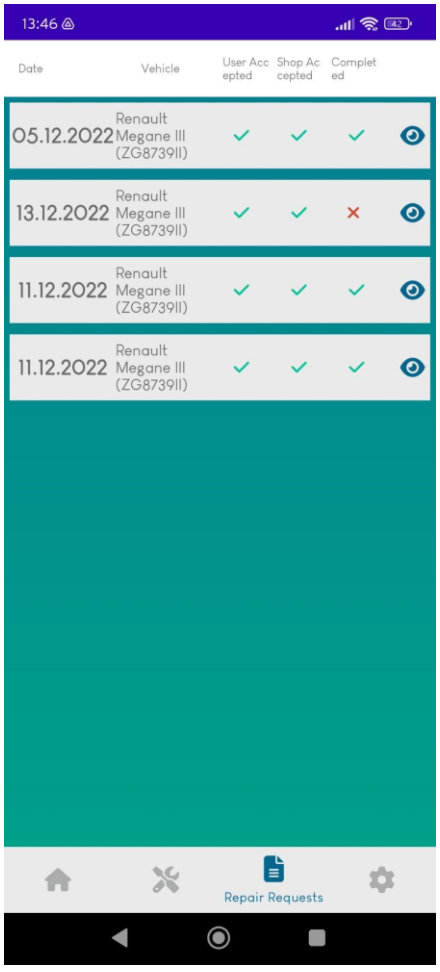
Prije nego automehaničar može potvrditi zahtjev za popravkom od klijenta, on mora unijeti procijenjene sate popravka u sekciji “Datum” kako bi sustav mogao izračunati cijenu i procijenjeni datum završetka.



Slika 5.2.2 - Ekran za pregled pojedinačnog popravka

Pod sekcijom “Datum” također je vidljivo kada klijent traži popravak te u koje vrijeme, dok su pod sekcijom “Detalji” vidljivi opis problema te detalji vozila za koje klijent traži popravak.

Na dnu ekrana je sekcija “Problemi” u kojoj se nalazi lista s problemima koje je automehaničar našao. Svaki problem ima svoj pristanak, svoju cijenu i svoj opis. Ako je trenutni popravak ustvari aktivan popravak, prikazuje se tipka s mogućnošću dodavanja novih problema. Nakon dodavanja novog problema, naslov sekcije dobiva ikonu s disketom koja označava da ima nespremljenih problema. Pritiskom na ikonu iskače dijaloški prozor za potvrdu spremanja. Automehaničar mora dati potvrdu za spremanje jer ne može brisati probleme ukoliko klijent odobri popravak problema.



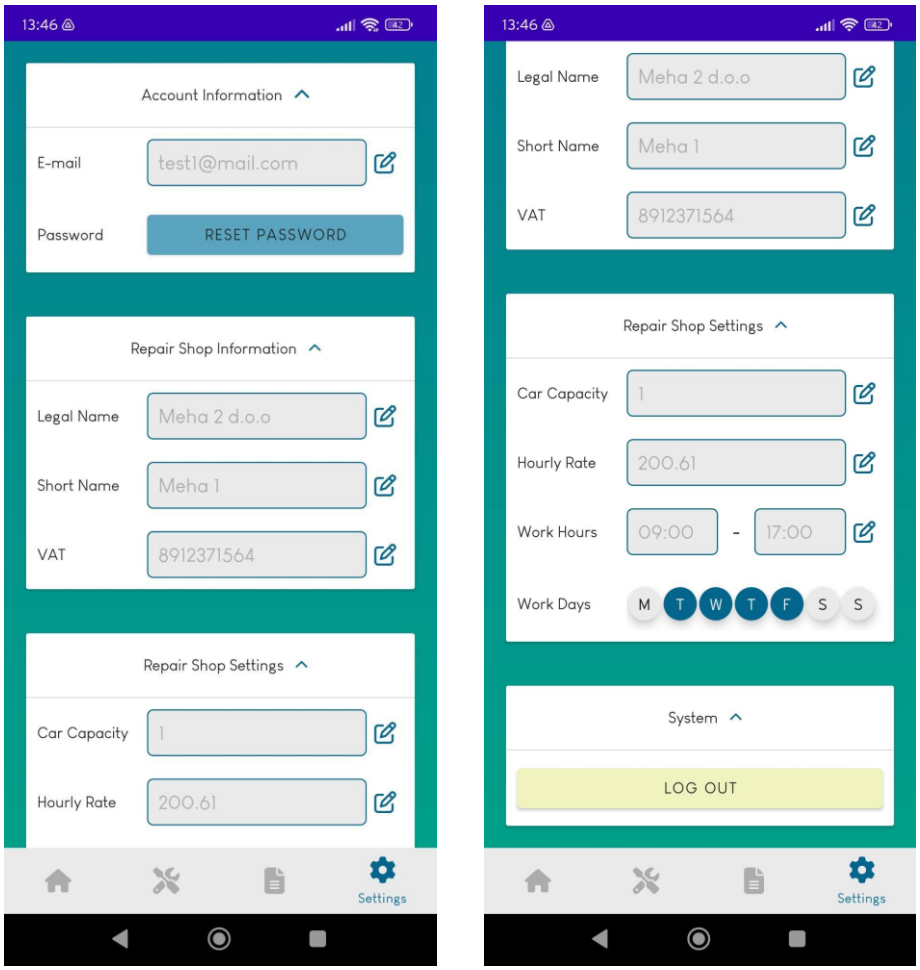
Slika 5.2.3 - Ekran za pregled svih popravaka

Automehaničar može vidjeti sve popravke na ekranu za pregled svih popravaka (slika 24). Svaki red sadrži informaciju o datumu popravka, vozilu u pitanju, danim pristancima i informaciju je li

popravak završen. Klikom na ikonu oka u zasebnom redu, automehaničar je preusmjeren na ekran za pregled pojedinačnog popravka (slika 23) za taj odabrani redak.

Postavke automehaničara (slika 25) nalaze se na istoimenom ekranu. Ekran je podijeljen u 4 sekcije: informacije o računu, informacije o automehaničaru, postavke automehaničara i sistemske postavke.

Pod informacijama o računu nalazi se mogućnost promjene adrese e-pošte i promjena lozinke dok se pod informacijama o autoservisu nalazi mogućnost promjene pravnog imena, skraćenog imena te osobnog identifikacijskog broja od automehaničara.



Slika 5.2.4 - Ekran za postavke sustava i računa automehaničara

Sekcija s postavkama automehaničara sadrži polje za uređivanje kapaciteta vozila kod automehaničara. Ova postavka omogućava automehaničaru da, ovisno o kapacitetu vozila, ima više istovremenih popravaka u jednom vremenskom intervalu. Klijent ne može označiti vremenski interval u kojem automehaničar ima popravak i ima popunjen kapacitet.

Postavke radnih sati i radnih dana se ovdje također mogu promijeniti. Klikom na ikonu za promjenu, omogućava se promjena vrijednosti polja radnih sati, a samim označavanjem polja se otvara dijaloški prozor s mogućnošću odabira sati.

Ispod radnih sati se nalaze radni dani te oni označavaju na koje dane je automehaničar vidljiv kod pretrage klijenta. Dani se mogu promijeniti jednostavnim klikom zasebnog dana, a sprema se klikom na disketu u naslovu sekcije. Spremanje promjena i odjava je ista kao i na klijentskoj aplikaciji.

6. Testiranje programskog rješenja

Nakon stavljanja Car Medic aplikacije u produkciju, 7 osoba ju je testiralo i podijelilo svoja mišljenja. Pet osoba su isprobale korisničko sučelje za klijenta, dok je dvoje ljudi automehaničarske struke, stoga su isprobali korisničko sučelje za automehaničare.

Osobe koje su isprobale klijentsko sučelje bile su jako zadovoljne s mogućnošću pronalaženja automehaničarskih radionica na karti i pozitivno su reagirale na jasne vidljivosti recenzija. S obzirom na to da je aplikacija u ranom stadiju, nije bilo velikog izbora radionica, no sama im se funkcionalnost, kao koncept za buduće razvijanje, svidjela. Što se tiče snalaženja po aplikaciji i intuitivnosti aplikacije, nitko nije imao problema. Prijedlog je bio da se u budućnosti dodatno poradi na komunikaciji između automehaničara i klijenta, odnosno da se ubaci opcija chata. Još je jedan prijedlog bio mogućnost promjene dometa kod geolokacije (trenutno je 10 km, no ‘pin’ se može pomicati ovisno o želji).

Automehaničari su se također brzo snašli u aplikaciji te im se najviše svidjela opcija gdje mogu pregledati dolazne zahtjeve i odmah vidjeti o čemu se okvirno radi. Komentar je bio kako je to super funkcionalnost, no ne znaju kako će to u praksi izgledati jer ljudi najčešće ne znaju objasniti problem s autom.

Treba imati na umu da Car Medic trenutno ima mali broj registriranih korisnika te da zbog toga nije moguće u potpunosti doživjeti punu svrhu aplikacije. Pravi benefit i učinkovitost aplikacije može se prepoznati i vidjeti tek s vremenom i povećanjem broja korisnika, no svakako do tada ima mjesta za napredak. Neke od funkcionalnosti koje bi se s vremenom dodale su mogućnost popravka i drugih vrsta vozila (bicikli i servisi za bicikle, motori i servisi za motore), funkcionalnost izdavanja računa unutar aplikacije i dr.

Zaključak

Program Car Medic je aplikacija za pružanje usluga klijentima i automehaničarima. Predstavlja kompleksnu Android aplikaciju koja olakšava klijentima da brzo i jednostavno pronađu kvalificiranog stručnjaka za popravke automobila. Ova aplikacija sadrži sve ključne funkcionalnosti koje olakšavaju uspostavljanje kontakta između klijenta i automehaničara te optimiziraju poslovanje automehaničara.

Pretraživanje po internetu nudi prevelik izbor automehaničarskih radionica, ali bez jasnih informacija. Car Medic sužava potragu i omogućuje korisniku da na temelju jasnih informacija i recenzija/iskustva drugih korisnika donese pravu odluku i odabere opciju koja mu je najprihvatljivija prema vremenskom okviru i cijeni popravka. Kada dođe do kvara na automobilu, nema potrebe za zvanjem prijatelja i poznanika, već je dovoljno otići na svoj pametni telefon koji će pokazati sve registrirane automehaničarske radionice u krugu od 10 km.

Istraživanje koje je provedeno u sklopu rada pokazalo je kako su vozači u Hrvatskoj zainteresirani za korištenje ovakve aplikacije. Aplikacija sadrži funkcionalnosti koje su korisnici smatrali bitnima, poput pretraživanja automehaničarske radionice u blizini, brzog popravka vozila i prikaza recenzija automehaničarskog rada od strane drugih korisnika. Car Medic je stoga idealno rješenje za vozače koji žele brzo i jednostavno riješiti probleme s automobilom bez nepotrebnog stresa i vremenske neizvjesnosti.

Car Medic, kao i svaka druga aplikacija, zahtijeva redovito održavanje i nadogradnju kako bi njena funkcionalnost i sigurnost bili na visokoj razini. Troslojna arhitektura koja se koristi za izradu ovog softvera olakšava buduće izmjene i ažuriranja.

Car Medic se ističe svojim modernim dizajnom, funkcionalnostima i jednostavnošću korištenja, te tako stvara povjerenje između automehaničara i klijenta, štedi vrijeme, osigurava transparentnost i smanjuje vrijeme potrebno za izbor ponuđene usluge. Ova Android aplikacija predstavlja inovativno rješenje na tržištu koje se može preuzeti putem Googleovog Play Storea i ponuditi široj javnosti.

Popis kratica

AAA	American Automobile Association	Američka udruga automobila
API	Application Programming Interface	Aplikacijsko programsko sučelje
ASP	Active Server Pages	Aktivne poslužiteljske stranice
DTO	Data Transfer Object	Objekti za prijenos podataka
HAK	/	Hrvatski autoklub
HTTP	Hyper Text Transfer Protocol	Protokol za prijenos hiperteksta
IDE	Integrated Development Environment	Integrirano razvojno okruženje
JSON	JavaScript Object Notation	JavaScript notacija objekta
.NET	Network Enabled Technologies	Mrežno omogućene tehnologije
OIB	/	Osobni identifikacijski broj
ORM	Object Relational Mapping	Objektno relacijsko mapiranje
REST	Representational State Transfer	Prijenos reprezentativnog stanja
SDK	Software Development Kit	Komplet za razvoj softvera
SQL	Structured Query Language	Strukturirani upitni jezik
XML	Extensible Markup Language	Proširivi jezik za označavanje

Popis slika

Popis tablica

Popis kôdova

Literatura

1. *Eurostat* (2020) Dostupno na: https://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20220727-1 [20.11. 2022.]
2. *Jutarnji.hr* (2022) Dostupno na: https://www.jutarnji.hr/autoklub/aktualno/prosjecna-starost-vozila-u-hrvatskoj-14-34-godine-a-evo-i-koliko-kilometara-prosjecno-vozimo-15147570 [11.11.2022.]
3. *Adorio* Dostupno na: https://www.adorio.hr/placa/automehanicar [22.11.2022]
4. *Adorio* Dostupno na: https://www.adorio.hr/poslovi?zanimanje=Automehani%C4%8Dar&grad=&radius=0&zupanija=0 [05.12.2022.]
5. *Hubspot - How buyers make decisions* (2017) Dostupno na: https://blog.hubspot.com/sales/how-buyers-make-decisions [29.11.2022.]
6. *Psychologist world - Psychology of Choice* Dostupno na: https://www.psychologistworld.com/cognitive/choice-theory [30.11.2022.]
7. *Decode - How to develop a user-friendly app* (2012) Dostupno na: https://decode.agency/article/user-friendly-app/ [30.11.2022.]
8. *Newsroom - Most U.S. Drivers Leery of Auto Repair Shops* (2016) Dostupno na: https://newsroom.aaa.com/2016/12/u-s-drivers-leery-auto-repair-shops [2.12.2022.]
9. Lamza, Posavec Vesna (2004) *Metode društvenih istraživanja (skripta)*. Zagreb: Institut društvenih znanosti Ivo Pilar
10. *IBM - What is three-tier architecture?* Dostupno na: https://www.ibm.com/topics/three-tier-architecture [14.12.2022.]
11. *Mobile Operating System Market Share Worldwide* Dostupno na: https://gs.statcounter.com/os-market-share/mobile/worldwide [16.12.2022.]
12. *Britannica – Android operating system* Dostupno na: https://www.britannica.com/technology/Android-operating-system [16.12.2022.]
13. *JavaPoint - Android Studio* Dostupno na: https://www.javatpoint.com/android-studio, [17.12.2022.]
14. Jemerov D. i Isakova S. (2017) *Kotlin in action*. Shater Island: Manning Publications Co.

15. Šimec A. i Filipec M. (2021) Kotlin u odnosu na Javu za razvoj na JVM i Android platformi, *Polytechnic & Design*, 9(3), str. 202 - 207
16. *Red Hat - What is an API?* (2022) Dostupno na: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces [18.12.2022.]
17. Kurtz J. i Wortman B. (2014) *ASP.NET Web API 2 - Building s REST Service from Start to Finish.* California: Apress
18. *Packt - The core architectural elements of a RESTful system* Dostupno na: https://subscription.packtpub.com/book/web-development/9781784399092/1/ch01lvl1sec10/the-core-architectural-elements-of-a-restful-system [17.12.2022.]
19. Lerman, J. (2010) *Programming Entity Framework: Building Data Centric Apps with the ADO.NET Entity Framework*. USA: O'Reilly Media
20. *Microsoft - Database First* (2020) Dostupno na: https://learn.microsoft.com/en-us/ef/ef6/modeling/designer/workflows/database-first [15.12.2022.]
21. Petrovečki M. i Kovačević Ž. (2019) Istaknute značajke SQL servera 2019, *Polytechnic & Design,* 7(1), str. 1-7
22. *Scaler Academy, SQL Server Architecture - Detailed Explanation* (2022) Dostupno na: https://www.interviewbit.com/blog/sql-server-architecture/ [17.12.2022.]
23. Philips B., Stewart C., Marsciano K. (2017) *Andorid Programming – The Big Nerd Ranch Guide.* USA: Big Nerd Ranch, LLC
24. *Microsoft Learn: Serialization* (2022) Dostupno na: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/ [09.01.2023.]