Razvoj sustava za upravljanje radom autoservisa

Delinger, Antun

Sveučilište Jurja Dobrile u Puli
Fakultet ekonomije i turizma
„Dr. Mijo Mirković“

Antun Delinger

RAZVOJ SUSTAVA ZA UPRAVLJANJE RADOM AUTOSERVISA

Diplomski rad

Pula, 2018.

Sveučilište Jurja Dobrile u Puli
Fakultet ekonomije i turizma
„Dr. Mijo Mirković“

Antun Delinger

RAZVOJ SUSTAVA ZA UPRAVLJANJE RADOM AUTOSERVISA

Diplomski rad

JMBAG: 0303043755, redoviti student
Studijski smjer: Poslovna informatika
Predmet: Softversko inženjerstvo
Mentor: doc.dr.sc. Tihomir Orehovački

Pula, 2018.

IZJAVA O AKADEMSKOJ ČESTITOSTI

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

Autor

_____

U Puli, _____, _____ godine

IZJAVA

o korištenju autorskog djela

Ja, *Antun Delinger* dajem odobrenje Sveučilištu Jurja Dobrile

u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom *Razvoj sustava za upravljanje radom autoservisa* koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

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

U Puli, _____ (datum)

Potpis

_____

Sadržaj

Uvod

Današnje poslovanje uvelike je otežano bez upotrebe interneta, društvenih mreža i online aplikacija. Kada govorimo o uslužnim djelatnostima, ključno je olakšati komunikaciju klijenata i tvrtki. Velika konkurentnost i sve veća orijentiranost ka modernim rješenjima jednostavno prisiljavaju tvrtke svih djelatnosti da se priključe tom trendu. Jedan od načina za olakšavanje komunikacije i općenito pristupa potencijalnih klijenata su svakako društvene mreže i mobilne aplikacije kojim klijenti na jednostavan način stupaju u kontakt s željenom tvrtkom i informiraju se o određenim uslugama. S obzirom da su društvene mreže ionako prihvaćene i odavno u upotrebi od strane gotovo svih tvrtki, otvara se veliko tržište i potražnja za mobilnim aplikacijama toga tipa.

Android operativni sustav trenutno je ako ne najpopularniji, onda sasvim sigurno najrasprostranjeniji operativni sustav za mobilne uređaje. Prema tome, Android sustav logičan je izbor kao platforma za izradu aplikacije za poslovanje. Dokumentacija, materijali i općenito baza informacija za izradu Android aplikacija izuzetno su opsežni i lako dostupni. Aplikacija razvijena za ovaj sustav vrlo lako se plasira na najpopularniju trgovinu aplikacija – Google Play – gdje ju mogu preuzeti milijuni korisnika.

Kako su aplikacijska rješenja za objekte poput restorana i slično već razvijena u velikom broju i mala je potreba za istima, potrebno je bilo pronaći neku granu koja bi bila u povojima po pitanju upotrebe online aplikacija, ali i gdje bi potreba za korištenjem bila veća. Tako je uočena potreba za izradom aplikacije za upravljanje autoservisom u smislu komunikacije sa korisnicima i naručivanjem raznih usluga putem aplikacije gdje bi objema stranama posao bio olakšan.

Sustav se sastoji od Android aplikacije koja je namijenjena krajnjim korisnicima te Web aplikacije namijenjene administraciji u autoservisu. Tehnologije korištene za Android aplikaciju su Xamarin.Android, programski jezik C#, SQLite te Google Firebase baza podataka. Tehnologije korištene za izradu web aplikacije su Node.js te Express framework.

1. Elektroničko poslovanje

Klasično poslovanje smatra se zastarjelim i ne pretjerano perspektivnim oblikom poslovanja. Iako još uvijek postoje tvrtke koje je prakticiraju elektroničko poslovanje u punom smislu, malo je onih koji se nisu okušali u nekom od područja elektroničkog poslovanja. Elektroničko poslovanje suvremeni je oblik organizacije poslovanja, a koji podrazumijeva intenzivnu primjenu informatičke i, osobito, internetske tehnologije. Čak se, štoviše, može ustvrditi kako je elektroničko poslovanje danas najsuvremeniji oblik organizacije poslovanja, kojem teže svi poduzetnici orijentirani agresivnom osvajanju što boljih tržišnih pozicija i intenzivnom ulaganju u razvojne poslovne aktivnosti. Najvažniji su razlozi zbog kojih se teži primjeni koncepta elektroničkoga poslovanja ovi: težnja za što boljim iskorištenjem svih raspoloživih poslovnih sredstava, a posebno informacija, nastojanje da se ostvari što bolja tržna odnosno konkurentska pozicija tvrtke, želja za ostvarivanjem boljih poslovnih učinaka, posebno onih najpreciznije mjerljivih – financijskih. Koncept elektroničko poslovanja primjenjiv je u svim poduzetničkim, odnosno gospodarskim djelatnostima. Elektroničko se poslovanje, kao tehnološki najnapredniji način organizacije poslovanja tvrtki i njihovih asocijacija, ostvaruje količinski (kvantitativno) i vrsnoćom (kvalitativno) intenzivnom primjenom informatičke, a naročito mrežne računalne tehnologije.[1] U elektroničko poslovanje spadaju sljedeći oblici:[2]

- Elektroničko trgovanje
- Elektronički marketing
- Elektroničko bankarstvo
- Elektroničke burze
- Računalni rezervacijski sustavi
- Prakticirati elektroničko poslovanje
- Postati važnim subjektom na internetskom tržištu
- Ostati u igri

[1] Panian, Željko. Eelektroničko poslovanje - šansa hrvatskoga gospodarstva u 21. stoljeću." *Ekonomski pregled* 51, br. 3-4., 2000., str. 272. https://hrcak.srce.hr/65494 (pristupljeno 1.lipnja. 2018)
[2] Loc.cit.

1.2. Mobilno poslovanje

Mobilno poslovanje kao dio elektroničkog poslovanja odnosi se na poslovanje temeljeno na mobilnim platformama i mobilnim aplikacijama. Mobilno poslovanje, za razliku od elektroničkog, još uvijek je mlado područje i još uvijek nije jasno definirano. Ono što je jasno je da se odnosi na prijenosne platforme i poslovanje u pokretu što jasno aludira na pametne mobilne uređaje. Postoji nekoliko definicija ovog oblika poslovanja, kao primjerice: „Mobilno se poslovanje može definirati kao korištenje mobilnih tehnologija u razmjeni dobara, usluga, informacija i znanja. Mobilno poslovanje je izvršavanje transakcija obavljenih pomoću pokretne opreme, mobilnih mreža, bežičnih ili javnih. Mobilno poslovanje uključuje širok spektar aktivnosti u okruženju poslovanja tvrtke s krajnjim korisnicima (B2C) i među tvrtkama (B2B).“ [3]

Shodno navedenoj definiciji, sljedeći bitan aspekt mobilnog poslovanja predstavljaju – mobilne aplikacije. Mobilna aplikacija je softver dizajniran za pokretanje na pametnom telefonu, tabletu ili nekom drugom pametnom uređaju. Svrha ovakvih aplikacija je interakcija sa korisnikom, a dostupne su o besplatnom i u komercijalnom obliku. Postoje dvije vrste mobilnih aplikacija: nativne i web orijentirane. Nativne aplikacije preuzimaju se sa marketa sa aplikacijama, instaliraju na uređaj te ih je moguće koristiti bez veze sa internetom. S druge strane, web orijentirane aplikacije dizajnirane su kao web stranice, pristupa im se iz Internet preglednika i potrebna je veza sa internetom za njihovu upotrebu. Postoje također i hibridne aplikacije – kombinacija izvornog koda u stilu nativnih aplikacije i korisničkog sučelja u stilu web orijentiranih aplikacija. [4]

Mobilne aplikacije postoje u raznim oblicima i postoje razne podjele, ali osnovna podjela je podjela na mobilne aplikacije i mobilne igrice.

1.2.3. Modeli mobilnog poslovanja

Model poslovanja objašnjava kako tvrtka posluje te na koji način dolazi do profita od poslovanja. Shodno tome, model mobilnog poslovanja također predstavlja način na

[3] R.Nayak: Wireless Tehnologies to Enable Electronic Business, Australia, Queensland University of Tehnology, 2010., str. 511.
[4] Baghbaniyazdi, Sina and Hamed Ferdosara. “The Most Successful Business Model of Mobile Applications: A Comparative Analysis of Six Iranian Mobile Games.“, Tehran Iran, University of Tehran, 2017:., str. 202.

koji tvrtka (u kontekstu izrade i plasiranja mobilnih aplikacija) posluje te na koji način dolazi do profita. Postoji nekoliko različitih modela mobilnog poslovanja, ali najviše ih se odnosi na igrice, dok se za aplikacija najčešće koriste Premium, Freemium ili Subscription modeli. Općenito, modeli mobilnog poslovanja dijele se na šest kategorija:[5]

1. **Komercijalni** (Premium): Model jednostavne logike – developer[6] aplikaciju postavlja na trgovinu aplikacija sa određenom cijenom. Trgovina aplikacija naplaćuje tu cijenu krajnjem korisniku te nakon toga isplaćuje tvorca aplikacija zadržavajući određenu proviziju.
2. **Besplatni** (Free): Aplikacije su potpuno besplatne za preuzimanje na trgovini aplikacijama. Ovo je najrasprostranjeniji model aplikacija.
3. **Komercijalno – besplatni** (Freemium)**:** Kombinacija besplatnog i komercijalnog modela – aplikacija je potpuno besplatna za preuzimanje i korištenje, ali ako korisnik želi bolje iskustvo ili više sadržaja, za to će morati platiti i na taj način steći komercijalnu verziju aplikacije. U osnovi, developer postavlja besplatnu i komercijalnu verziju aplikacije i unutar (ograničene) free aplikacije moguće je platiti naknadu i preuzeti mogućnosti komercijalne verzije.
4. **Kupovina unutar aplikacije** (In-app purchase): Varijanta besplatnog modela. Umjesto ponude (jednokratne) nadogradnje na komercijalnu verziju, korisniku je ponuđeno nekoliko kupovina – prelazak na veći nivo u igrici, kupovina točno određenog sadržaja u aplikaciji i slično.
5. **Reklamiranje unutar aplikacije** (In-app advertising): Također varijanta besplatnog modela. Ovaj model ne nudi mogućnost kupovine dodatnog sadržaja ili nadogradnje, nego prikazuje krajnjem korisniku jedan od oblika reklama. Reklame mogu biti jednostavne i nenametljive (tzv. banner ads) ili agresivne i nametljive koje od korisnika zahtijevaju reakciju (tzv. interstitial ads).
6. **Kombinacijski model:** Model koji kombinira neke od navedenih modela. Developeri mogu plasirati aplikaciju koja kombinira neki od modela te se ne

[5] Ibidem, str. 204
[6] Engl. Developer – tvorac aplikacije. Zbog lakše prepoznatljivosti i uvriježenosti u svakodnevni govor u radu je korišten termin “developer” umjesto “tvorac”.

moraju strogo držati jednog modela - primjerice Freemium i In-app Advertising model.

Ovisno o ciljanom tržištu i potencijalnim korisnicima, developer se odlučuje na jedan od navedenih modela. Glavni cilj kod odabira modela je zarada koja će se njime ostvariti.[7] Osim navedenih modela, postoji još i model izgradnje aplikacije za treću stranu (programer je angažiran od strane klijenta), model pretplate (korisnik plaća određenu mjesečnu naknadu za korištenje aplikacije) te model troška po instalaciji (zarada za svaku instaliranu aplikaciju preko aplikacije koju je developer objavio).[8]

Aplikacija za upravljanje radom autoservisa koristi besplatni model poslovanja za krajnje korisnike, odnosno krajnji korisnici mogu besplatno preuzeti i koristiti aplikaciju. Developer svoj rad naplaćuje modelom pretplate, a taj model pretplate dužan je plaćati vlasnik autoservisa koji bi se odlučio na kupovinu aplikacije.

1.3. Ciljano tržište

Odabir ciljanog tržišta u slučaju aplikacije za upravljanje radom autoservisa je jednostavan i jasan. Potencijalni klijenti za upotrebu (i/ili) kupovinu aplikacije predstavljaju sve autoservise na području države Hrvatske, a potencijalni korisnici predstavljaju sve već postojeće klijente tih autoservisa i onih koji će to tek postati. Prema tome, aplikacija namijenjena za određeni autoservis neće imati veliku bazu korisnika, nego samo onoliko koliko postoji zainteresiranih strana za taj autoservis. Aplikacija bi, sukladno prethodno navedenim uvjetima, bila naplaćivana vlasnicima autoservisa modelom pretplate, dok bi krajnji korisnici besplatno preuzimali aplikaciju.

Developer nema obvezu pozicionirati aplikaciju na tržište. Vlasnici autoservisa sami bi svojim korisnicima plasirali informaciju o novoj aplikaciji te ih na taj način uputili na preuzimanje. S obzirom na već postojeće klijente već postojećih autoservisa, aplikacija ne iziskuje dodatne troškove po pitanju marketinga, naprotiv – predstavlja potencijalno zanimljiv sadržaj za privlačenje novih korisnika zbog povećanja konkurentske prednosti. Upravo povećanje konkurentske prednosti predstavlja najveću prednost ove aplikacije. Naime, na domaćem tržištu trenutno ne postoji slična aplikacija i

[7] Loc.cit.
[8] Ž. Panian, op.cit., str. 276.

konkurencije nema, što bi trebalo potaknuti vlasnike autoservisa na kupovinu aplikacije što bi na koncu dovelo do upotrebe aplikacije od strane krajnjih korisnika.

1.4. SWOT analiza

SWOT analiza je instrument kojim se menadžment služi prilikom kreiranja strategije poduzeća. SWOT predstavlja akronim za:

- S – Strenghts (snaga)
- W – Weaknesses (slabosti)
- O – Oportunities (prilike)
- T – Threats (prijetnje)

SWOT analizom prikazuju se, dakle, unutrašnje snage i slabosti organizacije, kao i vanjske prilike i prijetnje s kojima se ta organizacija susreće. U SWOT analizi važno je zabilježiti ne samo čimbenike koje je moguće kvantificirati, već i one čimbenike koji se ne mogu kvantificirati, a mogu biti spomenuti kao kvalificirana izjava ili uvjerenje. SWOT analiza ima vremensku dimenziju, odnosno kad god je moguće, korisno je uspoređivati i pratiti SWOT analize napravljene za poduzeće u različitim točkama vremena te promatrati promjene stanja, odnosno kretanje poduzeća kroz ovu analizu.[9] Osim što je važno raditi SWOT analizu u kontinuitetu, bez obzira na vrijeme nastanka poduzeća (ili, u ovom slučaju, sustava), od ključne važnosti je provesti SWOT analizu u prvim koracima, odnosno već prilikom planiranja poduzeća. SWOT analiza u tom slučaju pomoći će nam za bolje razumijevanje sustava i njegovog odnosa s tržištem, već postojećim konkurentima i okolinom u kojoj se nalazi. Osim toga, SWOT analiza je jednostavan i pregledan uvid u prednosti i nedostatke sustava te na osnovu te analize možemo praviti prve procjene isplativosti i racionalnosti ulaganja u razvoj poduzeća. Osim procjene isplativosti, možemo dobiti uvid u prve nedostatke i početi planirati poboljšanja ili izvoditi promjene koje su još uvijek moguće. Kao što je već navedeno, SWOT analiza ima vremensku komponentu te snage i slabosti prikazuju trenutno, sadašnje stanje, dok prilike i prijetnje pokazuju buduća kretanja. To nam omogućava kvalitetnu analizu situacije koja u sebi sadržava i sadašnja, ali i buduća kretanja. U slučaju sustava kojim se bavi ovaj rad, naglasak je (prije izrade) bio na

[9] Marli Gonan Božac, SWOT analiza i tows matrica – sličnosti i razlike, Economic research - Ekonomska istraživanja, 2008, str. 21. https://hrcak.srce.hr/21453 (Pristupljeno 7. svibnja .2018)

prilike i prijetnje budući da su iste determinirale potencijalnu uspješnost sustava. Nakon što je utvrđeno da sustav ima realne mogućnosti uspjeti na tržištu, analizirane su snage i slabosti.

Snage	**Slabosti**
Gotovo nikakva konkurencija	Ograničenost na samo jednu platformu
Nije potrebno privlačiti korisnike, oni već postoje	Nemogućnost korištenja bez prijave i registracije
Usluga besplatna za krajnje korisnike	Nemogućnost korištenja bez veze sa internetom
Stvaranje konkurentske prednosti kroz unikatnu uslugu	Potrebno je određeno vrijeme prilagodbe za korištenje upravljačkog dijela aplikacije
Prilike	**Prijetnje**
Veliki broj potencijalnih klijenata	Mogućnost da korisnici ne prihvate novi način poslovanja
Namijenjenost lokalnom tržištu	Eventualna nezainteresiranost potencijalnih klijenata
Inovativnost na tržištu	Mogućnost da klijenti odbijaju naučiti služiti se upravljačkim dijelom aplikacije
U drugim gospodarskim granama ovakav oblik poslovanja je već prihvaćen	Stvaranje početnog zaleta – uhodavanje rada administrator - korisnik

Tablica 1. SWOT analiza sustava

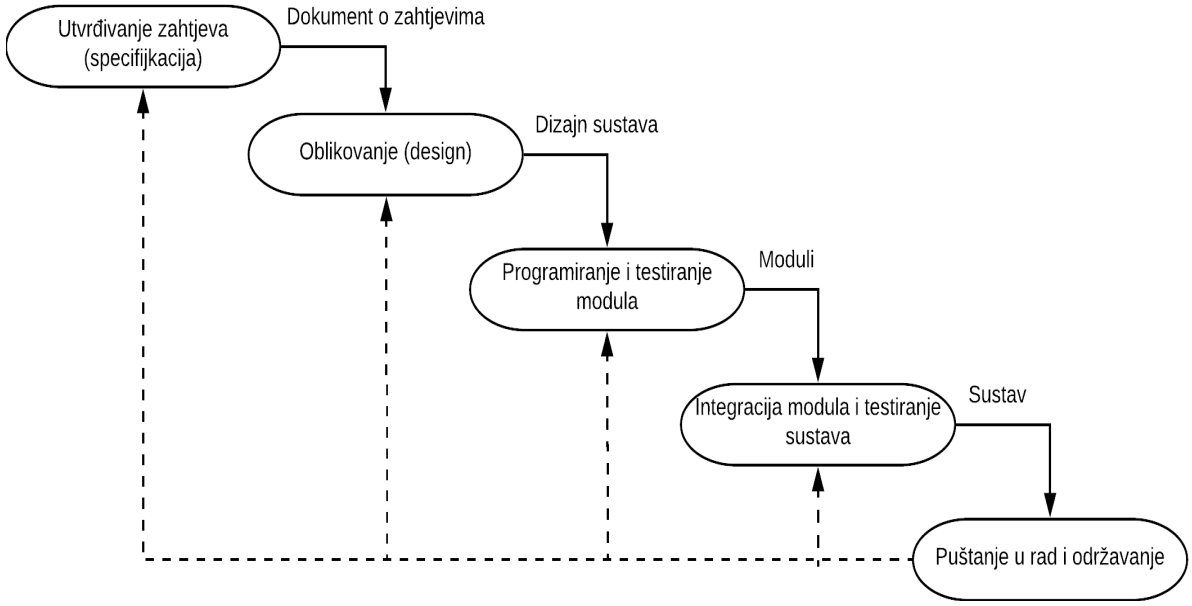
(Izvor: Izradio autor prema Marli Gonan Božac, SWOT analiza i tows matrica – sličnosti i razlike, Economic research - Ekonomska istraživanja, 2008, str. 20. https://hrcak.srce.hr/21453, (pristupljeno 7. svibnja .2018))

Prema tablici 1 vidimo prikaz SWOT analize za aplikaciju upravljanje radom autoservisa. Glavna prednost je, kako je i istaknuto u SWOT analizi, nedostatak konkurencije i samim time stvaranje konkurentske prednosti. Osnovna slabost je ograničenost na jednu platformu, a glavna prijetnja je strah od neprihvaćanja nove

usluge od strane korisnika. Kao najbitnija prilika izdvaja se veliki broj potencijalnih klijenata (autoservis postoji u gotovo svakom gradu) te opća prihvaćenost sličnih usluga u drugim granama gospodarstva.

2. Model sustava

Model koji je korišten u razvoju sustava je – model vodopada (waterfall model). Model vodopada predstavlja model u kojemu je softverski proces građen kao niz vremenski odvojenih aktivnosti. Model nosi naziv prema obliku dijagrama (slika 1).



Slika 1. Waterfall model sustava

(Izvor: Izradio autor prema Sommerville I.: Software Engineering, 9-th Edition. Pearson Education Inc, Boston MA, USA, 2010, str. 30)

Kao što je na slici 1 vidljivo, model vodopada sastoji se od 4 aktivnosti:

1. Utvrđivanje zahtjeva i specifikacija: analiziraju se zahtjevi softvera. Drugim riječima, utvrđuje se funkcionalnost softvera
2. Oblikovanje (design): oblikuje se građa sustava te sučelje između dijelova. Određuje se kako će i što raditi sustav
3. Implementacija (programiranje): oblikovano rješenje realizira se pomoću programskih jezika
4. Održavanje (evolucija): održavanje, mijenjanje i evoluiranje softvera u skladu s promijenjenim potrebama korisnika

Sustav koristi model vodopada kao model sustava iz razloga što je najprikladniji za njegovu izradu. Prije svega utvrđeni su zahtjevi i specifikacije u razgovoru s relevantnim ljudima, zatim je pokrenut proces oblikovanja (na osnovu specifikacija) da bi se zatim prešlo na programiranje oblikovanog sustava, integraciju i na koncu puštanje u rad te održavanje.

3. Funkcionalnost

Osnovna funkcionalnost aplikacije upravljanja radom autoservisa je upravo ono što sam naslov sugerira – olakšavanje rada autoservisa i omogućavanje jednostavne interakcije klijenata i organizacije. Cilj, dakle, aplikacije je preuzeti relevantne informacije i zahtjeve od klijenta, dostaviti ih organizaciji gdje ih revidira osoba zadužena za to. Nakon obrade podataka u organizaciji, putem aplikacije korisnik prima povratnu informaciju. Ta informacija na kraju može biti pozitivan odgovor o narudžbi ili jednostavan odgovor na pitanje. Bitno je razlikovati ove dijelove ovog sustava – klijenti pristupaju informacijama i šalju informacije putem mobilne Android aplikacije, dok organizacija prihvaća podatke te dajte povratnu informaciju putem web sučelja kreiranog za tu svrhu.

3.1. Dionici

Dionici sustava predstavljaju sve osobe koje su dio sustava za upravljanje autoservisom. U to spadaju:

- Registrirani korisnici
- Vodstvo autoservisa

3.2. Utvrđivanje zahtjeva

Utvrđivanje zahtjeva (specifikacija) je aktivnost unutar softverskog procesa gdje se otkrivaju i analiziraju zahtjevi na budući sustav. Rezultat je dokument o zahtjevima koji opisuje što sustav treba raditi, po mogućnosti bez prejudiciranja kako da se to postigne. Utvrđivanjem zahtjeva bavi se tim sastavljen od developera i budućih korisnika. [10] Zahtjevi su važni zbog uravnoteženog proizvodnog procesa sustava – bez skretanja s

[10] I., Sommerville.: Software Engineering, 9-th Edition. Pearson Education Inc, Boston MA, USA, 2010, str. 83

puta i održavanja kontinuiteta. Dobro utvrđeni zahtjevi predstavljaju dobar i kvalitetan temelj za izradu sustava.

Zahtjeve dijelimo na dvije vrste:

- Funkcionalni
- Nefunkcionalni

Zahtjevi su utvrđeni pomoću sljedećih metoda:

- Studija izvodljivosti: provedena je temeljem osobne procjene
- Otkrivanje i analiza zahtjeva: provedeno je razgovorom sa relevantnom osobom
- Dokumentiranje zahtjeva: zahtjevi su dokumentirani prirodnim jezikom, a podijeljeni su na korisničke zahtjeve i zahtjeve na sustav
- Validacija sustava: provedena je s ciljem otkrivanja eventualnih pogrešaka u utvrđenim zahtjevima. Osnovni zadatak validacije bio je prepoznati pogrešku na vrijeme i spriječiti prijenos pogrešaka u daljnje faze razvoja kada su one značajno skuplje za otklanjanje.

3.2.1. Studija izvodljivosti

Temeljem osobne procjene došlo se do zaključka kako je raspoloživa tehnologija dostatna za razvijanje sustava. Uočene potrebe korisnika realno su izvedive i u okvirima dostupnog budžeta, znanja i dostupnog vremena. Prema istoj procjeni, kompleksnost sustava unutar je dozvoljenih granica. Također, uzevši u obzir da je projekt samostalni rad, potrebe za financiranjem nema. Shodno navedenom, analiza i procjena ljudskih resursa također nije potrebna. Kao potrebni resursi za izradu projekta izdvajaju se:

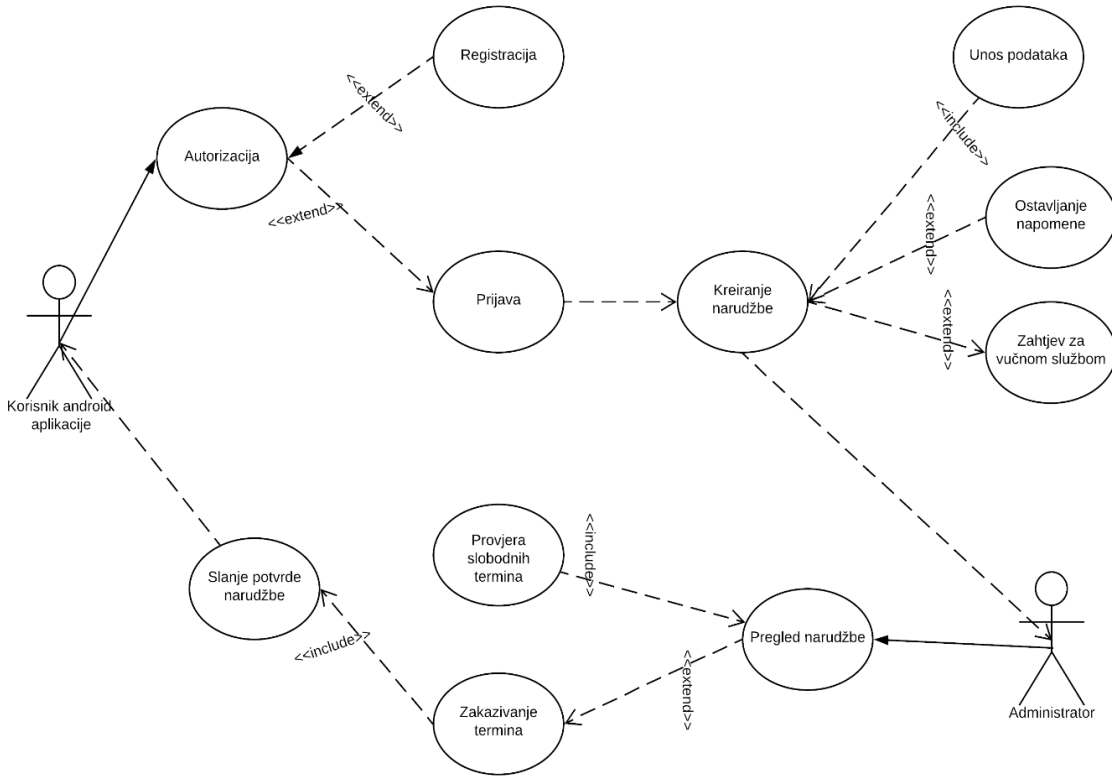
- Poznavanje c# programskog jezika
- Poznavanje rada android operativnog sustava
- Poznavanje SQL baza podataka
- Poznavanje Google Firebase baze podata
- Poznavanje Google Firebase Cloud Messaging tehnologije
- Poznavanje osnova razvoja web stranica (HTML i CSS jezici)
- Poznavanje JavaScript programskog jezika

- Poznavanje rada autoservisa

Gore navedeni zahtjevi su, prema osobno procjeni, u okviru mogućnosti autora te se projekt smatra isplativim. Kao dodatak, autor poznaje rad autoservisa te shodno tome sposoban je samostalno obaviti taj dio zadatka, osim kratkih konzultacija sa relevantnom osobom prilikom kasnijeg utvrđivanja zahtjeva.

3.2.2. Otkrivanje i analiza zahtjeva

Zahtjevi su otkriveni uslijed razgovora sa osobom koja iza sebe ima dugogodišnje iskustvo u djelatnosti gdje bi sustav trebao djelovati te osim toga zahtjevi su otkriveni zamišljanjem scenarija korištenja i usporedbom sa sličnim uslugama. Analiza zahtjeva provedena je pomoću grafičkog jezika UML kojima su prikazani use case i sequence dijagrami.

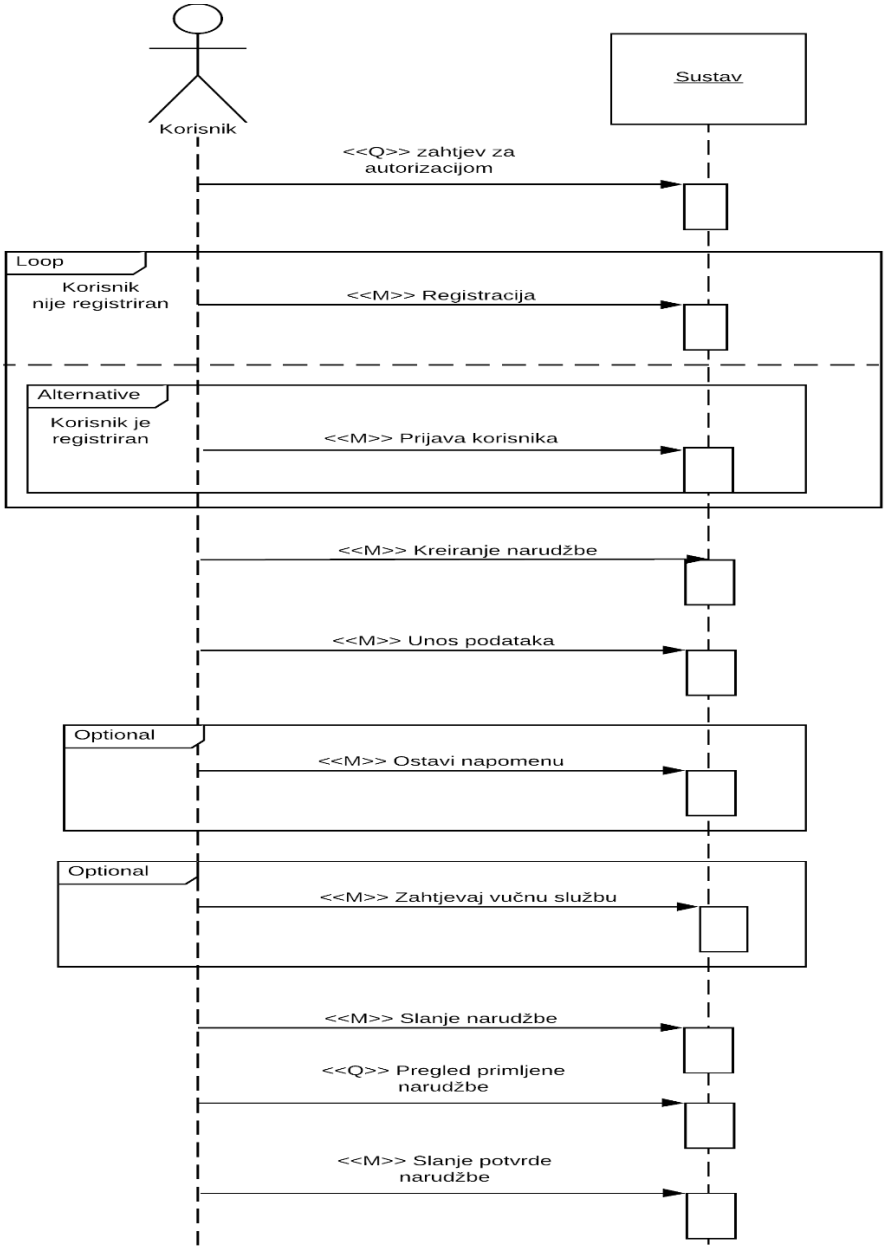


Slika 2. Dijagram slučaja uporabe

(Izvor: izradio autor prema I., Sommerville.: Software Engineering, 9-th Edition. Pearson Education Inc, Boston MA, USA, 2010, str. 107)

Na dijagramu slijeda uporabe (use case) dijagramu (slika 2) vidljiv je rad sustava. Postoje dva aktera – korisnik android aplikacije i administrator sustava (web aplikacija).

Korisnik android sustava kroz prvi slučaj uporabe mora izvršiti autorizaciju koja sadržava registraciju i prijavu. Nakon prijave korisnik kreira narudžbu koja sadržava unos podataka, a može sadržavati i ostavljanje napomene te (također opcionalno) zahtjev za vučnom službom. Narudžbu preuzima administrator u autoservisu te ju pregledava. Pregled uključuje provjeru slobodnih termina te, ukoliko postoji slobodni termin i ukoliko su sve primljene informacije u skladu sa zahtjevima servisa zakazuje se termin. Potvrda narudžbe i zakazanog termina zatim se šalje korisniku android aplikacije.



Slika 3. Dijagram slijeda uporabe

(Izvor: izradio autor prema I., Sommerville.: Software Engineering, 9-th Edition. Pearson Education Inc, Boston MA, USA, 2010, str. 127)

Dijagram slijeda uporabe (sequence dijagram) opisuje slijed korištenja sustava, proširuje dijagram obrazaca s detaljima realizacije pojedinog obrasca. Uvijek ima dva vremenska pravca: korisnik i sustav. Prema slici 3 vidimo prikaz dijagrame slijeda uporabe za sustav upravljanja radom autoservisa. Korisnik je obvezan registrirati se ili prijaviti u sustav prije nego počne sa radom u sustavu. Nakon toga slobodan je kreirati narudžbu i izvršiti neku od opcionalnih radnji – ostavljanje napomene i zahtjeva za vučnom službom. Nadalje, korisnik šalje narudžbu, autoservis ju pregledava i zatim šalje potvrdu natrag korisniku.

3.2.3. Dokumentiranje zahtjeva

Zahtjevi su podijeljeni na dvije kategorije (korisnički zahtjevi i zahtjevi na sustav), a prikupljeni su razgovorom sa relevantnom osobom te kroz osobnu procjenu developera.

1. Korisnički zahtjevi

- Aplikacija mora biti intuitivna i jednostavna za koristiti
- Mogućnost „zapamti me“ opcije kako bi se izbjeglo prijavljivanje kod svakog otvaranja aplikacije
- Mogućnost slanja poruke autoservisu
- Mogućnost ostavljanja napomene prilikom kreiranja narudžbe
- Dostupnost svih informacija o autoservisu

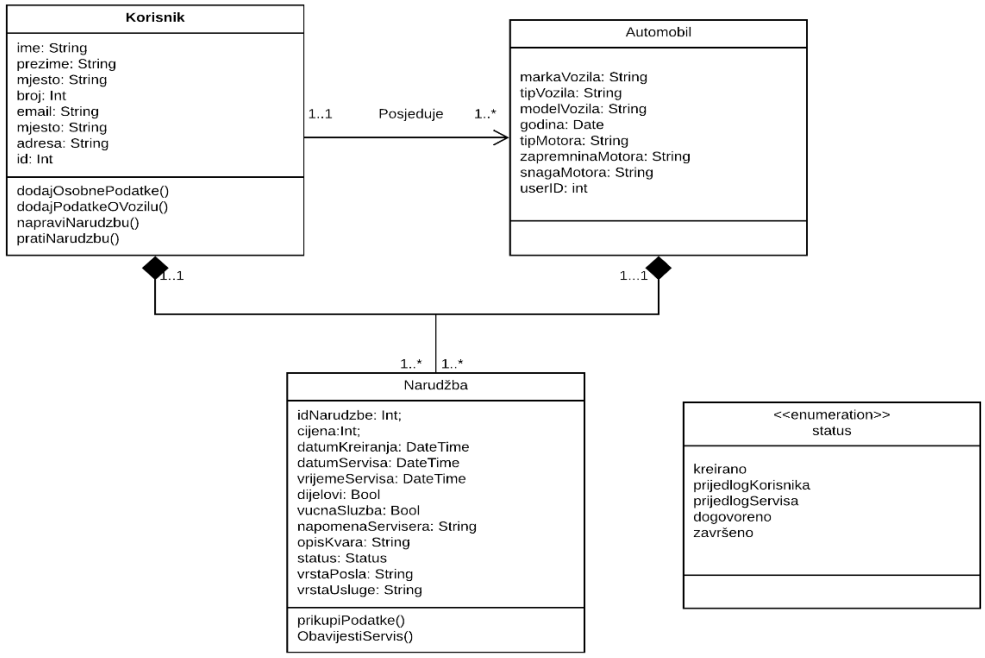
2. Zahtjevi na sustav

- Omogućiti jednostavno i efikasno kreiranje narudžbi
- Omogućiti praćenje starih narudžbi i na taj način vođenje evidencije i servisima vozila
- Omogućiti dodavanje i izmjenu podataka o vozilu kao i stavljanje napomena o vozilu
- Sustav mora biti dizajniran tako da autoservis dobije osnovne informacije (ime, prezime, email, broj telefona) o korisniku odmah prilikom registracije
- Narudžba ne može biti kreirana bez dodatnih podataka o vlasniku vozila (ulica i broj te mjesto)

- Sustav mora omogućiti da se prilikom narudžbe unesu drugi podaci o vlasniku vozila u odnosu na one koji su dodani prilikom registracije
- Prilikom kreiranja narudžbe podaci o vozilu su obavezni
- Zadužena osoba u autoservisu mora biti u mogućnosti vidjeti sve informacije o korisniku i vozilu automobila
- Zadužena osoba u autoservisu mora imati mogućnost kako potvrđivanja narudžbe, tako i mogućnost ponuditi izmjene u narudžbi

3.2.4. Modeliranje sustava

Modeliranje je proces razvoja apstraktnih modela sustava. Postoje dvije vrste modela: modeli postojećeg sustava i modeli novog sustava. Glavno svojstvo modela je da on ističe važne osobine sustava, a ispušta detalje.[11] Model sustava prikazan je na slici (slika 4) pomoću klasnog dijagrama.



Slika 4: Klasni dijagram

(Izvor: izradio autor prema I., Sommerville.: Software Engineering, 9-th Edition. Pearson Education Inc, Boston MA, USA, 2010, str. 139)

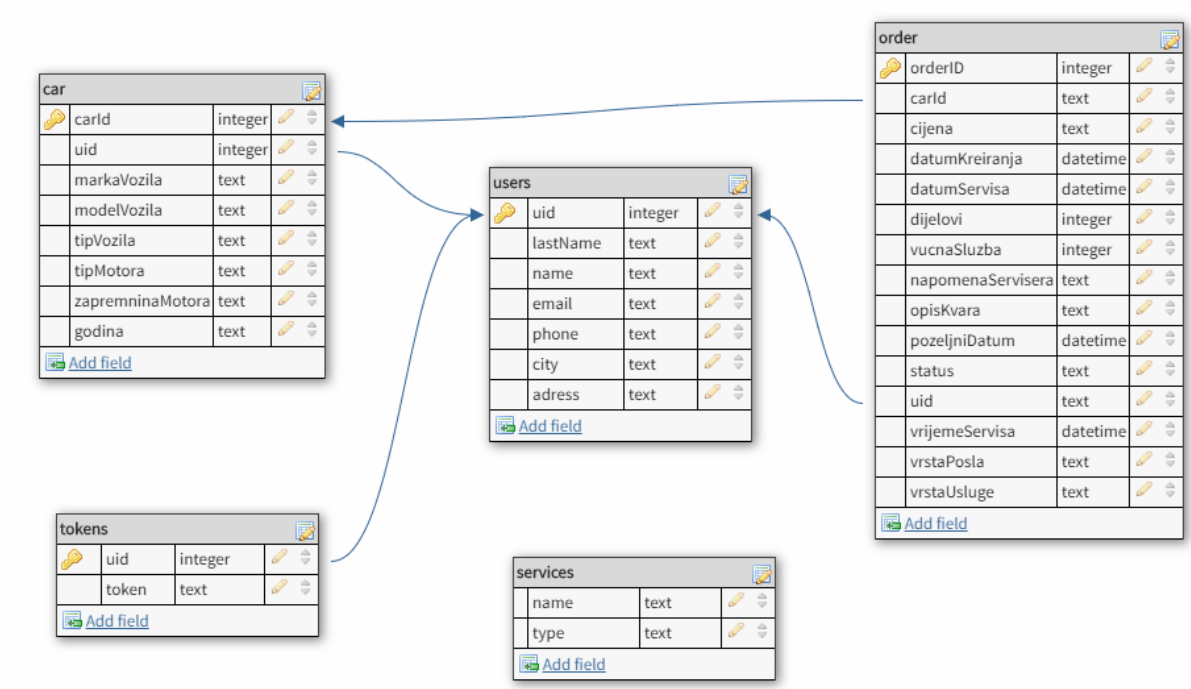
[11] Ibidem, str. 119

Klasnim dijagramom modeliran je sustav te su prikazane najvažnije klase sustava s najbitnijim atributima. Korisnik otpočinje komunikaciju s klasom automobil. Jedan korisnik može posjedovati jedan ili više automobila, dok jedan automobil može imati samo jednog vlasnika. Klasa narudžba spojena je s klasama korisnik i automobil tipom veze agregacija koja označava sadržavanje - klasa narudžba sadržava atribute klase korisnik i automobil. Jedna narudžba može imati jednog korisnika, dok jedan korisnik može imati više narudžbi. Isto vrijedi i za vezu narudžba – automobil. Klasa narudžba i usluga također su povezani vezom agregacija te jedna predstavljaju vezu jedan na jedan – jedna narudžba može imati jednu uslugu i jedna usluga može pripadati jednoj narudžbi.

Drugim riječima, da bi se u sustavu izvršila narudžba, korisnik mora dodati svoje informacije u sustav te informacije u svojemu vozilu. Te informacije, zajedno sa informacijom o tipu usluge, predane su u obliku narudžbe i poslane na obradu.

4. Baza podataka

4.1. Konceptualni model podataka



Slika 5: Konceptualni model baze podataka
(Izvor: izradio autor)

Slika 5 prikazuje jednostavan konceptualni model podataka. Entitet „users“ sadrži primarni ključ „uid“ koji se pojavljuje kao vanjski ključ u druge tri tablice – „order“, „car“ i „tokens“ što ga čini najvažnijim entitetom. Entitet „car“ predstavlja vozilo koje „user“

može posjedovati te ima jedinstveni ključ „carId“. Entitet „order“ predstavlja pojedini zahtjev ili narudžbu koje entitet „users“ može kreirati, a kao jedinstveni ključ istaknut je „orderID“, sadrži dva vanjska ključa – „uid“ (referira se na entitet „users“) i „carId“ (referira se na entitet „car“. Entitet „tokens“ se odnosi na ključ uređaja kojim entitet „users“ pristupa sustavu. Posljednji entitet, entitet „services“ predstavlja listu usluga koje autoservis nudi klijentima. Nije vezan za druge entitete, a atribut „name“ označen je kao unikatan (engl. Unique) budući da se naziv usluge ne može ponavljati.

4.2. Firebase baza podataka

Firebase je baza podataka u vlasništvu tvrtke Google. Dio je Firebase projekta koji se sastoji od velikog broja proizvoda, kao primjerice Cloud Messaging, AdMob, Authentication i slično. Niti jedna od usluga nije obavezna za korištenje druge, sve su potpuno opcionalne i neovisne jedna o drugoj. U sklopu ovog projekta korištene su tri usluge: RealTime Database, Cloud Messaging i Authentification.

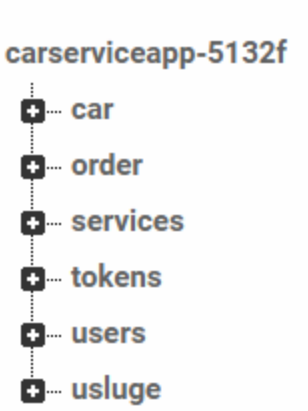
Firebase RealTime Database je baza podataka koja se ažurira u stvarnom vremenu, bez potrebe za dodatnim osvježavanjem i slanjem upita u bazu podataka te čekanjem na odgovor. Podaci se automatski ažuriraju na sustavu na kojemu su implementirani i to u istom trenutku kada je došlo do promjene podataka. Firebase baza podataka također omogućava izvanmrežno korištenje budući da se svi podaci spremaju na memoriju uređaja za slučaj da je uređaj ostao bez veze na Internet. U tom slučaju, podaci se čitaju iz memorije uređaja, a nakon ponovnog povezivanja na Internet podaci u memoriju uređaja se sinkroniziraju s najnovijom instancom podataka. Ono što Firebase bazu podataka izdvaja je arhitektura. Naime, Firebase baza podataka je NOSQL baza podaka, što znači da ne dijeli arhitekturu s ostalim, češće upotrebljavanim SQL bazama podataka. Podaci se u Firebase bazi spremaju kao JSON[12] objekti te Firebase baza poprima izgled JSON stabla.[13]

4.2.1. Primjena Firebase baze podataka

Unatoč konceptualnom modelu, u Firebase bazi podataka svi podaci su tipa String zbog mehanizma funkcioniranja Firebase baze podataka. Nadalje, prilikom kreiranja svakog čvora njemu je dodijeljen globalno jedinstveni ključ, a on se sastoji od 120

[12] JSON – format za razmjenu podataka
[13] L. Moroney, The Definitive Guide to Firebase, Springer Science, Business Media New York, 2017, str.51

bitova informacije – prvih 48 predstavlja vremensku oznaku, a ostatak znakova čine nasumični znakovi što rezultira minimalnim šansama za stvaranjem dva objekta koja posjeduju isti ključ. Taj ključ se zatim kodira base64 kodom u ASCII način kodiranja znakova. Kod je prilagođen na taj način da kreirani ključevi budu kronološki poredani, ovisno o trenutku generiranja. [14] Način kreiranja i upisa u bazu podataka opisan je u poglavlju 5.3. Upis u bazu podataka u potpunosti se odvija u okruženju mobilne aplikacije, dok se u okruženju web aplikacije izvršava samo ažuriranje već postojećih podataka. Svi elementi baze podataka prikazani su u grafičkom obliku kako su prikazani na upravljačkoj ploči Firebase baze podataka zbog bolje preglednosti u odnosu na prikaz u JSON formatu.



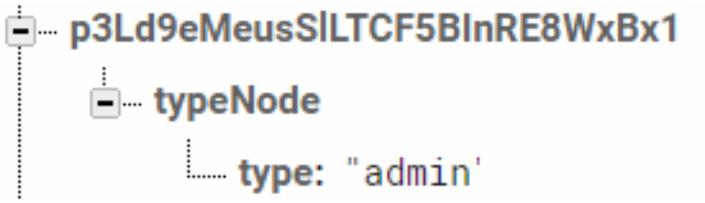
Slika 6: Jednostavan prikaz baze podataka
(Izvor: izradio autor)

Slika 6 pokazuje najjednostavniji prikaz čvorova baze podataka. Vidljivi su svih pet entiteta, a entitet „services“ podijeljen je u dva čvora – „services“ i „usluge“ zbog lakšeg dohvaćanja podataka. Svaki čvor ispod naziva ima dodijeljen jedinstveni ključ. Na primjeru korisnika, odnosno čvora „users“, naziv čvora je, očekivano, „users“, zatim se unutar čvora nalaze svi korisnici koji su registrirani u sustav. Svaki čvor korisnika naslovljen je s njegovim jedinstvenim ključem zbog lakšeg upravljanja podacima, a taj jedinstveni ključ nastao je prilikom registracije korisnika. Ispod jedinstvenog ključa korisnika, nalazi se jedinstveni ključ tog čvora koji je Firebase kreirao priliko kreiranja tog čvora. Odmah ispod nalaze se podaci o korisniku. Čvor korisnika prikazan je na slici 7.

[14] Loc.cit.

Slika 7: Prikaz korisnika u bazi podataka

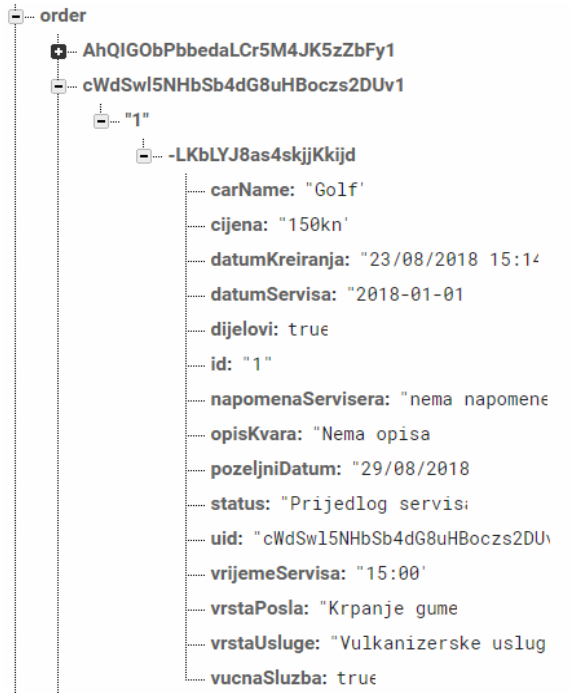
Svaki korisnik mobilne aplikacije ima istu strukturu podataka u Firebase bazi podataka. No, korisnik web aplikacije ima jedinstvenu strukturu i to iz razloga jer je njegova račun kreiran od strane administratora. Njegove informacije, dakle, nisu potrebne te se u bazu podataka sprema samo jedna informacija – tip korisnika. Taj atribut koristi se prilikom prijave u web aplikaciju; prijaviti se može samo korisnik koji pod tip korisnika (čvor „typeNode“) ima vrijednost „admin“. Struktura je prikazana na slici 8.



Slika 8: prikaz atributa tipa korisnika u bazi podataka
(Izvor: snimka zaslona na računalu autora)

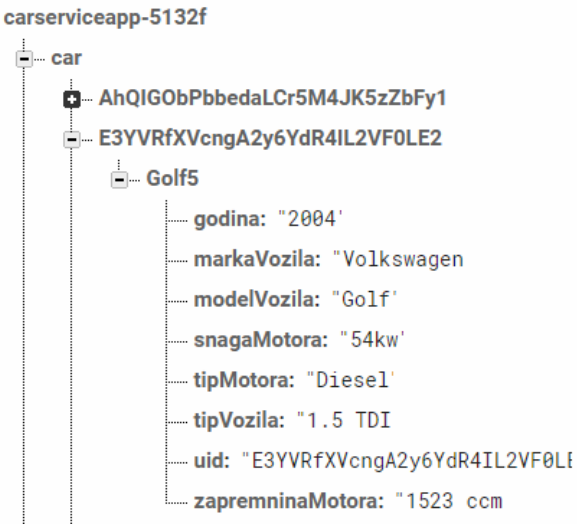
Drugi najvažniji čvor u bazi podataka – čvor „Order“ – odnosi se na svaki kreirani zahtjev i sve relevantne informacije. Čvor u potpunosti slijedi konceptualni model podataka, a također slijedi pravilo Firebase baze podataka - naslovljen je s jedinstvenim ključem korisnika te jedinstvenim ključem tog čvora. Razlika u ovom slučaju je u drugom čvoru ispod jedinstvenog ključa; na tom mjestu nalazi se jedinstveni broj zahtjeva tog određenog korisnika. Taj broj kod svakog korisnika počinje s brojem jedan te se automatski uvećava za jedan kod svakog novog zahtjeva. Bitno je napomenuti da se to uvećavanje ne izvodi automatski u bazi podataka kao što je to slučaj kod SQL baza podataka; Firebase baza podataka nema tu mogućnost pa je taj

dio programiran kroz algoritam smješten u mobilnoj aplikaciji, a koji se izvodi prilikom upisa podataka o zahtjevu u mobilnoj aplikaciji. Čvor „order“ prikazan je na slici 9.



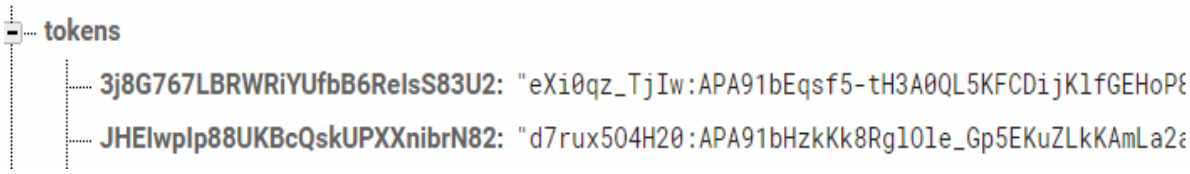
Slika 9: Prikaz podataka o zahtjevu u bazi podataka
(Izvor: snimka zaslona na računalu autora)

Entitet „car“ također je izveden sukladno konceptualnom modelu. No, ovaj čvor nije izveden u skladu s konvencionalnim kreiranjem čvorova u Firebase bazi podatka pa je tako ovdje ispod jedinstvenog ključa korisnika smješten jedinstveni naziv vozila konkretnog korisnika.



Slika 10: Prikaz podataka o vozilu u bazi podataka
(Izvor: snimka zaslona na računalu autora)

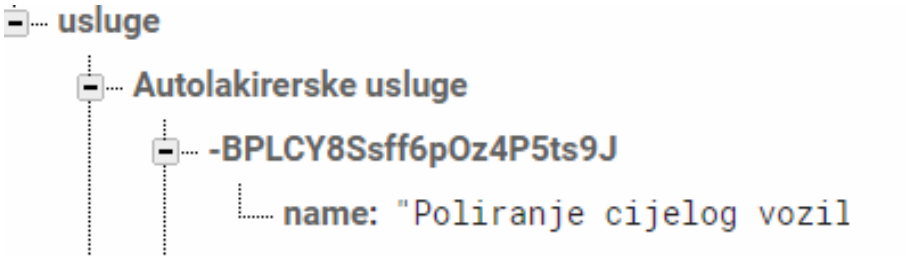
Prema konceptualnom modelu, jedinstveni ključ za entitet „car“ zadan je „carId“ tipa Integer, no budući da Firebase baza podataka nema tip podataka Integer, odabran je tekstualni oblik jedinstvenog ključa. Još jedan od razloga bio je i lakša prepoznatljivost i bolje korisničko iskustvo s ovakvim načinom upotrebe jedinstvenog ključa vozila budući da će prema njemu, jedinstvenom ključu, korisnik identificirati pojedino vozilo koje je dodao u sustav. Slika 10 prikazuje praktičnu izvedbu entiteta „car“ u Firebase bazi podataka. Posljednji entitet vezan uz korisnika je „tokens“, a on je izveden točno onako kako je zadano u konceptualnom modelu. Jednostavna struktura sa samo jednim čvorom unutar glavnom čvora; taj čvor naslovljen je jedinstvenim ključem korisnika, a njemu je pridodana vrijednost tokena[15]. Izvedba je prikazana na slici 11.



Slika 11: Prikaz informacija o tokenu korisnika u bazi podataka

(Izvor: snimka zaslona na računalu autora)

Posljednji entitet, entitet „services“, odvojen je (kao što je već ranije spomenuto) na dva čvora. Prvi čvor, istoimenog naziva, „services“ sadrži samo informaciju o kategoriju posla koju tvrtka nudi, a drugi čvor nudi kompletne informacije o kategoriji i o tipu posla. Razlog je lakše čitanje podataka u bazi podataka, a s obzirom da se ne očekuje velika količina podataka u tim čvorovima nema opasnosti od pada performansi zbog redundancije. Na slici 12 prikazan je čvor „usluge“ i njegova izvedba.



Slika 12: Prikaz usluga prikazanih u bazi podataka

(Izvor: snimka zaslona na računalu autora)

[15] Token – jedinstveni ključ uređaja na kojemu je instalirana Android aplikacija, a koji koristi Firebase Cloud Messaging.

4.3. SQLite baza podataka

SQLite je relacijski DBMS[16] upečatljivih karakteristika: lagan, jednostavan za održavanje, bez potrebe za konfiguracijom, te SQL – orijentiran. Drugim riječima, osmišljen je kako bi predstavljao kompaktan, samodostatan i jednostavan sustav baza podataka. SQLite sprema cijelu bazu podataka u jednu nativnu datoteku koja drži sve tablice i indexe. Kao mana SQLite DBMS-a najčešće se navodi – očekivano s obzirom da za cilj ima jednostavnost i kompaktnost – manjak mogućnosti u odnosu na robusnije baze podataka. To je kompromis koji se može podnijeti s obzirom na prednosti SQLitea. Jedna od najvažnijih je ta da cijela SQLite baza podataka ne iznosi više od 500kb te je prema tome pogodna za uređaje koji nemaju velike količine memorije kao što su pametni mobiteli, pametni satovi i slično.[17]

4.3.1. Primjena SQLite baze podataka

Budući da je tehnologija izrade mobilne aplikacije nekonvencionalna, odnosno odabrana je varijanta programskog jezika C# u kombinaciji sa xamarin.android[18], to je rezultiralo nešto slabijom podrškom za Firebase bazu podataka. Točnije, baza podataka je nešto teža za implementaciju, a u slučaju ovog sustava izvanmrežne mogućnosti Firebase baze podataka pokazale su se nedovoljno konzistentnima i stabilnima kako bi bile implementirane u aplikaciju. Stoga je uz Firebase bazu podataka implementirana i SQLite baza podataka. Problemi s nemogućnošću izvanmrežnog rada Firebase baze podataka reflektirali su se na samo korisničko iskustvo – aplikacija nije učitavala podatke iz memorije uređaja nego je i za najjednostavnije dohvate morala slati upit poslužitelju na server te čekati odgovor servera. To je glavni razlog upotrebe dodatne izvanmrežne baze podataka, a uz čiju upotrebu je eliminirano čekanje prilikom jednostavnih upita.

Upotreba SQLite baze podataka je izvedena na način da je prilikom kreiranja upisa u Firebase bazu podataka izveden upis i u SQLite bazu podataka. Upisi su gotovo identični, uz jedinu razliku što su poštivana pravila SQL baze podataka, odnosno SQLite baza podataka je kreirana prema konceptualnom modelu. SQLite baza podataka koristi se u mobilnoj aplikaciji prilikom čitanja podataka o korisniku,

[16] Akronim za Database Management System, odnosno sustav za upravljanje bazama podataka
[17] S. Haldar, SQLite Database System Design and Implementation, Second Edition, Version 2, Self-Publishing, 2015, str. 37
[18] Softver koji omogućava razvoj Android aplikacija pomoću C# programskog jezika

autoservisu, uslugama i tipovima posla autoservisa te informacije o vozilima. Te informacije se, kao što je već spomenuto, zapisuju prilikom svakog upisa u Firebase bazu podataka, a sinkroniziraju se prilikom svakog pokretanja aplikacije. Jedina iznimka su informacije o zahtjevima iz razloga što se te informacije mogu promijeniti u svakom trenutku te korisnik treba imati najnovije informacije prilikom svakog pokretanja. Više o načinu implementacije i korištenju SQLite baze podataka u stvarnoj primjeni prikazano je u poglavlju 5.4.

5. Oblikovanje i implementacija sustava

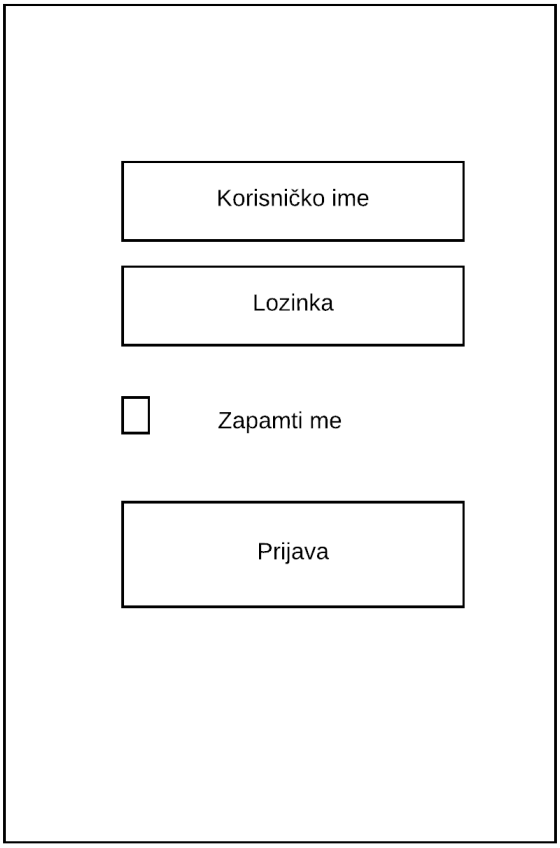
Sustav je, kao što je već prema utvrđenim zahtjevima zaključeno, planiran za Android mobilni operativni sustav. Sustavom bi se upravljalo putem web aplikacije koja je detaljnije razrađena u 9. poglavlju.

5.1. Prototip sučelja

Obje aplikacije, web i mobilna, namijenjeni su, sudeći po utvrđenim zahtjevima, za nacionalno tržište te samim time nema potrebe za dvojezičnošću unutar aplikacija. Dakle, sučelje sustava biti će izvedeno na Hrvatskom jeziku. Naglasak na sučelju je da ono mora biti vizualno oku ugodno te jednostavno i intuitivno. Po pojedinačnom prozoru ne smije biti previše elemenata kako bi se zadržala preglednost istog. Osim naglaska na minimalizmu, još je jedno bitno ograničenje: jednostavnost boja. Ne smije biti previše kombinacija boja – jedna do dvije boja po jednoj stranici predstavlja optimalan odabir.

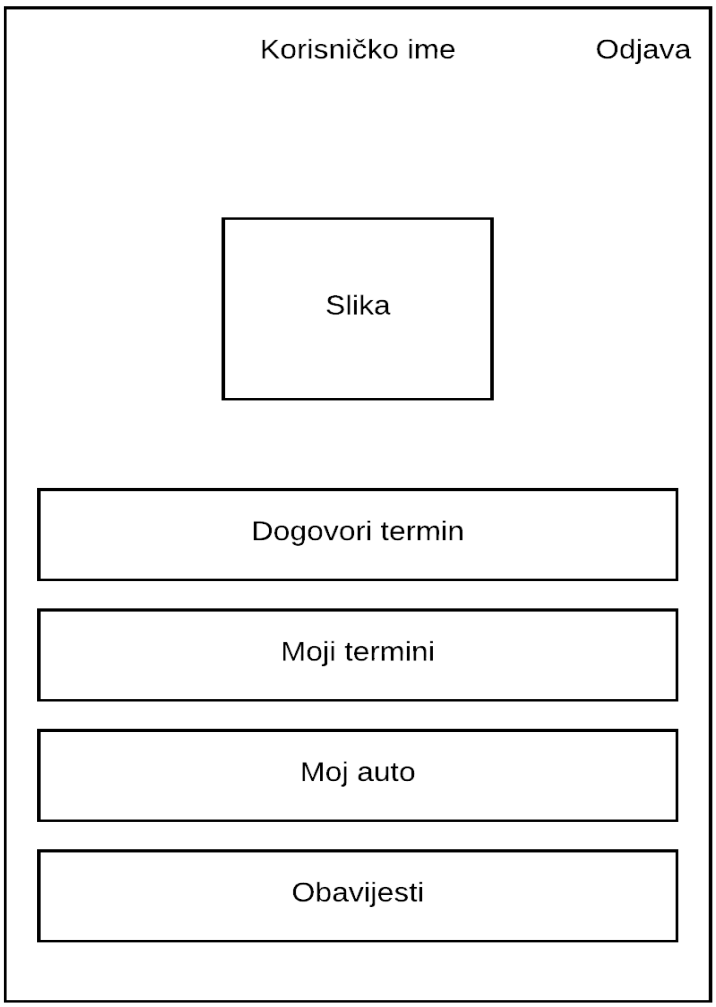
5.1.2. Prototip sučelja mobilne aplikacije

Prema zahtjevima, korisnik bi prilikom otvaranja aplikacije trebao imati mogućnost prijave u sustav pomoću korisničkog imena ili lozinke. Taj ekran sastoji se od osnovnih elemenata kao što su gumb za prijavu i polja za unos korisničkog imena i lozinke. Prototip prvog ekrana (ekrana za prijavu) prilikom otvaranja prikazan je na slici 13. Glavna stranica je zamišljena sa slikom prikladnom automobilskoj industriji, a koja je ili slobodna za komercijalnu upotrebu ili izrađena vlastoručno od strane autora.



Slika 13: Prototip stranice za prijavu
(Izvor: Izradio autor)

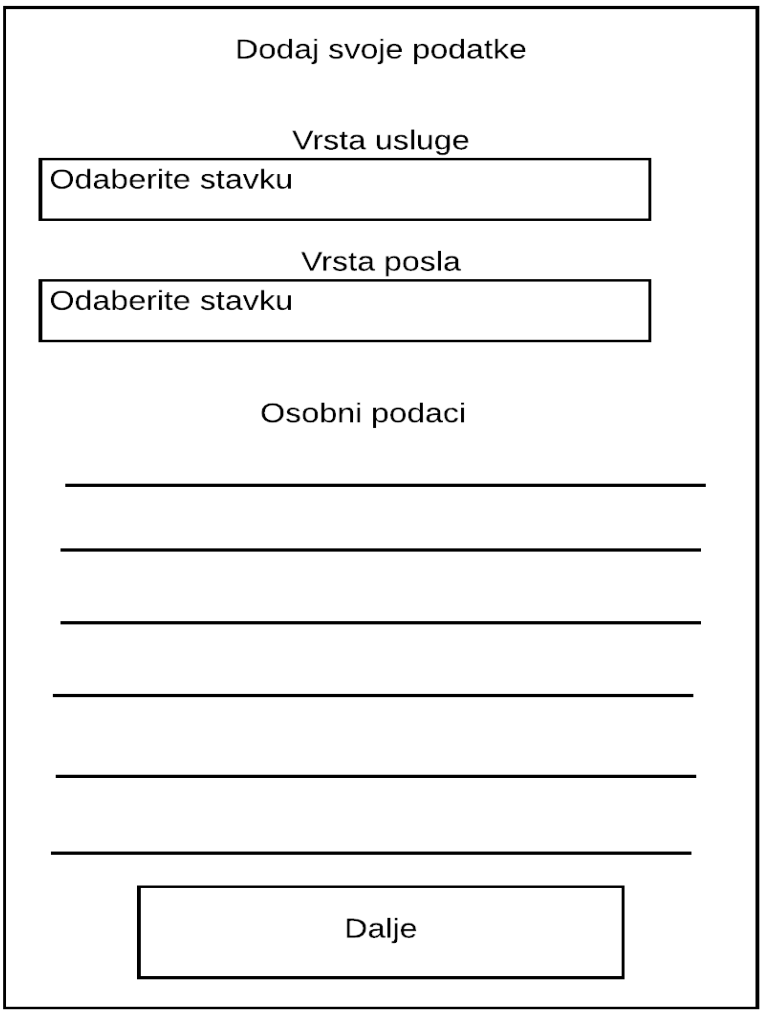
Odmah ispod slike nalazi se nekoliko gumbova koji će voditi na nove stranice. Mora postojati tekst koji označava koji korisnik je trenutno prijavljen u sustav te mora postojati gumb za odjavu, gumb za prikaz osobnih podataka te gumb za prikaz informacija o autoservisu. Gumbi za odjavu, prikaz osobnih podataka te prikaz informacija o servisu trebali bi, ali nije nužno, biti izvedeni u obliku padajućeg izbornika u desnom gornjem dijelu prozora aplikacije. Tematska slika i oznake prijavljenog korisnika te gumb za odjavu ili, drugim riječima, gornji dio ekrana, trebaju predstavljati jednu trećinu ekrana, a ostali elementi, odnosno gumbi za navigaciju aplikacijom, trebaju zauzeti ostatak ekrana, što zbog preglednosti, što zbog izgleda ekrana. Naglasak je na minimalnosti – elementi ne smije biti puno, točno koliko je potrebno da se ispuni funkcionalnost za koju je stranica namijenjena. Elementi ne smiju biti sitni, ali niti preveliki, točno onoliko da ne zauzimaju previše prostora, ali isto tako i da stranica ne izgleda prazno. Stranica ne smije imati više kombinacija boja, maksimalno dvije. Izgled glavne stranice prikazan je na slici 14.



Slika 14 : Prototip glavne stranice mobilne aplikacije

(Izvor: Izradio autor)

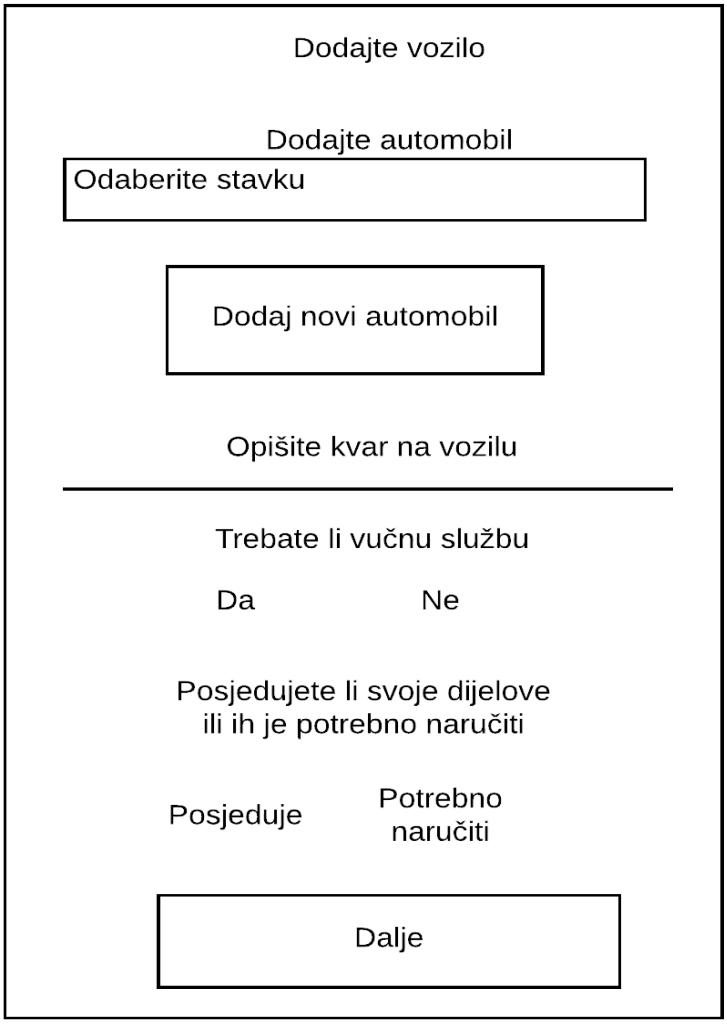
Prvi gumb, gumb „Dogovori termin“, vodi na stranicu (slika 15) koja započinje proces kreiranja sastanka (termina). Na toj stranici korisnik mora odabrati koju vrstu usluge i posla zahtijeva te mora dodati svoje osobne podatke. Ova stranica je sadržajnija nego druge stranice – posjeduje nekoliko različitih elemenata.Osim teksta koji naslovljava stranicu, tu se nalaze još i dva padajuća izbornika gdje je moguć odabir usluge i posla, zatim polja za unos osobnih podataka te stranica sadrži gumb „Dalje“ koji vodi na sljedeću stranicu za dogovaranje termina „Dodaj vozilo“. Stranica za kreiranje novog termina mora biti jasna i intuitivna za korištenje budući da predstavlja prvi korak u kreiranju novog termina korisnika.



Slika 15: Prototip prozora za dodavanje podataka

(Izvor: Izradio autor)

Stranica koja se otvara nakon klika na gumb „Dalje“ u stranici „Dodaj svoje podatke“ (slika 16) naziva se „Dodajte vozilo“. Na stranici dodajte vozilo (slika 16) nalazi se izbornik gdje korisnik može izabrati neki od prethodno dodanih automobila ili može dodati novi automobil klikom na gumb „Dodaj novi automobil“. Postoji još i prostor za dodavanje teksta gdje korisnik opisuje kvar na vozilu te dva upita na koja korisnik daje pozitivan ili negativan odgovor, a odnose se na odabir vučne službe te odabir o naručivanju dijelova.

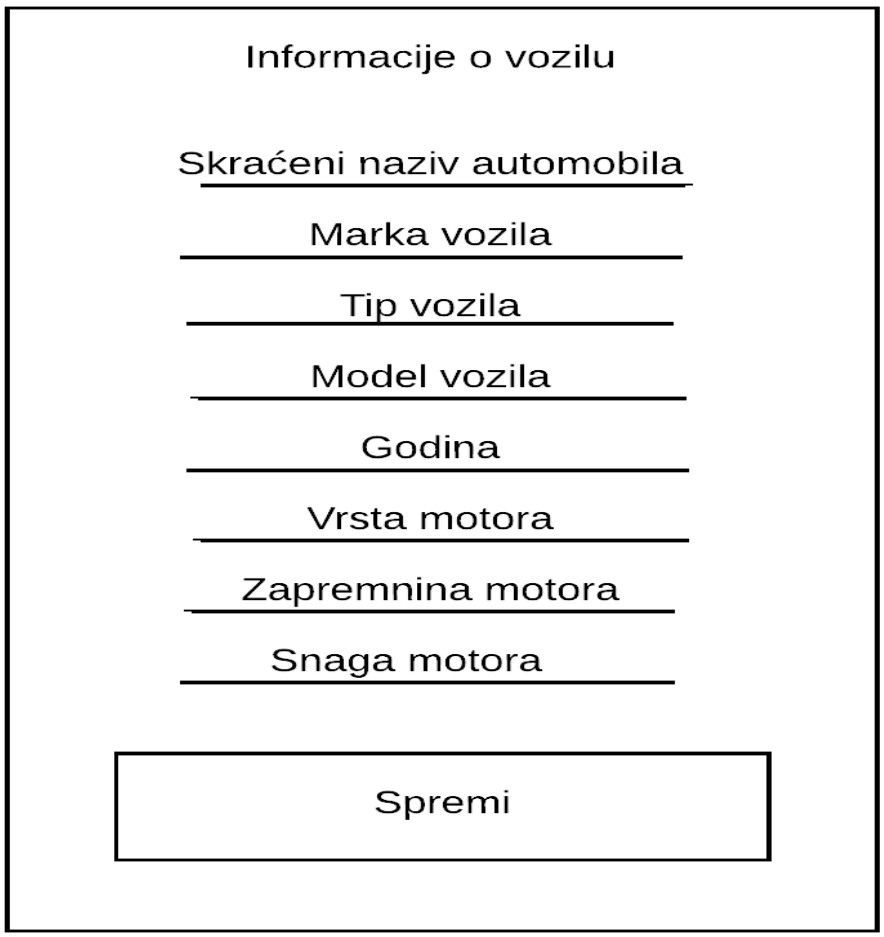


Slika 16: Prototip stranice „Dodajte vozilo“

(Izvor: Izradio autor)

Klikom na gumb „Dodaj novi automobil“ otvara se stranica u obliku fragmenta, a na kojoj se dodaju informacije o vozilu kojeg želimo dodati u sustav. Sve informacije se spremaju u online bazu podataka, a vozilo nakon spremanja možemo pronaći u izborniku „Dodajte automobil“ pod nazivom koji smo upisali pod „Skraćeni naziv vozila“. S obzirom da je ovaj prozor otvoren u obliku fragmenta, on je nešto skraćenih dimenzija i korisnik ne mora prelaziti u novi activity[19] da bi dodao novo vozilo, nego jednostavno ispunjava podatke u fragmentu te mu je nakon dodira na gumb „Spremi“ vozilo dostupno za odabir.

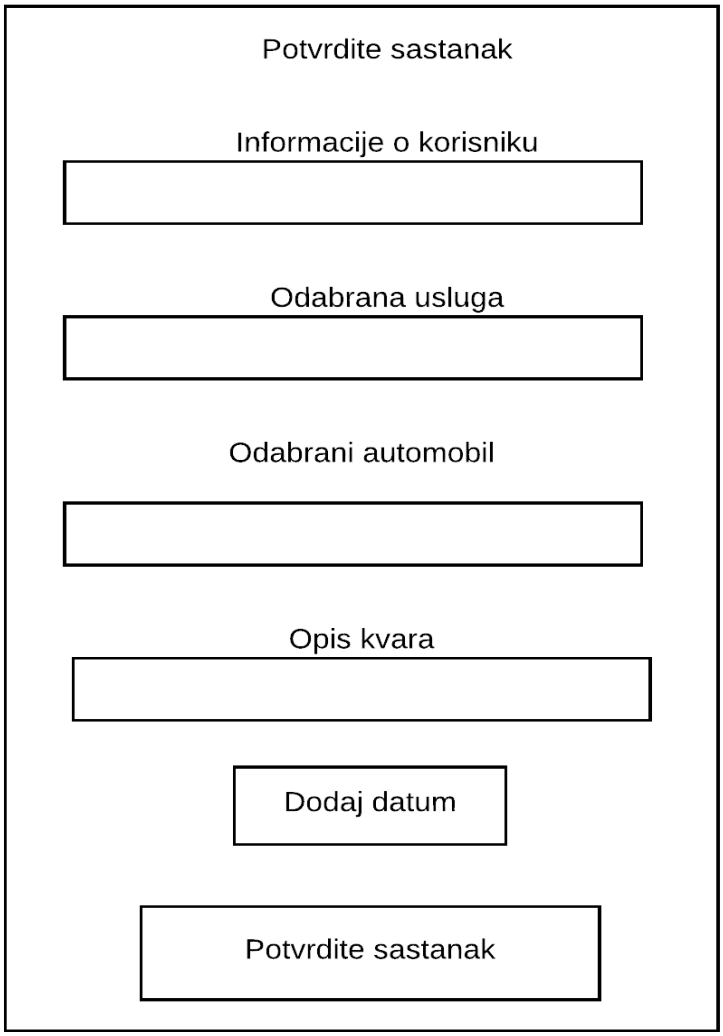
[19] Opisano u poglavlju 4.3.3



Slika 17: Prototip za fragment dodavanja novog vozila

(Izvor: izradio autor)

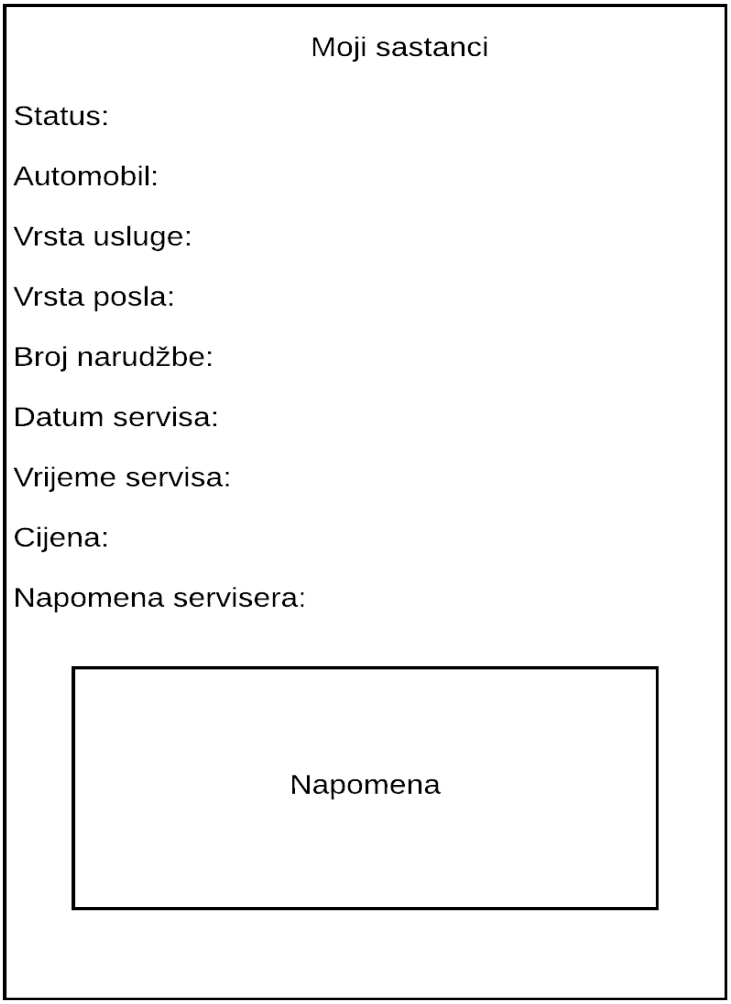
Klikom na gumb „Dalje“ na stranici „Dodajte vozilo“ (slika 17) otvara se nova stranica naziva „Potvrdite sastanak“ (slika 18), a koja sadrži informacije dodane na prethodnim stranicama. Dodir na svaku od informacija nudi mogućnost izmjene istih, a gumb „Potvrdite sastanak“ radi ono što i sam naziv sugerira – potvrđuje sastanak te isti šalje u online bazu podataka koji kasnije revidira autoservis. Prethodne tri stranice grafički prikazuju najvažniju funkcionalnost mobilne aplikacije – kreiranje zahtjeva za sastankom ili pregledom vozila. Svaka od stranica ima obavezna polja koja se moraju ispuniti kako bi se prešlo na sljedeću stranicu. Na koncu, potvrdom sastanka na posljednjoj stranici, zahtjev se šalje u autoservis. Na stranicama se upisuju sve informacije potrebne za brzo i kvalitetno obavljanje servisa, a svaka od informacija olakšava i ubrzava tok konvencionalnog dogovaranja servisa. Primjerice, umjesto telefonskog poziva za provjeru informacija o vozilu, sve informacije o vozilu su sadržane u kreiranom sastanku (nakon dodavanja vozila).



Slika 18: Prototip stranice „Potvrdite sastanak“

(Izvor: Izradio autor)

Dodirom na gumb „Moji sastanci“ unutar glavnog izbornika korisniku se nudi mogućnost odabira već kreiranih sastanaka. Dodirom na jedan od sastanak otvara se stranica naziva „Moji sastanci“ gdje korisnik može dobiti uvid u stanje svog zahtjeva. To uključuje pregled statusa sastanka (u obradi, zaprimljeno, vozilo u tijeku popravka, vozilo spremno za preuzimanje), automobil koji je naveden kao vozilo koje ima kvar, vrsta usluge i posla koje su navedeni prilikom kreiranja sastanka, broj narudžbe, datum i vrijeme servisa te završna cijena. Cijena se utvrđuje nakon što servis zaprimi zahtjev za servisom te nakon konzultacija daje povratnu informaciju o cijeni. Na kraju, korisnik ima uvid u napomenu servisera koju serviser ostavlja na kraju popravka vozila (ukoliko je ista potrebna).

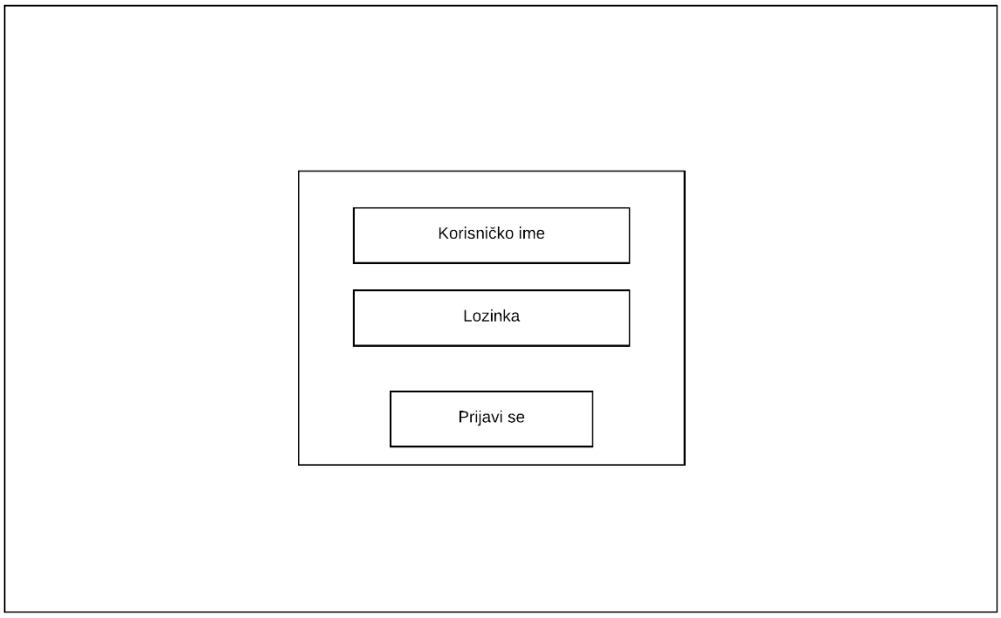


Slika 19: Prototip stranice „Moji sastanci“

(Izvor: izradio autor)

5.1.3. Prototip sučelja web aplikacije

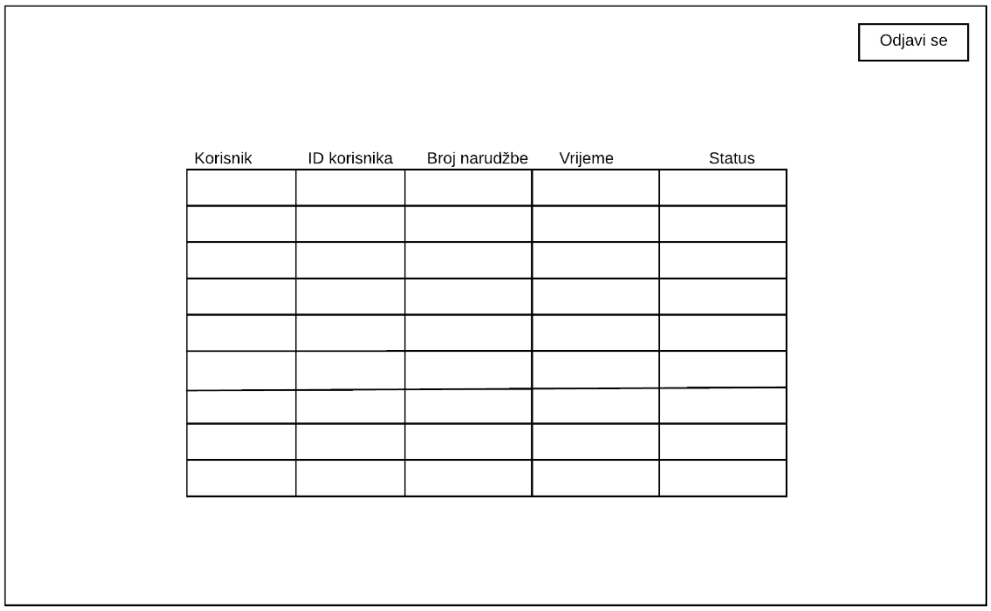
Sučelje web aplikacije zamišljeno je kao jednostavno i minimalističko sučelje. Prva stranica koja se korisniku otvara je stranica za prijavu. Stranica sadrži dva polja za unos teksta u koje je predviđen unos korisničkog imena i lozinke. Kao treći element stranice postavlja se gumb za prijavu koji, očekivano, služi za prijavu korisnika u sustav. Prazan prostor izvan prozora gdje su smješteni gumb i polja za unos trebao bi biti u boji, a sam prozor s gumbom i poljima trebao bi, prema zahtjevima, biti u bijeloj boji. Gumb za prijavu, slijedeći tematski izgled mobilne aplikacije, trebao bi biti crvene boje.



Slika 20: Prototip prozora za prijavu u sustav

(Izvor: izradio autor)

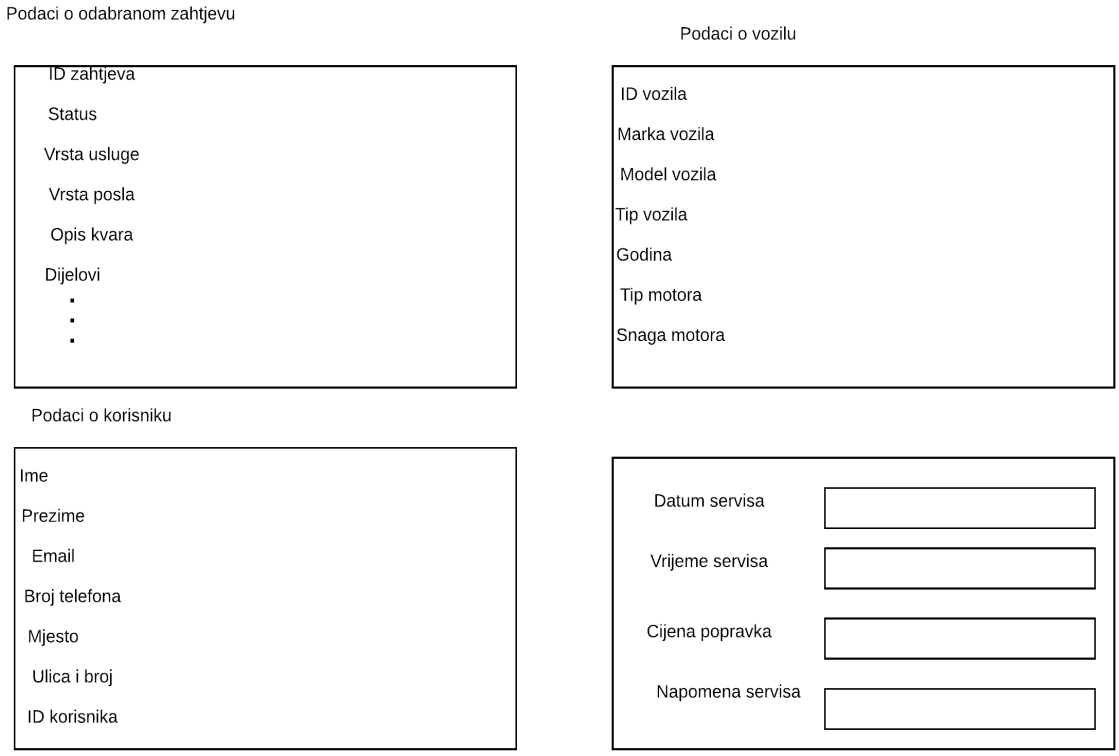
Drugi prozor, prozor za pregled zahtjeva zaprimljenih od korisnika, sastoji se od tablice za prikaz podataka i gumba za odjavu. Unutar tablice nalaze se podaci o novim zahtjevima, a klik na neki od njih vodi na detaljniji pregled.



Slika 21: Prototip prozora za pregled zaprimljenih zahtjeva

(Izvor: Izradio autor)

Treći, ujedno i posljednji, prozor za pregled odabranog zahtjeva osmišljen je na način da prikazuje sve informacije o zahtjevu, sve informacije o vozilu vezanom za taj zahtjev te informacije o korisniku koji podnosi zahtjev. Za kraj je osmišljena forma preko koje će se slati odgovor na zahtjev. Prozor za interakciju sa zahtjevima prikazan je na slici 22.



Slika 22: Prototip prozora za upravljanje zahtjevima
(Izvor: snimka zaslona na računalu autora)

5.2. Razvojni alati za izradu sustava

5.2.1. Microsoft Visual Studio

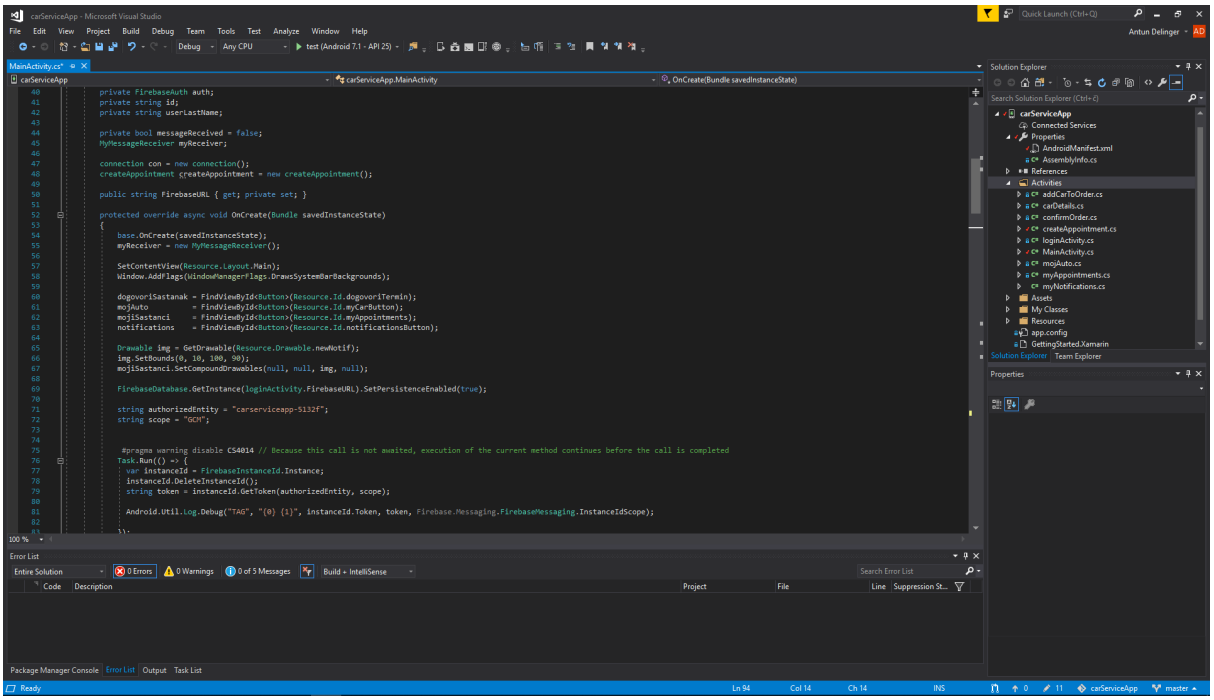
Microsoft Visual Studio jest integrirano razvojno okruženje (IDE) tvrtke Microsoft. Koristi se za razvoj računalnih programa za Windows, web-stranica, aplikacija i usluga. Koristi Microsoftove platforme za razvoj poput aplikativnih programskih interfejsa (API) za Windows, Windows Forms, Windows Presentation Foundation, Windows Store, Microsoft Silverlight itd. Može proizvesti nativni kôd i upravljani kôd (eng. managed code). Visual Studio sadrži uređivač izvornog kôda koji podržava IntelliSense (komponenta koja predlaže ostatak kôda) kao i refaktoriranje kôda. Integrirani program

za otklanjanje grešaka (engl. debugger) radi na nivou izvornog i strojnog koda. Program također sadrži alate poput dizajnera oblika koji se koristi za kreiranje aplikacija s grafičkom korisničkim sučeljem, web-dizajnera, dizajnera klasa i dizajnera shema baza podataka. Prihvata proširenja koja poboljšavaju funkcionalost na skoro svakom nivou dodajući podršku sistema za upravljanje izvornim kôdom i dodajući nove nizove alata poput tekstualnih uređivača i vizualnih dizajnera za jezik određenih domena ili za druge dijelove procesa razvoja softvera (poput klijenta Team Explorer).

Visual Studio podržava različite programske jezike i dozvoljava programeru i sustavu za otklanjanje grešaka da podržavaju (u različitoj mjeri) gotovo bilo koji programski jezik, pod uvjetom da usluga za taj jezik postoji. Ugrađeni jezici su C, C++ i C++/CLI (preko Visual C++), VB.NET (preko Visual Basic .NET)-a, C# (preko Visual C#) i F#. Podrška za ostale programske jezike poput M-a, Pythona, i Rubyja kao i ostalih dostupna je instalacijom jezičkih servisa koji se mogu zasebno instalirati. Također podržava XML/XSLT, HTML/XHTML, JavaScript i CSS.[20] Microsoft Visual Studio postoji u raznim izdanjima, besplatnim i komercijalnim, a koja ovise o načinu uporabe. Za ovaj rad korištena je besplatna inačica Visual Studia – Community Edition. Cijela mobilna aplikacija (grafičko sučelje i aplikacijska logika) izrađeni su u Visual Studio-u.

Microsoft Visual Studio nije primarno namijenjen razvoju Android mobilnih aplikacija pa je stoga odabir ovog alata neobičan, ali uz pomoć alata Xamarin.Android omogućen je razvoj Android mobilnih aplikacija u ovom razvojnom okruženju. Xamarin.Android omogućava korištenje Android biblioteka i standardnih Android klasa te razvijanje grafičkog korisničkog sučelja u .xml datoteci čineći iskustvo razvoja aplikacije približnim onom na alatima primarno namijenjenim za razvoj Android aplikacija kao što je primjerice Android Studio. Primarni razlog, s druge strane, za razvoj aplikacije u ovom okruženju je mogućnost upotrebe C# programskog jezika za razvoj aplikacije s kojim je autor ovog rada već prethodno upoznat te nije bilo potrebe za prilagodbom na drugi programski jezik.

[20] N. Randolph, D. Gardner, M. Minutillo, C. Anderson, Professional Visual Studio 2010, Indianapolis, Wiley Publishing, Inc., 2010., str. 1

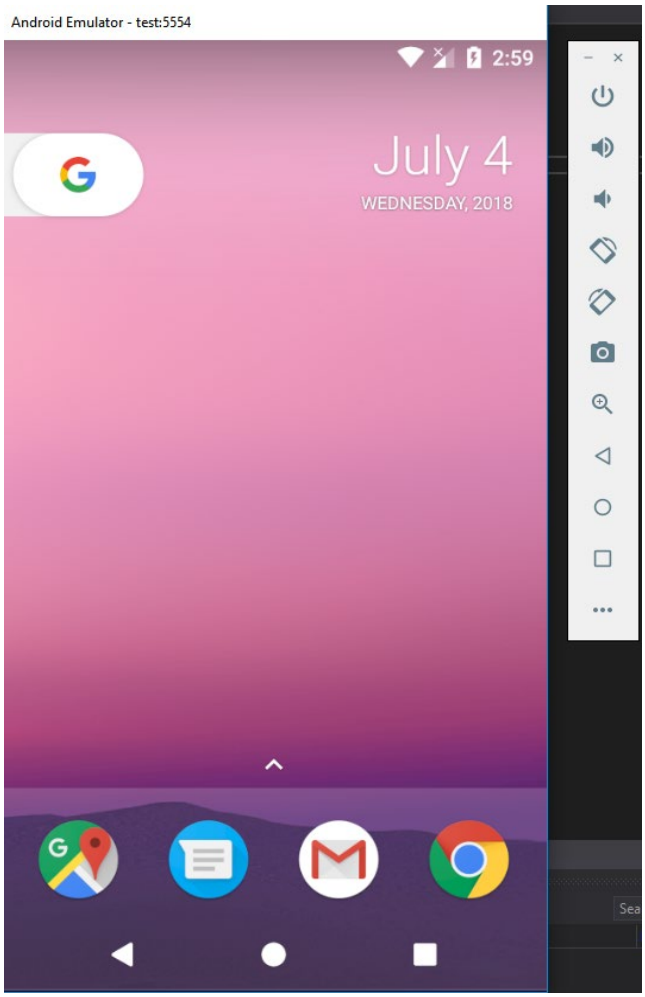


Slika 23: Izgled sučelja programa Microsoft Visual Studio
(Izvor: snimka zaslona na računalu autora)

Prema slici 23 može se vidjeti izgled IDE-a korištenog za izradu sustava. S Desne strane nalazi se izbornik u kojem su smještene svi za projekt relevantni dokumenti i dodaci. Veliki prostor koji dominira prozorom namijenjen je pisanju koda, a prostor u zadnjoj donjoj trećini prozora prikazuje pogreške u kodu, upozorenja i slično.

5.2.2. Android emulator

Android emulator virtualna je simulacija stvarnog mobilnog uređaja. Služi za izradu i testiranje mobilne aplikacije, a u potpunosti simulira ponašanje stvarnog uređaja. Emulator koji je korišten pri izradi mobilne aplikacije je Microsoftov Emulator koji se automatski instalira uz Visual Studio. Prednost korištenja emulatora u odnosu na fizički uređaj je u tome što korištenjem emulatora otvaraju se razne mogućnosti kao što je isprobavanje različitih uređaja koji imaju različite rezolucije, puno je brže i jednostavnije te nema nikakvog rizika od oštećenja uređaja.



Slika 24: Izgled Android Emulatora

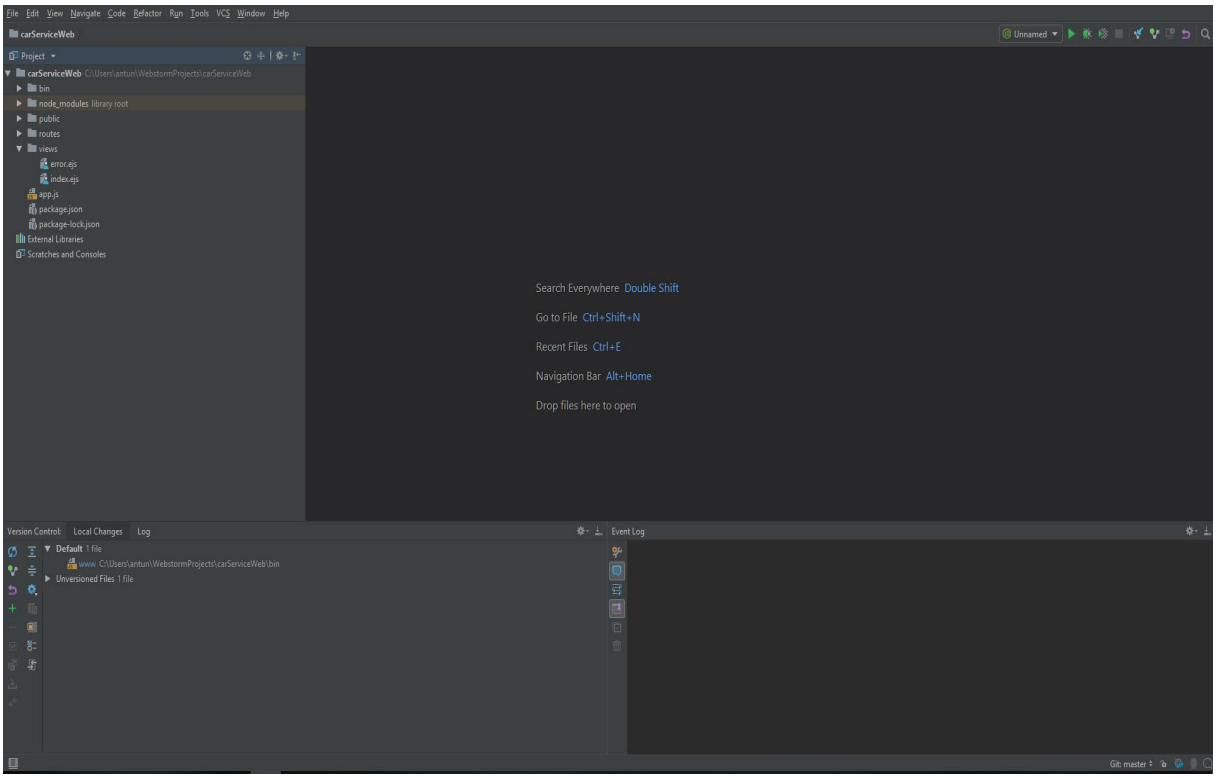
(Izvor: snimka zaslona na računalu autora)

5.2.3. WebStorm

WebStorm je razvojno okruženje tvrtke JetBrains koja nudi nekoliko razvojnih rješenja u svojoj paleti proizvoda, a od kojih je najpoznatiji Intellij IDEA, primarno orijentiran Java programiranju. Web storm pak, s druge strane, okrenut je ka razvoju web aplikacija, posebice s naglaskom na JavaScript programski jezik te Node.js okruženje. WebStorm nudi automatsko prepoznavanje grešaka, automatski dovršetak riječi, prepoznavanje i ispravljanje riječi i druge mogućnosti. Sve mogućnosti odnose se na sve jezike za koje je WebStorm namijenjen – Javascript, Node.js, HTML i CSS. WebStorm također sadrži mogućnost upotrebe nekih od najmodernijih i najpopularnijih frameworka[21] kao što su Angular, React, Vue.js, Express i slično. Ovo razvojno

[21] Termin za gotovu softversku cjelinu koja omogućava programerima brže razvijanje novih rješenja

okruženje sadržava sve ono što bi jedno moćno i sadržajno razvojno okruženje trebalo sadržavati – upravljanje kodom (praćenje, označavanje, podcrtavanje), testiranje i otklanjanje neispravnosti (engl. Debugging). Također, omogućava i korištenje predložaka projekata (primjerice, Express Framework) koji ubrzavaju proces razvoja aplikacije budući da oni izgrađuju temelj projekta umjesto samog korisnika. WebStorm razvojno okruženje odabrano je kao razvojni alat zbog svoje sve veće popularnosti te iznimno dobroj reputaciji kako WebStorma samog, tako i tvrtke JetBrains koja stoji iza razvoja ovog IDE-a. U prilog WebStormu također ide mogućnost besplatnog licenciranja za autore.



Slika 25: Izgled WebStorm razvojnog okruženja

(Izvor: snimka zaslona na računalu autora)

5.3. Android operativni sustav

Riječ android danas je izuzetno popularna i prepoznatljiva riječ, a uglavnom prva asocijacija na riječ android je – mobilni operativni sustav. Iako se radi o (vjerojatno) najpopularnijem operativnom sustavu za mobilne uređaje, android je imao teške početke i nije uvijek bio dobro prihvaćen kao danas. Android je nastao kao tvrtka koju su osnovali Andy Rubin, Chris White, Nick Sears te Rich Miner u 2003. godini. Osnivači androida nastojali su kreirati mobilne uređaje koji su sposobni pamtitiinformacije o

lokaciji te preference svojih korisnika. Na koncu, Google odlučuje kupiti Android tvrtku 2005. godine. Odmah prilikom preuzimanja, Google je osmislio strategiju ulaska na mobilne tržište te krenuo s provođenjem iste. Nakon sklapanja ugovora sa softverskim i hardverskim partnerima kao prvim korakom ka ulasku na mobilno tržište, 2007. nastaje Open Handset Alliance (OHA) – udruženje tvrtki kojima je cilj ubrzanje rasta i razvoja mobilne platforme. Udruženje je predstavljalo ujedinjenost različitih grana – proizvođača mobilnih uređaja, proizvođača slušalica, proizvođača softvera itd. Nedugo nakon osnivanja OHA-a, Google je predstavio svoj prvi proizvod – android. Međutim, sve do 2008. godine nije postojao još niti jedan uređaj s android operativnim sustavom dostupan javnosti. Napokon, nakon 5 godina razvoja androida, u listopadu 2008. godine javnosti je postao dostupan prvi mobilni uređaj dostupan javnosti – HTC G1. Izlazak tog uređaja označio je početak jedne ere. [22] Prema web stranici za statistiku[23] Android operativni sustav (podaci važe za 2017.godinu) drži udio od 76.97% svjetskog tržišta za mobilne uređaje. Vidimo da android sustav ostvaruje veliku dominaciju na tržištu pametnih mobitela, a to može zahvaliti svojom bitnom karakteristikom – velikim rasponom dostupnih uređaja. Naime, android operativni sustav možemo naći na mobitelima svih cjenovnih razreda i raznih tehničkih karakteristika. To omogućuje svakom korisniku da si pronađe točno onakav uređaj kakav mu treba i to je jedan od glavnih razloga tržišne dominacije, ne računajući tu pravovremeni plasman na tržište. Osim što android operativni sustav trenutno uvjerljivo drži drugo mjesto (Apple iOS operativni sustav drži drugo mjesto sa 18%, ostali konkurenti su ispod 1%) ono što ga čini tako bitnim operativnim sustavom je njegova izuzetna tendencija konstantnog rasta. Primjerice, prema već navedenoj stranici za statistiku, 2013. godine android operativni sustav držao je 41.3% tržišta.

U samo 5 godina Android operativni sustav gotovo je udvostručio svoj tržišni udio što dovoljno govori o koliko raširenom operativnom sustavu je riječ. No, nije sve tako idealno glede android sustava. Primjerice, mali broj android uređaja zapravo koristi najnoviju inačicu tog sustava, dok je većina uređaja još uvijek na nekoliko godina starim inačicama. Kao što možemo vidjeti na slici 26 (službeni Google podaci), u 2018. godini (godina pisanja ovog rada) inačica operativnog sustava s najvećim udjelom (25,5%) je Marshamallow (6.0) koja je izdana u 2015. godini, zatim ga slijedi Nougat

[22] Joshua J. Drake, Zach Lanier, Colling Mulliner, Pau Oliva Fora, Stephen A.Ridley, Georg Wicherski, Android Hacker's Handbook, Wiley Inc, 2014 god, str. 2
[23] http://gs.statcounter.com/os-market-share/mobile/worldwide (Pristupljeno 18. lipnja 2018.)

(7.0) koji je Google izdao godinu nakon Marshmallova, a treća inačica po zastupljenosti je Lollipop 5.1 (2014. godina), dok najnoviju verziju (Oreo) ima zanemariv postotak zastupljenosti (<5%).

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.4%
4.1.x	Jelly Bean	16	1.5%
4.2.x		17	2.2%
4.3		18	0.6%
4.4	KitKat	19	10.3%
5.0	Lollipop	21	4.8%
5.1		22	17.6%
6.0	Marshmallow	23	25.5%
7.0	Nougat	24	22.9%
7.1		25	8.2%
8.0	Oreo	26	4.9%
8.1		27	0.8%

Slika 26: Postotak udjela pojedine verzije android sustava

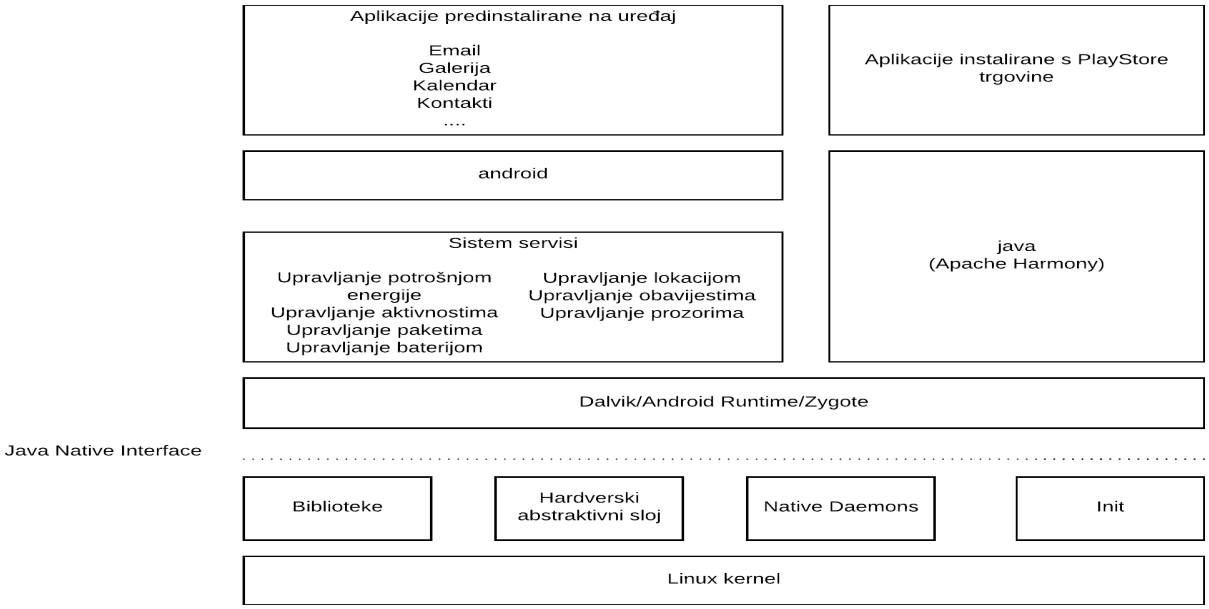
(Izvor: Službena Google statistika, pristupljeno 7. srpnja 2018., https://developer.android.com/about/dashboards/)

5.3.1. Arhitektura android operativnog sustava

Općenito, arhitektura android sustava često se naziva „Java na linuxu“, iako, to je suviše pojednostavljena izjava i nepravedno je prema složenoj arhitekturi kao što je arhitektura android sustava opisivati ju tako jednostavno. Android arhitektura dijeli se na 5 slojeva, uključujući:

- Android aplikacije
- Android framework[24]
- Dalvik virtualni sustav
- Nativni kod korisničkog prostora
- Linux kernel

[24] Framework: Termin za gotovu softversku cjelinu koja omogućava programerima brže razvijanje novih rješenja



Slika 27: Arhitektura android sustava

(Izvor: izradio autor prema Joshua J. Drake, Zach Lanier, Colling Mulliner, Pau Oliva Fora, Stephen A.Ridley, Georg Wicherski, Android Hacker's Handbook, Wiley Inc, 2014 god, str. 26)

Android aplikacije omogućavaju developerima da prošire funkcionalnost android uređaja bez da pristupaju donjim slojevima arhitekture. To im omogućava android framework koji nudi bogat set API-a (Aplication Programming Interface)[25], što omogućava developerima obavljanje uobičajenih zadataka kao što je oblikovanje grafičkog sučelja aplikacije, pristupanje datotečnom sustavu te slanje poruka između dijelova aplikacija. I android aplikacije i android framework razvijeni su u Java programskom jeziku te su izvršeni unutar Dalvik virtualnog sustava, dok Java Native Interface omogućava komunikaciju gornjeg i donjeg sloja arhitekture[26], što je vidljivo na slici 27. Google je razvio Dalvik virtualni stroj koji je efikasniji od Java virtualnog stroja u pogledu korištenja memorije te su napravljene na način da dopuštaju izvođenje više instanci unutar resursa mobilnog uređaja. Kako bi pokrenuli Dalvik virtualni stroj, aplikacijski kod mora biti transformiran iz standardne Java datoteke u Dalvikov.dex format, koji ostavlja 50% manji memorijski otisak od Java bytecodea. Dalvik je odnedavno zamijenjen ART sustavom. Razlika između Dalvik VM-a i ART-a je u kompiliranju. Dalvik VM je koristio JIT način kompiliranja koji je kod prevodio pri izvođenju i korisnikovim napredovanjem kroz aplikaciju dodatni kod se prevodi u strojni jezik. ART sustav se kompilira pri instaliranju tako da sprječava zastoj programa.

[25] Gotova softverska rješenja za ciljane slučajeve
[26] J.J. Drake et al., Op.cit. str. 26

Za izradu sustava za koji je vezan ovaj rad najvažnije su aplikacijska komponenta android operativnog sustava, a ona uključuje: android manifest, aktivnosti (engl. Activity), namjere (engl. Intent), usluge (engl. Services), Broadcast Receiver te pružatelje sadržaja (engl. Content Provider).

5.3.2. Android manifest

Sve android aplikacije moraju sadržavati AndroidManifest.xml datoteku koja sadrži informacije o aplikaciji, kao što su:

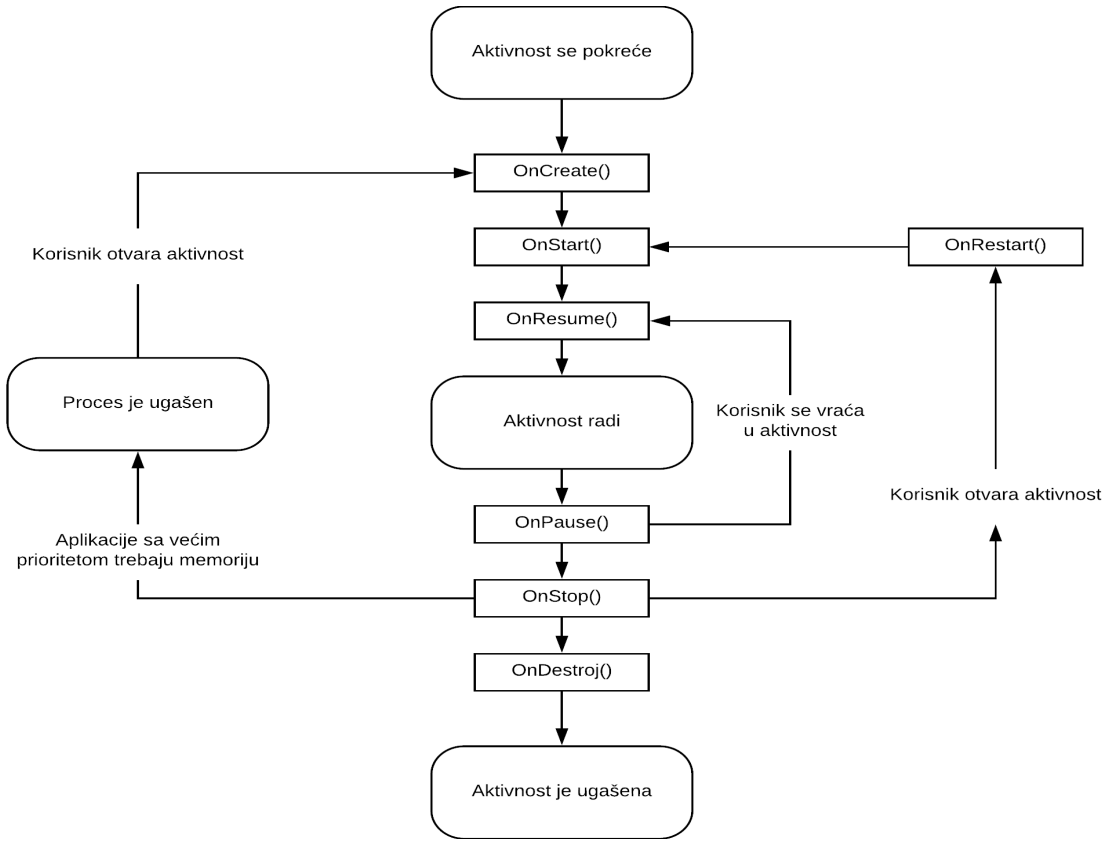
- Unikatni naziv paketa (npr. com.name.someapp) te informacije o verziji
- Aktivnosti, servise, BroadcastReceivere
- Dozvole (pravila) aplikacije
- Informacije o vanjskim bibliotekama i paketima koji su korišteni u razvoju aplikacije

Android manifest ima određena pravila kojih se potrebno pridržavati, primjerice jedina dva elementa koja su obavezna i koji se mogu samo jednom pojaviti su „<manifest>“ i „<application>“.Većina drugih elemenata se može pojaviti i više puta. Element u android manifest datoteci može sadržavati samo drugi element. Elementi koji su na istom rangu unutar manifest datoteke nisu složeni po redu. Manifest datoteka je uglavnom automatski kreirana od strane razvojnog okruženja (IDE-a). Android manifest je također važan zbog toga što se u njemu moraju dodati dozvole koje će aplikacija koristiti u svom radu. To se odnosi na primjerice korištenje telefonskog imenika, kamere, sms aplikacije i slično. Da bi se dodala dozvola potrebno je koristiti „<uses-permission>“ element.

5.3.3. Aktivnosti

Jednostavno rečeno, aktivnost je aplikacijska komponenta koju korisnik aplikacije može vidjeti. Kreirana je pomoću Activity klase, a sastoji se od prozora aplikacije i pripadajućih elemenata grafičkog sučelja. Niži sloj menadžmenta aktivnosti upravljan je sa prikladno nazvanim servisom – menadžerom aktivnosti (engl. Activity Manager) koji ujedno obrađuje i namjere koje za zadatak imaju pokretanje različitih aktivnosti.

Aktivnost je povezana sa XML datotekom u kojoj su definirani elementi sučelja i na taj način korisniku je prikazan određeni prozor u aplikaciji.



Slika 28: Životni ciklus aktivnosti u android operativnom sustavu

(Izvor: izradio autor prema P. K. Dixit, Android, Odisha, India, Vikas Publishing House, 2014, str. 128)

Upravljanje aktivnostima nije jednostavno s obzirom da korisnik za vrijeme normalnog korištenja aplikacije otvara aktivnosti jednu za drugom i vraća se na prethodno otvorene aktivnosti. Zato i postoji menadžer aktivnosti koji smo gore spomenuli, a koji se brine o životnom ciklusom aktivnosti. Štoviše, aktivnost se, nakon prvog pokretanja, stavlja na čekanje dok je neka druga aktivnost aktivna, a nakon što se ta ista aktivnost koja je bila na čekanju ponovno pozove, ona se vraća u uporabu. Na taj način se bitno ubrzava rad aplikacije. Aktivnosti prestaju s radom i potpuno se brišu iz memorije tek nakon što dugo vremena nisu u uporabi.

Developer može upravljati ovim ciklusom podataka tako što dodaje izmjene na sljedeće metode(naredbe):

- onCreate() – poziva se prilikom kreiranja aktivnosti
- onStart() - poziva se prilikom početka aktivnosti

- onResume() – poziva se prilikom početka ili nastavka aktivnosti
- onPause() - poziva se nakon pauziranja aktivnosti (pozadinski način rada)
- onStop() – poziva se nakon zaustavljanja aktivnosti
- onRestart() – poziva se nakon ponovnog pokretanja aktivnosti
- onDestroy() – poziva se netom prije zaustavljanja aplikacije
- onSaveInstanceState() – poziva se ukoliko je potrebno očuvanje stanja
- onRestoreInstanceState() – poziva kod prvog pokretanja aktivnosti nakon što je stanje već spremljeno s onSaveInstanceStace() metodom

5.3.4. Servisi

Servisi su aplikacijske komponente koje se izvode u pozadini te su bez korisničkog sučelja. Primjerice, servis može pokretati pozadinsku glazbu ili dohvaćati podatke s interneta. Neki od primjera servisa su SmsReceiverService i BluetoothOppService. [27] Pojavljuje se u dva oblika:

- Service – komponenta je započeta u trenutku kada je pozvana pomoću „startService()“ metode. Sama se zaustavlja nakon što je izvršena.
- Bound – postaje vezana kada se pozove metoda „bindService()“ koja nudi komunikaciju između komponenti i servisa. Aktivna je samo dok je druga aplikacijska komponenta vezana uz nju.

Dakle, servis tipično može biti zaustavljen, pokrenut ili vezan, ovisno o namjerama. Aplikacijske komponente pokreću servis na isti način na koji koristi i aktivnosti – preko intent (namjera) komponente.

5.3.5. Broadcast Receiver

Broadcast receiver je komponenta koja reagira i identificira promijene u radu sustava. Primjerice, ukoliko korisnik ugasi ekran na uređaju, aplikacija to može detektirati preko broadcast receivera. Programeri ovu komponentu koriste ukoliko žele

[27] J. J. Drake et al., op.cit., str. 38

programirati određeno ponašanje aplikacije ovisno o određenim promjenama u radu sustava.

Postoje dvije vrste broadcast receivera[28]:

- Normalni – svi primatelji primaju poruke u isto vrijeme (ili različito, ali neodređenim redoslijedom)
- Naručeni – primatelji primaju poruke jedan po jedan po unaprijed određenom redoslijedu koji je određen unutar android manifest datoteke kroz atribut „android:priority“.

5.3.6. Namjere

Namjere (engl. Intents) predstavljaju ključnu komponentu komunikacije unutar aplikacije. Namjere su objekti koji sadržavaju informacije o operaciji koja će se izvršiti te su te informacije relevantne za primatelja. Gotovo svaka akcija unutar aplikacije – dolazak obavijesti, promjena prozora i slično – uključuje korištenje namjera. Namjere, osim što sadrže informaciju o tome što će se sljedeće dogoditi, sadržavaju i dodatne informacije koje korisnik želi proslijediti nekoj aktivnosti. Te informacije mogu biti raznog oblika (booean, integer, string…), a dodati se mogu kroz metodu „addExtra()“.

Namjera može biti eksplicitna i implicitna, ovisno o određenosti primatelja. U slučaju eksplicitne namjere primatelj je jasno definiran, dok to nije slučaj sa implicitnom namjerom gdje je definiran jedino tip primatelja, ali ne i točno koji primatelj.

[28] loc.cit.

5.3.7. Pružatelji sadržaja

Pružatelji sadržaja (engl. Content Providers) ponašaju se kao strukturirano sučelje između aplikacija, odnosno – drugim riječima – obnašaju funkciju razmjene podataka između aplikacija. Primjerice, pružatelji aplikacije kontakti i aplikacije kalendar upravljaju centralnim repozitorijem kontakta i kalendara, a kojemu mogu pristupiti i druge aplikacije (uz prikladna dopuštenja). Jednostavnijim riječima, podaci aplikacija su odvojeni te im mogu pristupati druge, nevezane aplikacije kao što je recimo slučaj sa aplikacijom kontakti gdje kontakte koje koristi ta aplikacija može koristiti i neka druga, nezavisna aplikacija. Pružatelji usluga koriste metode kao što su „Insert()“, „Update()“, „Delete()“ i slično, a vrlo često se koriste za rad s bazama podataka. Kao i u slučaju ostalih aplikativnih komponenti, i pružatelji usluga mogu biti ograničeni sa dozvolama u manifest datoteci.

5.3.8. Android Framework

Predstavlja „ljepilo“ ili sponu između aplikacija i vremena izvršavanja (engl. runtime) te korisniku nudi pakete i pripadajuće klase za izvršavanje uobičajenih zadataka. To uključuje upravljanje i sređivanje elemenata korisničkog sučelja, pristupanje dijeljenim podacima te prosljeđivanje poruka između aplikacijskih komponenti. Drugim riječima, Android Framework omogućava da se aplikacija kreira uz pomoć komponenata koje su već napravljene, dakle ponovnom upotrebom gotovog koda. Uobičajeni paketi koje Framework nudi su oni koji počinju sa android.* (npr. android.content). Osim njih, android nudi i razne java Java klase (nalaze se u .Java* imenskom prostoru). Android Framework uključuje i servise za upravljanje funkcionalnostima klasa koje nudi. Tako android Framework nudi sljedeće servise:

- Menadžer aktivnosti – upravljanje namjerama i aktivnostima
- View sustav – upravljanje elementima sučelja koje korisnik može vidjeti
- Menadžer paketa – upravljanje paketima klasa koje možemo instalirati
- Menadžer telefonskih usluga – upravlja servisima i informacijama vezane uz telefon i radio usluge
- Menadžer resursa – upravljanje resursima nevezanim za kod aplikacije, kao što su grafički elementi
- Lokacijski menadžer – upravljanje lokacijskim informacijama (GPS, WiFi)
- Menadžer obavijesti – upravljanje obavijestima (zvukovi, vibracije…)

5.4. Android aplikacija

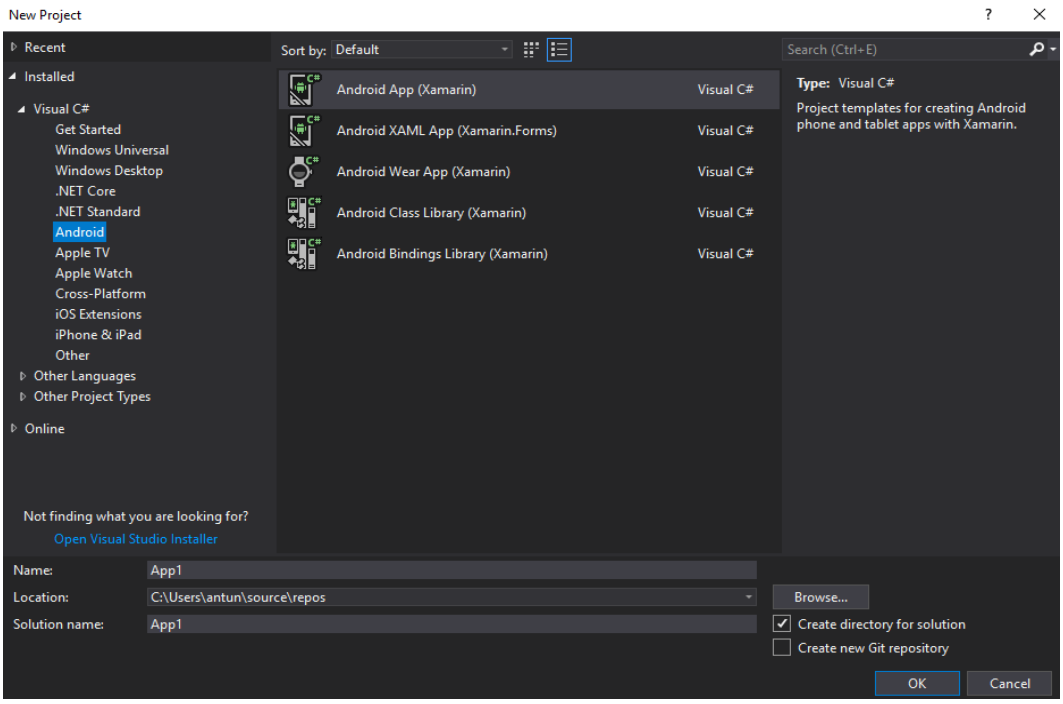
Android aplikacija zamišljena je kao client-side aplikacija, odnosno bila bi prezentirana krajnjem korisniku. Kao što je već utvrđeno zahtjevima, aplikacija bi služila za olakšavanje rada u autoservisu, kako za autoservis sam tako i za klijente tog autoservisa. Nadalje, aplikacija bi zahtijevala registraciju e-mail adresom kako bi se mogla koristiti. Nakon same registracije, korisnik može poslati zahtjev za dolaskom na servis unutar kojeg bi poslao sve informacije relevantne za popravak tog vozila, kao i sam tip popravka. Nakon poslanog zahtjeva, autoservis nakon pregleda zahtjeva daje povratnu informaciju o raspoloživom terminu, cijeni popravka i slično. Aplikacija je izrađena pomoću C# programskog jezika, u razvojnom okruženju Visual Studio pomoću xamarin.android rješenja koje omogućuje korištenje android SDK-a(Software Development Kit) sa C# programskim jezikom.

5.4.1. C# programski jezik

C# je Microsoftov jednostavan i moderan objektno – orijentiran programski jezik. Korijeni C# programskog jezika protežu se iz skupine C programskih jezika (C, C++), a posebice je sličan Java programskom jeziku. C#, osim što je objektno orijentiran, također je i komponentno – orijentiran programski jezik. Ključno u ovom pristupu je što te komponente predstavljaju programerski model sa svojstvima (engl. properties), metodama (engl. methods) i događajima (engl. events). C# ima unificiran sustav pisanja. Svi tipovi u C#, uključujući i primitivne tipove kao što su int i double, nasljeđeni su iz istog korijena – tipa objekt. Prva verzija ovog programskog jezika razvijena je u veljači 2002. godine, a s ciljem omogućavanje .NET platformi dostizanje maksimuma svojih mogućnosti. Autor rada prethodno je upoznat sa C# programskim jezikom te je stoga C# dobio prednost u odnosu na učestalije jezike u razvoju Android aplikacija kao što su Java ili Kotlin.

5.4.2. Izrada mobilne aplikacije

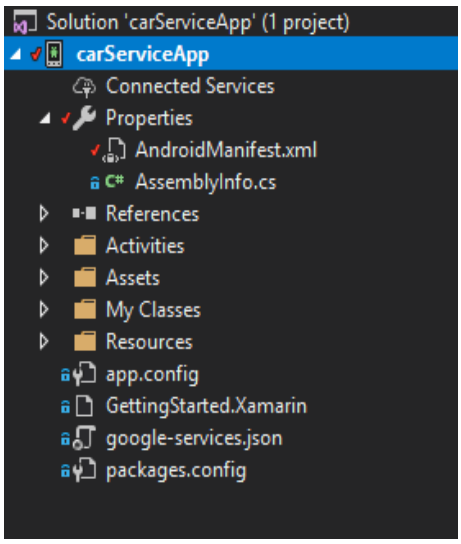
Prvi korak prilikom izrade mobilne aplikacije je kreiranje novog projekta u razvojnom okruženju. Potrebno je u izborniku pod „Visual C#“ odabrati „Android“ te zatim odabrati „Android app“.



Slika 29: Kreiranje novog projekta

(Izvor: snimka zaslona na računalu autora)

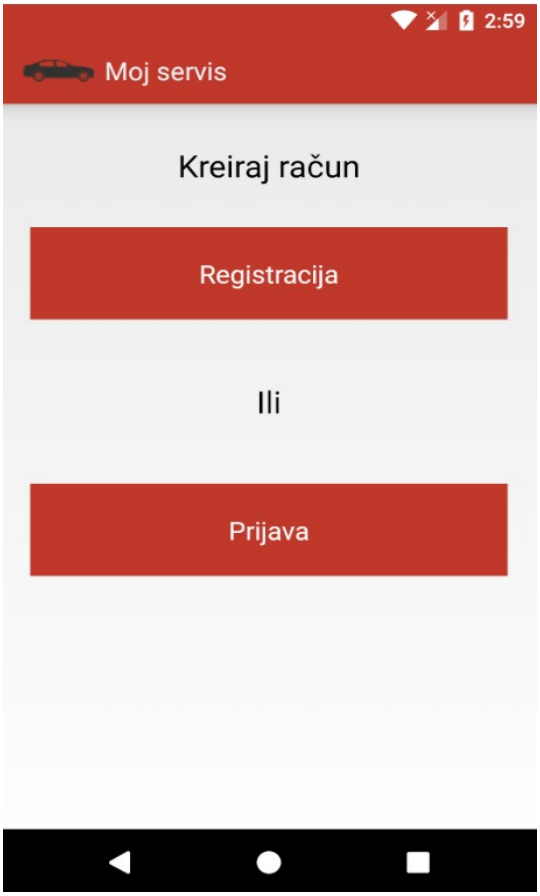
Nakon kreiranja novog projekta, otvara nam se taj projekt i mogućnost dodavanja komponenata u isti.



Slika 30: Preglednik projekta

(Izvor: snimka zaslona na računalu autora)

Na slici 30 prikazan je preglednik projekta u kojemu se nalaze folderi s aktivnostima, resursima i vlastitim klasama. Također, preko preglednika je moguće pristupiti manifest datoteci te ju uređivati. Aplikacija „moj servis“ sastoji se od devet aktivnosti – zaslon za prijavu, početni zaslon, kreiranje sastanka, nastavak kreiranja sastanka, potvrda sastanka, detalji vozila, moji sastanci i obavijesti. Svaka od tih aktivnosti upravlja nekim od ključnih funkcionalnosti aplikacije. Ostale funkcionalnosti koje zahtijevaju možda manje prostora ili manju funkcionalnost, izvedene su kroz fragmente. Aplikacija sadrži deset fragmenata. Osim toga, aplikacija sadrži još četrnaest klasa koje su korištene za bazu podataka, Firebase Cloud Messaging ili za kreiranje vlastitih listi za pregled stavki.



Slika 31: Zaslon za prijavu u aplikaciju

(Izvor: snimka zaslona na računalu autora)

Na slici 31 nalazi se zaslon za prijavu. Kao što vidimo, sučelje je jednostavno – prevladavaju bijela i crvena boja. Gore se nalazi ikona aplikacije i naziv aplikacije. Zaslon ima dva gumba – registracija i prijava. S ovim zaslonom povezana je loginLayout.axml datoteka u kojoj su programirani dijelovi sučelja koji su vidljivi korisniku. Za zaslon za prijavu upotrijebljen je Linear Layout pomoću kojeg se elementi

sučelja dodaju vertikalnom „top – down“ metodom, odnosno elementi se slažu od vrha prema dolje. Tekstualni elementi sučelja, kao što je „Kreiraj račun“ na slici 31, dodaju se na sljedeći način:

```
1.      < TextView android: textAppearance = "?android:textAppearanceLarge"
2.      android: gravity = "center"
3.      android: layout_width = "fill_parent"
4.      android: layout_height = "0.0dip"
5.      android: text = "Kreiraj račun"
6.      android: layout_weight = "20.0" / >
```

Osim što prozor prikazan na slici 31 prikazuje tekst i gumbe, nakon klika na „prijava“ ili na „registracija“, također imamo i element traka za napredak (engl. Progress bar) koji, kao što naziv sugerira, korisniku pokazuje napredak prilikom prijave ili da se nešto događa u pozadini. Gumbi i element traka za napredak su dodani na sljedeći način:

```
1.      < Button android: text = "Prijava"
2.      android: layout_width = "match_parent"
3.      android: layout_height = "0.0dip"
4.      android: layout_weight = "15.0"
5.      android: id = "@+id/Loginbutton"
6.      android: layout_marginLeft = "20.0dip"
7.      android: layout_marginRight = "20.0dip"
8.      android: background = "#c0392b"
9.      android: textColor = "#FFFFFF" / >
10.     < ProgressBar android: layout_width = "match_parent"
11.     android: layout_height = "wrap_content"
12.     android: id = "@+id/progressBar1"
13.     android: layout_gravity = "center" / >
```

Datoteka „loginLayout.axml“ sama po sebi nema nikakvog smisla budući da ona samo prikazuje elemente sučelja. Za upravljanje logikom potrebna nam je aktivnost, odnosno datoteka „loginActivity.cs“ u kojoj je sadržana aplikativna logika, odnosno – ponašanje aplikacije. Povezivanje između te dvije datoteke obavlja se pozivanjem sljedeće metode koja se mora dodati u svaku aktivnost:

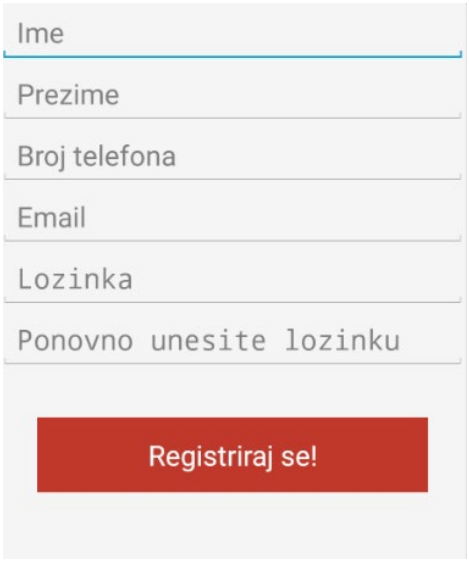
```
1.      SetContentView(Resource.Layout.logInLayout);
```

Elementi dodani u .axml datoteci povezuju se s trenutnom aktivnosti uz pomoć „FindViewById<>()“ metode. No, prije toga moramo kreirati lokalnu varijablu tog tipa, odnosno objekt tipa „Button“ primjerice.

```
1.      private Button buttonSignIn;
2.      buttonSignIn = FindViewById < Button > (Resource.Id.Loginbutton);
```

Klikom na „Registracija“ otvara se fragment u kojem se korisniku nude polja za upis teksta preko kojeg će upisati svoje podatke te lozinku. Otvaranje fragmenta događa se nakon klika na gumb „Registracija“, a pomoću eventa buttonClick:

```
1.      private void MButtuonSignUp_Click(object sender, EventArgs e) {
2.          FragmentTransaction transaction = FragmentManager.BeginTransaction();
3.          signUpDialog dialog = new signUpDialog();
4.          dialog.Show(transaction, "dialog fragment");
5.      }
```



Slika 32: Fragment „Registriraj se“

(Izvor: snimka zaslona na računalu autora)

Fragment predstavlja dio korisničkog sučelja (manji prozor od aktivnosti). Moguće je kombinirati više fragmenata u jednoj aktivnosti te koristiti jedan fragment u više različitih aktivnosti. Dakle, fragment je modularna sekcija aktivnosti, koja ima svoj vlastiti životni ciklus, može primati dolazne informacije te ju je moguće pokretati i zatvarati dok je aktivnost još pokrenuta – to je, dakle, nešto kao pod-aktivnost.[29]

Klikom na „Registriraj se“, prvo se provjerava jesu li sva polja u fragmentu ispunjena, ukoliko nisu korisnik je upozoren sa „Sva polja moraju biti ispunjena“ porukom te zatim nakon toga provjerava se podudaraju li se lozinke unesene u dva za to predviđena polja. Ukoliko se ne podudaraju, korisnik je obaviješten porukom „Lozinke se ne podudaraju“. Ukoliko je obje provjere prođu bez potrebe za interakcijom aplikacije, podiže se događaj (engl. Event) koji prikuplja dodane informacije, zatvara fragment i

[29] P.K. Dixit, Op.cit., str. 43

predaje ih natrag u aktivnost „loginActivity.cs“. Aktivnost „loginActivity“ prethodno je pretplaćena na događaj klika na gumb „Registracija“. Nakon preuzimanja informacija od fragmenta za registraciju, poziva se metoda „CreateUserWithEmailAndPassword()“ gdje se kao parametar šalje korisnikov email i lozinka te Firebase kreira novog korisnika u svojoj bazi podataka za autentifikaciju korisnika. Metoda za kreiranje novog korisnika nalazi se u klasi „FirebaseAuth“ koju je potrebno instancirati te je instanca te klase nazvana „auth“.

```
1.      private void Dialog_onSignUpComplete(object sender, onSignUpEventArgs e) {
2.          signup_inputName = e.firstName;
3.          signup_inputLastName = e.lastName;
4.          signup_inputPhoneNumber = e.phone;
5.          signup_inputEmail = e.email;
6.          signup_inputPassword = e.password;
7.          auth.CreateUserWithEmailAndPassword(signup_inputEmail, signup_inputPassword).Add
OnCompleteListener(this, this);
8.          progressBar.Visibility = ViewStates.Visible;
9.      }
```

Iz koda priloženog iznad vidljiva je pretplata na događaj „onSignUpComplete“ što se odnosi na završetak procesa registracije kojeg korisnik mora obaviti, a koji završava s klikom na gumb „Registriraj se!“. Linije koda od broja 2 do broja 6 odnose se na preuzimanje informacija iz događaja te pohranjivanje u (privatne) lokalne varijable, a linija broj 7 predstavlja pozivanje metode za registraciju korisnika na koju su dodani slušači događaja (engl. event listener) „AddOnCompleteListener() koji reagiraju na određeni događaj, u ovom slučaju reagirat će na završen proces registracije. Na kraju, zadnja linija koda (8) odnosi se na „progressBar“ kojemu se daje naredba kako treba postati vidljiv, odnosno kako bi se trebao vrtjeti sve dok mu nije zadano drugačije.

```
1.      public void OnComplete(Task task) {
2.          if (task.IsSuccessful) {
3.              Toast.MakeText(this, "Uspješno ste se registrirali. Možete se prijaviti.", T
oastLength.Long).Show();
4.              FirebaseUser user = FirebaseAuth.GetInstance(app).CurrentUser;
5.              id = user.Uid;
6.              CreateUser();
7.              buttonSignIn.PerformClick();
8.              progressBar.Visibility = ViewStates.Invisible;
9.          } else {
10.             Toast.MakeText(this, "Neuspješna registracija. Možda postoji problem sa vezo
m ili korisnik sa dodanim emailom već postoji. ", ToastLength.Long).Show();
11.             progressBar.Visibility = ViewStates.Invisible;
12.         }
13.     }
```

Nakon završetka procesa registracije, slušač događaja poziva metodu „OnComplete(Task)“ metodu koja prima parametar tipa „Task“. Prvi korak u kodu gore je provjera je li zadatak uspješno izvršen, a što se postiže pod brojem s kodom pod broj 2. Ukoliko je uvjet („ako je zadatak uspješno izvršen“) ispunjen, izvršava se kod u kojemu se uzima instanca trenutnog korisnika i njegov unikatni ključ (Uid) koji će poslužiti prilikom kreacije korisnika u Firebase bazi podatatak, a što se obavlja pozivom vlastite metode „createUser()“. Nakon što je pozvana metoda „CreateUser()“, poziva se metoda „performClick()“ koja će simulirati klik korisnika na gumb „Prijavi se“ kako bi se odmah otvorio fragment za prijavu korisnika bez da korisnik to sam mora obaviti. Nadalje, isključuje se vidljivost „progressBar“ kotačića što indicira završetak procesa registracije. U slučaju da je zadatak u početku bio neuspješan, preskače se taj dio koda i izvršava donji dio koda gdje korisnik prima obavijest kako registracija nije uspješna i kako bi ju trebao pokušati ponoviti. To se obavlja već postojećom metodom „makeText()“ koja se nalazi u klasi „Toast“. Prije nego pređemo na prikaz procesa prijave, slijedi prikaz kreiranja korisnika u bazi podataka. Naime, korisnik je registriran u Firebase bazi podataka za autentifikaciju, no nije i kreiran u Firebase NOSQL bazu podataka. To se izvršava pomoću gore spomenute metode „CreateUser()“, a kod izgleda ovako:

```
1.      User OfflineUser = new User();
2.      OfflineUser.uid = id;
3.      OfflineUser.name = signup_inputName;
4.      OfflineUser.lastName = signup_inputLastName;
5.      OfflineUser.email = signup_inputEmail;
6.      OfflineUser.phone = signup_inputPhoneNumber;
7.
8.      Account user = new Account();
9.      user.uid = id;
10.     user.name = signup_inputName;
11.     user.lastName = signup_inputLastName;
12.     user.email = signup_inputEmail;
13.     user.phone = signup_inputPhoneNumber;
14.     user.city = "";
15.     user.adress = "";
16.
17.     con.db.CreateTable < User > ();
18.     con.db.Insert(OfflineUser);
19.
20.     var firebase = new FirebaseClient(FirebaseURL);
21.     var item = firebase.Child("users").Child(id).PostAsync < Account > (user);
22.     }
```

Kako aplikacija koristi dva sustava baze podataka – SQLite i Google Firebase baze podataka, korisnik je u ovoj metodi kreiran u obje baze. Prva dva dijela koda odnose se na kreiranje objekt tipa „User“, osnosno „Account“ gdje je prvi objekt namijenjen za

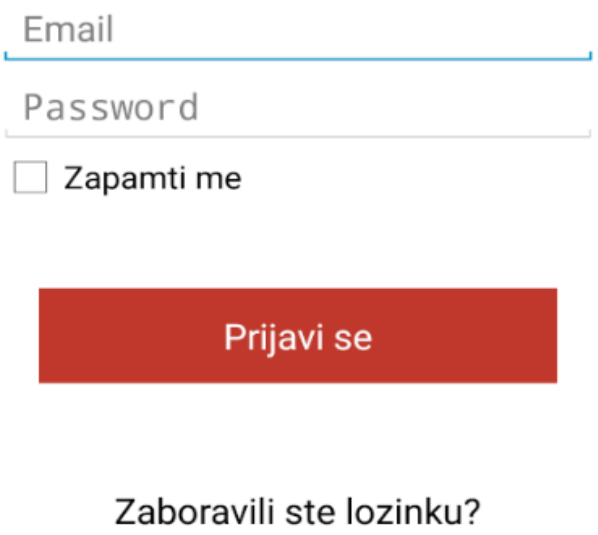
SQLite bazu podataka, a drugi za Firebase bazu podataka. Različiti objekti su kreirani iz razloga što postoje elementarne razlike u radu baza podataka. Također, jedan od razloga korištenja SQLite baze podataka je i korištenje iste prilikom algoritma za memoriranje korisnika pri prijavi. Korisnik, dakle, označava kvačicu „Zapamti me“, a u offline bazi podataka (SQLite) taj podatak se zapisuje. Takva praksa je brža i u slučaju pisanja i čitanja iz baze podataka u odnosu na Online bazu podataka. Iako postoji mogućnost Offline korištenja Firebase baze podataka, zbog nekompatibilnosti ista nije iskorištena u radu ove aplikacije pa je SQLite poslužio kao vatrogasno rješenje.

U gornjem kodu pod broj 17 i 18 upisan je korisnik u SQLite bazu podataka, naredbom „con.db.CreateTable<User>()“ te „con.db.Insert(OfflineUser)“ od kojih prva predstavlja kreiranje tablice tipa objekt „User“, a druga predstavlja dodavanje novog retka u tablici s podacima instance tog istog objekta „OfflineUser“. „con.db“ predstavlja instancu vlastite „Connection.cs“ klase. Posljednjih nekoliko redaka u gornjem kodu predstavljaju dodavanje korisnika u Firebase bazu podataka, a što se obavlja „PostAsync<>()“ metodom koja kreira korisnika i automatski mu dodjeljuje unikatni ključ.

Klasa korisnika za bazu podataka ima 7 svojstava (engl. Property) od kojih svaki predstavlja jednu kategoriju u bazi podataka (ime, prezime, email, broj itd.) izgleda ovako:

```
1.      public class Account {
2.          public string uid      {get;set;}
3.          public string name     {get;set;}
4.          public string lastName {get;set;}
5.          public string email    {get;set;}
6.          public string phone    {get;set;}
7.          public string city     {get;set;}
8.          public string adress   {get;set;}
9.      }
```

Kao što je gore navedeno, nakon završetka registracije korisnika, automatski se otvara fragment za prijavu korisnika. Na slici 33 prikazan je izgled fragmenta za prijavu korisnika. Postoje dva polja za dodavanje tekstualnog sadržaja, polje za dodavanje kvačice za memoriranje korisnika, gumb za prijavu te klikabilno tekstualno polje „Zaboravili ste lozinku?“.



Slika 33: Fragment za prijavu korisnika

(Izvor: snimka zaslona na računalu autora)

Fragment za prijavu korisnika programiran je slično kao i fragment za registraciju korisnika. Što se tiče osnovne funkcionalnosti – prijave korisnika – ono sadrži iste elemente kao fragment za registraciju: događaj koji registrira dodir na gumb „Prijavi se“, provjeru ispravnosti dodanih podataka, događaj koji registrira završetak procesa i predaju podataka u aktivnost „loginActivity“. Zatim, nakon povratka u početnu aktivnost, preuzimaju se podaci, predaju lokalnim varijablama, poziva se metoda „SignInWithEmailAndPassword()“ koja prima parametre tipa email i lozinka, uspoređuje ih s podacima u bazi podataka te na osnovu toga odobrava ili odbija prijavu korisnika. Na prethodno spomenutu metodu dodaje se slušač događaja „AddOnSuccesListener()“ i „AddOnFailureListener()“ koji u principu obavljaju istu zadaću kao i „OnCompleteListener()“ samo što su uspješnost, odnosno neuspješnost zadatka odvojenu u zasebne metode. „OnSuccess“ metoda izgleda ovako:

```
1.              public void OnSuccess(Java.Lang.Object result) {
2.
3.                      updateUser(login_rememberMe);
4.                      updateServices();
5.                      chooseCar.updateCars();
6.                      createAppointment.updateUser();
7.                      Intent intent = new Intent(this, typeof(MainActivity));
8.                      StartActivity(intent);
9.
10.             }
```

Prije svega, pozvana je vlastita metoda „updateUser()“ koja prima parametar tipa boolean, odnosno prima tvrdnju točno ili netočno, ovisno o tome je li korisnik označio

kvačicu „Zapamti me“. Ta informacija je preuzeta iz fragmenta za prijavu na gore opisan način.

```
1.      private async void updateUser(bool rememberMe) {
2.          int boolValue = 0;
3.          if (rememberMe == true) boolValue = 1;
4.
5.          User user = new User();
6.          var firebase = new FirebaseClient(loginActivity.FirebaseURL);
7.          FirebaseUser CurrentUser = FirebaseAuth.GetInstance(app).CurrentUser;
8.          id = CurrentUser.Uid;
9.
10.         var data = await firebase.Child("users").Child(id).OnceAsync < Account > ();
11.         foreach(var item in data) {
12.
13.             user.name = item.Object.name;
14.             user.lastName = item.Object.lastName;
15.             user.phone    = item.Object.phone;
16.             user.email    = item.Object.email;
17.             user.city     = item.Object.city;
18.             user.adress = item.Object.adress;
19.
20.         }
21.         con.db.Execute("UPDATE User SET rememberMe = '" + boolValue + "' WHERE uid = '"
+ id + "' ");
22.         con.db.Execute("UPDATE User SET name = '" + user.name + "', lastName = '" + user
.lastName + "', phone = '" + user.phone + "', " + "email = '" + user.email + "', city = '"
+ user.city + "', adress = '" + user.adress + "' WHERE uid = '" + id + "' ");
23.     }
```

Unutar metode „updateUser()“, korištena je integer vrijednost 0 i 1 budući da SQLite boolean vrijednost ne interpretira kao točno i netočno, nego kao 0 i 1. Ta vrijednost upisana je u SQLite bazu podataka iz koje će se, prilikom ponovne prijave, provjeravati ima li potrebe za ponovnim upisivanjem autentifikacijskih podataka korisnika.

Na isti način napravljena je sinkronizacija sa SQLite bazom podataka i u slučaju metoda „updateCars()“ i „updateServices“(). Nadalje, kreira se nova instanca klase Intent (namjera), pomoću koje se pokreće nova aktivnost. Naziv te nove aktivnosti je „MainActivity“ i ona predstavlja glavnu aktivnost aplikacije. U slučaju da je pozvana metoda „OnFailure()“ nakon pokušaja prijave, korisnik biva obaviješten o neuspjeloj prijavi te je zamoljen da pokuša ponovno s istom. Nakon te obavijesti, ponovno je automatski pokrenut fragment za prijavu. Ukoliko je korisnik označio „Zapamti me“ kvačicu, kao što je ranije već spominjano, korisnik neće trebati upisivati podatke za prijavu prilikom sljedećeg pokretanja aplikacije.

Provjera je li korisnik označio kvačicu „Zapamti me“ izvedena je na sljedeći način:

```
1.      if (IsOnline()) {
2.          FirebaseUser user = FirebaseAuth.GetInstance(app).CurrentUser;
3.          if (user != null) {
4.              checkIfRememberMeIsChecked();
5.          }
6.          if (login_rememberMe && user != null) {
7.              Intent intent = new Intent(this, typeof(MainActivity));
8.              StartActivity(intent);
9.              this.FinishAffinity();
10.         }
11.     }
```

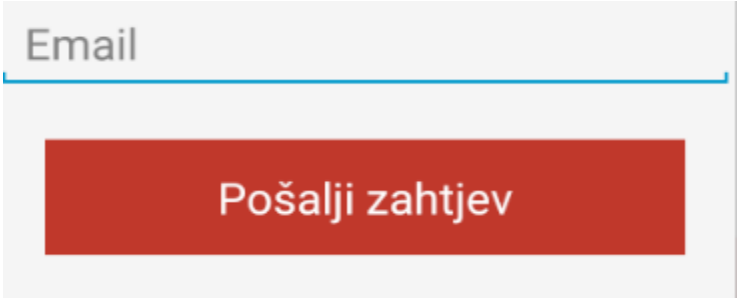
Pomoću ranije kreirane metode „IsOnline()“ provjerava se je li korisnik spojen na Internet. Metoda „IsOnline()“ je često puta korištena kasnije u aplikaciji.

```
1.      public bool IsOnline() {
2.          var cm = (ConnectivityManager) GetSystemService(ConnectivityService);
3.          return cm.ActiveNetworkInfo == null ? false : cm.ActiveNetworkInfo.IsConnected;

4.      }
```

Nadalje, nakon izvršavanja „updateServices()“ metode, kreira se nova instanca „FirebaseUser“ klase kako bi se provjerilo postoji li korisnik koji je trenutno prijavljen u sustav. Ukoliko postoji, provjerava se je li taj korisnik prilikom prethodne prijave označio kvačicu „Zapamti me“. Provjera se obavlja pomoću prethodno kreirane metode „checkIfRememberMeIsChecked()“ gdje se prolazi kroz SQL tablicu „User“ s korisnikom koji je trenutno prijavljen i provjerava se redak koji sadržava boolean varijablu „rememberMeChecked“. Ukoliko su oba uvjeta zadovoljena (korisnik je online i označio je „Zapamti me“ kvačicu) pomoću namjere se otvara glavna aktivnost – „mainActivity“. Na kraju, pomoću „finishAffinity()“ metode zatvara se trenutna aktivnost.

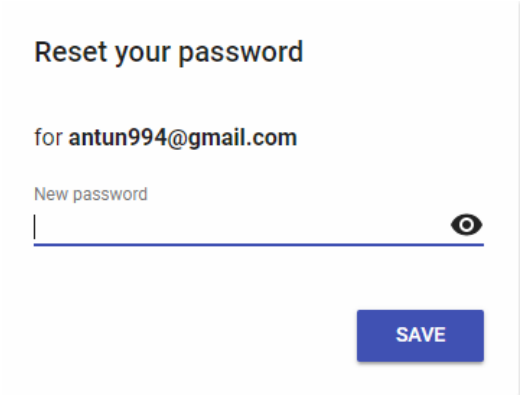
Zadnji element u fragmentu za prijavu – „Zaboravili ste lozinku“ element – pokreće (preko namjere) novi fragment, odnosno fragment za povratak lozinke. Za taj fragment je također vezan događaj preko kojeg će se preuzeti informacije o podacima koje korisnik mora dodati – email samo u ovom slučaju. Ovog puta ne preuzima aktivnost te podatke, nego fragment za prijavu.



Slika 34: Fragment za povrat lozinke

(Izvor: snimka zaslona na računalu autora)

Nakon što je korisnik upisao mail, poziva se metoda „SendPasswordResetEmail()“ koja prima parametar tipa string (email). Metoda se nalazi u klasi „FirebaseAuth“ te je sve već u njoj napravljeno i jedino što treba je instancirati klasu,pozvati metodu i predati parametar. Nakon ispunjavanja email podatka te zatim dodira na „Pošalji zahtjev“ gumb, zatvara se fragment te automatski otvara fragment za prijavu. Korisnik na email prima link za ponovno postavljanje lozinke. Klikom na link korisniku se (u Internet pregledniku) otvara jednostavan fragment za povrat lozinke. Fragment je kreiran od strane tvrtke Google te ga nije potrebno kreirati.

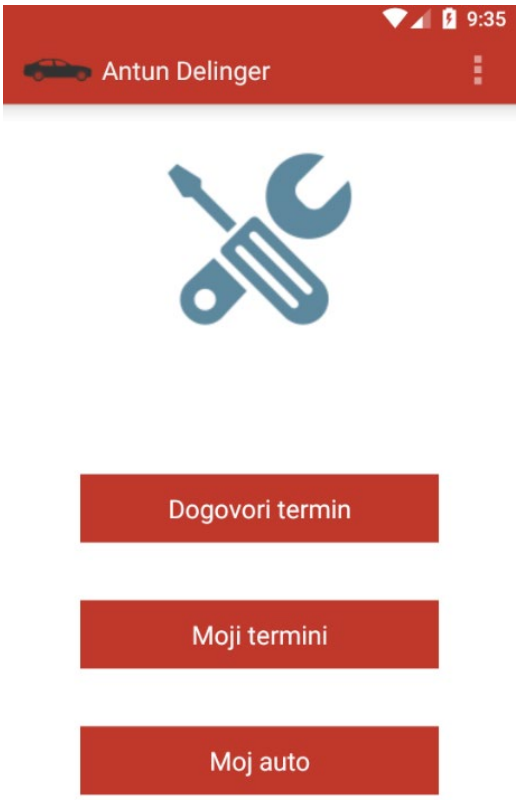


Slika 35: Fragment za povratak lozinke

(Izvor: snimka zaslona na računalu autora)

Korisnik upisuje novu lozinku i klikom na gumb „SAVE“ (koliko je zadovoljio sve uvjete o jačini lozinke) mijenja svoju lozinku te se može prijaviti u sustav s novom lozinkom. Time je završen proces prijave i registracije.

Ukoliko je korisnik uspješno završio prijavu preko namjere, kao što je ranije već opisano, pokreće se glavna aktivnost „MainActivity“.



Slika 36: Prikaz glavne aktivnosti

(Izvor: snimka zaslona na računalu autora)

Prema slici 36, vidljivo je kako je glavna aktivnost izvedena onako kako je prema zahtjevima i prototipu utvrđeno. S lijeve strane, pri vrhu, nalazi se ikona aplikacije, zatim naziv korisnika koji je prijavljen u aplikaciju te s desne strane imamo ikonu koja označava padajući izbornik. Ispod se nalazi tematska ikona te ispod toga imamo 4 gumba. Elementi grafičkog sučelja dodani su u „main.axml“, a svi elementi koji se nalaze na ovoj aktivnosti su već ranije opisani. Za dohvat trenutnog korisnika korištena je vlastita metoda „getUserInfo()“:

```
1.      FirebaseUser users = FirebaseAuth.GetInstance(loginActivity.app).CurrentUser;
2.      id = users.Uid;
3.      List < User > data = con.db.Query < User > ("SELECT * FROM User WHERE uid = '" + id
+ "' ");
4.      foreach(var item in data) {
5.          userName = item.name;
6.          userLastName = item.lastName;
7.      }
```

S obzirom da su korisnikovi podaci već od ranije dodani u offline bazu podataka, nije bilo potrebe spajati se online i dohvaćati podatke. Prema tome, kao što se vidi u kodu iznad, dohvat je izvršen iz offline baze podataka. Prije svega bilo je potrebno dohvatiti

trenutnog korisnika prijavljenog u sustav (za što nije potrebno spajati se na server online) te zatim pomoću metode „Query()“ dohvatiti podatke iz tablice koja ima korisnika s jedinstvenim ključem koji je dodan u „WHERE“ klauzulu u SQL upitu. Podaci koji su dohvaćeni su predani lokalnim varijablama tipa string – „userName“ i „userLastName“. Zatim, uz pomoć svojstva trenutne aktivnosti „Title“ jednostavno je promijenjen naziv aktivnosti iz predefinirane vrijednosti u vrijednost imena i prezimena korisnika. Pokretanjem glavne aktivnosti upisuje se token vrijednost uređaja koja identificira uređaj na kojemu je trenutno instalirana aplikacija. Poziva se izvođenje novog zadatka (engl. Task) unutar kojeg će se izvesti preuzimanje tokena.

```
1.      await Task.Run(() => {
2.          var instanceId = FirebaseInstanceId.Instance;
3.          token = instanceId.GetToken(authorizedEntity, scope);
4.          Android.Util.Log.Debug("TAG", "{0} {1}", instanceId.Token, token, Firebase.Messa
ging.FirebaseMessaging.InstanceIdScope);
5.          makeDatabaseRecord(token);  });
```

Na kraju novokreiranog zadatka poziva se metoda „makeDatabaseRecord()“ kojoj se prosljeđuje vrijednost tokena koja je upravo preuzeta.

```
1.      private void makeDatabaseRecord(string token) {
2.          FirebaseUser users = FirebaseAuth.GetInstance(loginActivity.app).CurrentUser;
3.          id = users.Uid;
4.          var firebase = new FirebaseClient(loginActivity.FirebaseURL);
5.          var item = firebase.Child("tokens").Child(id).PutAsync < string > (token);
6.      }
```

Metoda sprema podataka o tokenu u Firebase bazu podataka pod čvorom „tokens“ te čvorom ispod „id“ gdje se upisuje korisnikov jedinstveni ključ kako bi se znalo koji token kojem korisniku pripada. Dodatno, token se mijenja prilikom svakog ponovnog instaliranja aplikacije te je stoga potrebno ažuriranje tokena. Iz tog se razloga zapis u bazi podataka ažurira prilikom svakog pokretanja aplikacije. U slučaju promjene tokena, poziva se metoda „OntokenRefresh()“ koja registrira promijene tokena. Metoda je prepisana (engl. Override) na način da provjerava je li korisnik trenutno logiran u sustav. Ukoliko korisnik nije logiran u sustav, a dogodila se promjena tokena, briše se instanca tog korisnika, što znači da će se prilikom pokretanja glavne aktivnosti upisati novi, ažurirani token. Ukoliko se dogodila promjena tokena, a da je korisnik logiran u sustav, odmah u „OnTokenRefresh()“ se ažurira token i upisuje nova vrijednost u bazi podataka budući da je poznat korisnik i njegov jedinstveni ključ.

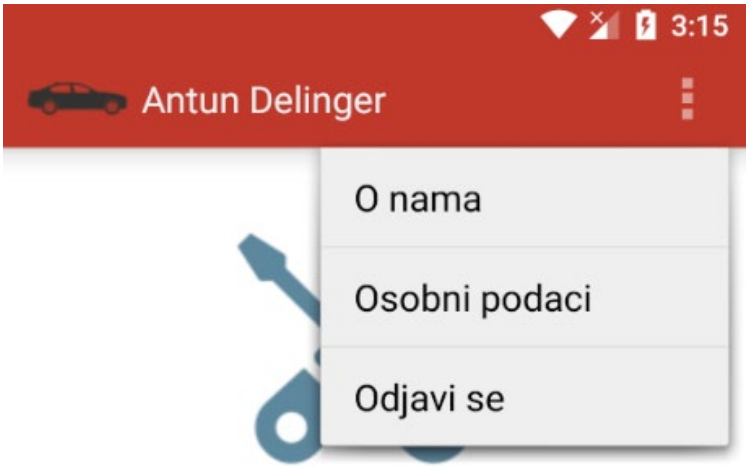
```
1.      public override void OnTokenRefresh() {
2.          base.OnTokenRefresh();
3.          Android.Util.Log.Debug("Refreshed Token:", FirebaseInstanceId.Instance.Token);
4.          FirebaseUser users = FirebaseAuth.GetInstance(loginActivity.app).CurrentUser;
5.          if (users != null) {
6.              id = users.Uid;
7.              var firebase = new FirebaseClient(loginActivity.FirebaseURL);
8.              var item = firebase.Child("tokens").Child(id).PutAsync < string > (FirebaseI
nstanceId.Instance.Token);
9.          } else {
10.             var instanceId = FirebaseInstanceId.Instance;
11.             instanceId.DeleteInstanceId();
12.         }
13.     }
```

Time je završen proces dohvata i upisa dokena uređaja i aplikacija nastavlja s normalnim izvođenjem. Druga bitna stavka vezana uz token, ali i ključna za primanje obavijesti putem Firebase Cloud Messaging, jest klasa „MessagingService“. Potrebno je, dakle, kreirati klasu proizvoljnog naziva, u ovom slučaju „myMessagingServerice“ te zatim naslijediti (engl. Inheritance) svojstva i atribute iz ranije spomenute klase.

```
1.      class MyFirebaseMessagingService: FirebaseMessagingService {
2.
3.          public override void OnMessageReceived(RemoteMessage message) {
4.              base.OnMessageReceived(message);
5.              SendNotification(message.GetNotification().Body);
6.       }
7.          private void SendNotification(string body) {
8.
9.              var intent = new Intent(this, typeof(MainActivity));
10.             intent.AddFlags(ActivityFlags.ClearTop);
11.             var pendingIntent = PendingIntent.GetActivity(this, 0, intent, PendingIntent
Flags.OneShot);
12.             var defaultSoundUri = RingtoneManager.GetDefaultUri(RingtoneType.Notificatio
n);
13.             var notificationBuilder = new NotificationCompat.Builder(this).SetContentTit
le("Moj servis obavijest").SetContentText(body).SetSmallIcon(Resource.Drawable.ifsedan28581
0).SetAutoCancel(true).SetSound(defaultSoundUri).SetContentIntent(pendingIntent);
14.
15.             var notificationManager = NotificationManager.FromContext(this);
16.             notificationManager.Notify(0, notificationBuilder.Build());
17.      } }
```

Nakon nasljeđivanja klase, potrebno je prepisati pravila metode „OnMessageReceived()“ čime se definira ponašanje aplikacije prilikom primanja obavijesti. Metoda dalje poziva drugu metodu, odnosno metodu „SendNotification()“ u kojoj su definirani svi potrebni parametri za primanje poruke kao što je zvuk dolazne poruke i slična ponašanja. Neki od ovih parametara mogu biti prepisani ukoliko su drugačije zadani od pošiljatelja obavijesti, primjerice, pošiljatelj obavijesti može definirati naslov i sadržaj tijela obavijesti kako želi.

Dodirom na ikonu u obliku tri točkice, otvara se padajući izbornik koji nudi tri opcije (slika 37).



Slika 37: Izbornik za odjavu

(Izvor: snimka zaslona na računalu autora)

Meni koji je označen ikonom s tri točkice kreiran je u „actionbar_main.xml“ datoteci, a kod izgleda ovako:

```
1.      < item android: id = "@+id/logOut"
2.      android: orderInCategory = "100"
3.      android: showAsAction = "never"
4.      android: title = "Odjavi se" / >
5.      < item android: id = "@+id/support"
6.      android: title = "O nama" / >
7.       < item android: id = "@+id/personalInfo"
8.      android: title = "Osobni podaci" / >
9.       </menu>
```

Kreirane su tri različite stavke (engl. Item) od kojih svaka predstavlja jednu od opcija prikazanih na slici broj 37. Da bi smo taj meni prikazali na glavnoj aktivnosti, potrebno je nadjačati (engl. Override) metodu „OnCreateOptionsMenu()“. Pomoću „Inflate()“ metode smo kreirali novi izbornik.

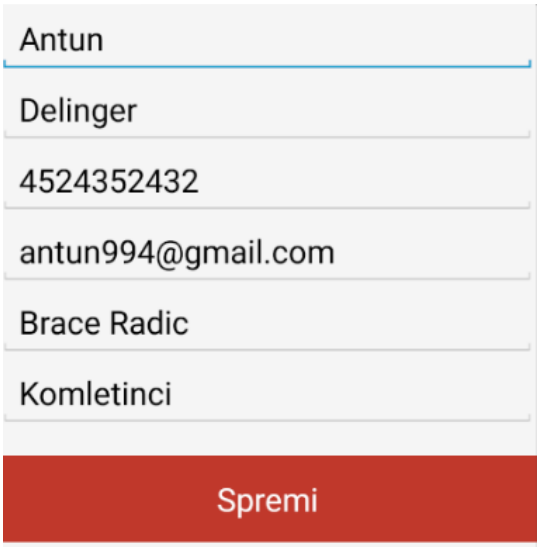
```
1.      public override bool OnCreateOptionsMenu(IMenu menu) {
2.          MenuInflater.Inflate(Resource.Menu.actionbar_main, menu);
3.          return base.OnCreateOptionsMenu(menu);
4.      }
```

Nakon što smo kreirali izbornik, potrebno je programirati logiku. Vrlo je jednostavna procedura jer se ne zahtjeva ništa drugo osim pronalaska unikatnog ključa pojedinog izbornika te reakcije koja se događa na odabranu stavku.

```
1.        public override bool OnOptionsItemSelected(IMenuItem item) {
2.              int id = item.ItemId;
3.              if (id == Resource.Id.logOut) {
4.                  auth.SignOut();
5.                  Intent intent = new Intent(this, typeof(loginActivity));
6.                  StartActivity(intent);
7.              }
8.              if (id == Resource.Id.support) {
9.                  FragmentTransaction transaction = FragmentManager.BeginTransaction();
10.                 aboutUs aboutUs = new aboutUs();
11.                 aboutUs.Show(transaction, "aboutUs");
12.             }
13.             if (id == Resource.Id.personalInfo) {
14.                 FragmentTransaction transaction = FragmentManager.BeginTransaction();
15.                 personalInfo personalInfo = new personalInfo();
16.                 personalInfo.Show(transaction, "personalInfo");
17.             }
18.             return base.OnOptionsItemSelected(item);
19.     }
```

Nakon usporedbe unikatnog ključa (id), utvrđuje se koja stavka je kliknuta. U slučaju „Odjavi se“ pozvana je „SignOut“ metoda iz klase „FirebaseAuth“ pomoću koje se korisnika odjavljuje iz sustava. Nakon toga pomoću namjere se pokreće aktivnost „loginActivity“ gdje se korisnik može ponovno prijaviti.

Ukoliko je korisnik odabrao stavku „osobni podaci“ pokrenut će se fragment „personalInfo“. Fragment otvaramo tako što kreiramo instancu te klase i klase FragmentTransaction, na koncu uz pomoć metode „Show()“ (klasa „fragmentActivity“) pokrećemo novi fragment tako što mu predajemo instancu klase „fragmentTransaction“ i „personalInfo“.



Slika 38: Fragment za prikaz osobnih podataka

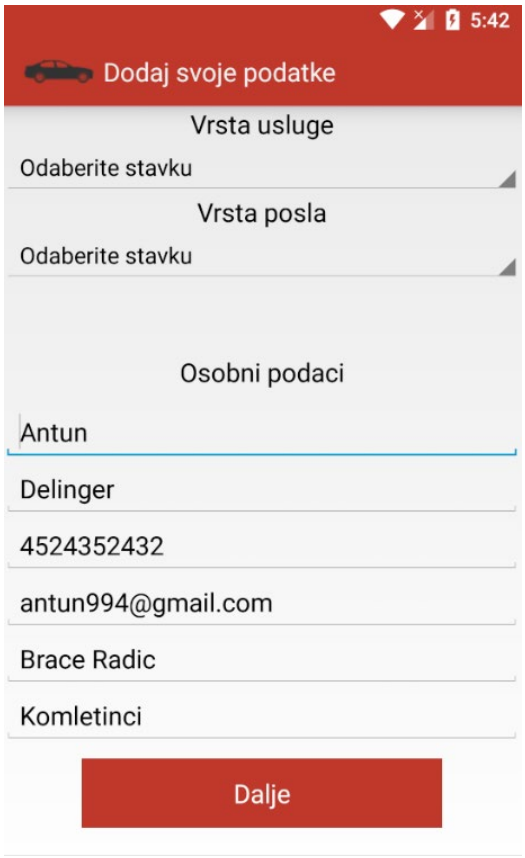
(Izvor: snimka zaslona na računalu autora)

Podatke koji su prikazani možemo mijenjati i spremiti dodirom na gumb „Spremi“, a moguće ih je i izbrisati tako što korisnik ostavi prazna polja i spremi promjenu. U slučaju odabira „O nama“ otvara se sličan fragment kao i kod osobnih podataka samo s podacima autoservisa. Kod je identičan uz jedinu razliku što je instancirana druga klasa.

Slika 39: Fragment za prikaz podataka o autoservisu
(Izvor: snimka zaslona na autorovom računalu)

Podatke na fragmentu za prikaz podataka o autoservisu nije moguće mijenjati niti brisati, njih može jedino zadužena osoba iz servisa mijenjati. Nakon padajućeg izbornika, sljedeći po redu je gumb „Dogovori na termin“. Dodirom na taj gumb, uz pomoć namjere, otvara se nova aktivnost – „createNewAppointment“, a koja služi, kako sam naziv sugerira, za kreiranje novog sastanka. Doduše, predstavlja prvi korak u kreiranju sastanka.

Za kreiranje sastanka korisnik mora proći kroz tri aktivnosti – „createNewAppointment“, „addCarToOrder“ i „confirmOrder“. Unutar aktivnosti možemo odabrati vrstu usluge (u zasebnom izborniku koji se otvara na dodir), vrstu posla (također u zasebnom izborniku) te su nam prikazani od prije dodani osobni podaci. Osobni podaci mogu se mijenjati, a ukoliko nisam dodali broj telefona i adresu, morat ćemo to napraviti u ovoj aktivnosti jer nije moguće nastaviti proces kreiranja sastanka bez tih podataka. Također, odabir usluge i posla je neophodan za nastavak.



Slika 40: Aktivnost za kreiranje novog sastanka
(Izvor: snimka zaslona na računalu autora)

Spomenuti izbornik za odabir usluge i posla (engl. Spinner), dodaje se u .axml datoteci.

```
1.      < Spinner android: layout_width = "match_parent"
2.      android: layout_height = "wrap_content"
3.      android: entries = "@array/dropdownType"
4.      android: id = "@+id/vrstaUsluge" / >
```

Takav izbornik se povezuje s aktivnosti na isti način kao i drugi elementi sučelja. No, u izbornik se mogu dodati predefinirane stavke. Primjerice, u datoteku „String.xml“ dodajemo stavku kao resurs. Kao što vidimo u kodu gore, naredbom „android: entries“ specificirana je putanja na niz stavki „array/dropDownType“. Kako je kreiran taj niz stavki možemo vidjeti u kodu dolje:

```
1.      <? xml version = "1.0"
2.      encoding = "utf-8" ?> < resources >
3.       < string name = "app_name" > carServiceApp < /string>
4.      < string - array name = "dropdownType" >
5.      < item > Odaberite stavku < /item>
6.       < /string-array>
7.      < /resources>
```

Izrada drugih elemenata je već prethodno objašnjena. Preuzimanje podataka preuzima se iz SQLite baze podataka kako bi se ubrzao dohvat podataka. Način

preuzimanja iz SQLite baze je već ranije objašnjen. Dodirom na gumb „Dalje“ vrši se nekoliko provjera. Prije svega, moraju sva polja biti ispunjena, a osim toga, moraju biti dodani vrsta usluge i vrsta posla kako bi se moglo nastaviti.

```
1.      if (userInput_ime.Text == "" || userInput_prezime.Text == "" || userInput_broj.Text
== "" || userInput_email.Text == "" || userInput_mjesto.Text == "" || userInpu_ulicaibr.Tex
t == "") {
2.          Toast.MakeText(this, "Sva polja moraju biti ispunjena!", ToastLength.Short).Show
();
3.          return;
4.      }
5.      if (vrstaUsluge.SelectedItem.ToString() == "Odaberite stavku") {
6.          Toast.MakeText(this, "Vrsta usluge mora biti odabrana", ToastLength.Short).Show(
);
7.          return;
8.      }
9.      if (vrstaPosla.SelectedItem.ToString() == "Odaberite stavku") {
10.         Toast.MakeText(this, "Vrsta posla mora biti odabrana", ToastLength.Short).Show()
;
11.         return;
12.     }
```

Zbog osiguravanja efikasnog slanja zahtjeva za novim sastankom, postoji provjera vezna uz Internet vezu. Naime, zbog sigurnosti, mora postojati veza na Internet kako bi se mogao napraviti zahtjev za sastankom, a samim time nije dopušten nastavak u sljedeću aktivnost bez pristupa internetu.

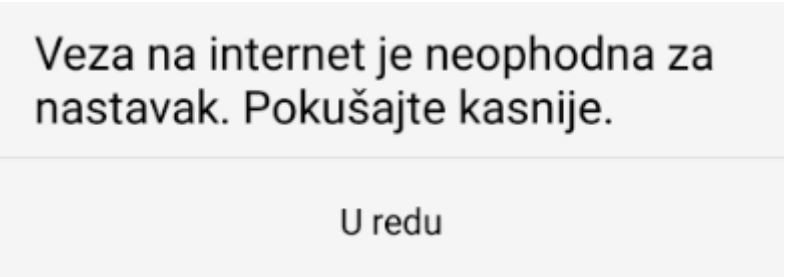
```
1.      if (!IsOnline()) {
2.          MainActivity.checkIfOnline(this, this, this);
3.          return;
4.      }
```

Ukoliko je uvjet ispunjen (nema internetske veze), poziva se statička metoda „checkIfOnline()“ iz klase „MainActivity“.

```
1.      public static void checkIfOnline(Context context, Activity activity, IDialogInterfac
eOnDismissListener listener) {
2.          AlertDialog.Builder dialogs = new AlertDialog.Builder(context);
3.          dialogs.SetOnDismissListener(listener);
4.          dialogs.SetMessage("Veza na internet je neophodna za nastavak. Pokušajte kasnije
.");
5.          dialogs.SetPositiveButton("U redu", (senderAlert, args) => {
6.              activity.Finish();
7.              dialogs.Dispose();
8.          });
9.          Dialog alertDialogs = dialogs.Create();
10.         alertDialogs.Show();
11.         return;
12.     }
```

Ova metoda korištena je više puta kroz aplikaciju kada je uočen izostanak veze s internetom, a upravo iz tog razloga metoda i je statička kako bi se mogla pozivati iz

drugih aktivnosti. Ova metoda, naime, kreira novi dijalog koji obavještava korisnika o tome kako je izgubio vezu s internetom te kako ne može nastaviti koristiti aplikaciju.



Slika 41: Dijalog koji izvještava o prekidu veze s internetom
(Izvor: snimka zaslona na računalu autora)

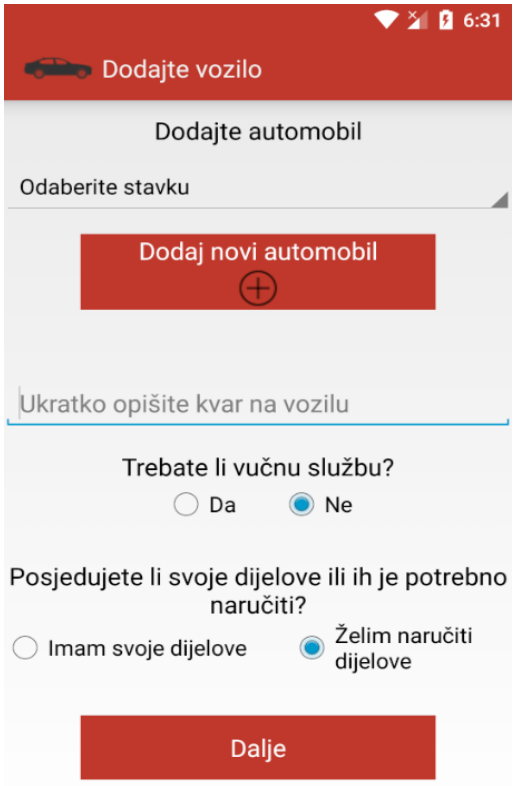
Dodir na „U redu“ gumb ili izlazak iz dijalog poziva „OnDismiss()“ metodu, a koja je definirana kao obavezna sučeljem (engl. Interface) „IDialogInterface“.

```
1.      public void OnDismiss(IDialogInterface dialog) {
2.          dialog.Dispose();
3.          this.Finish();
4.          Intent intent = new Intent(this, typeof(loginActivity));
5.          StartActivity(intent);
6.      }
```

Unutar metode zadano je ponašanje aplikacije u slučaju zatvaranja dijaloga, a to je gašenje trenutne aktivnosti i pokretanje aktivnosti za ponovnu prijavu. Ukoliko je veze na Internet bila prisutna i nije bilo potrebe za pokretanjem dijaloga, poziva se ažuriranje korisničkih podataka. Ažuriranje podataka, na Firebase bazi podataka, obavlja se isto kao i upis podataka, uz pomoć „OnceAsync()“ metode, a jedina je razlika ta da ukoliko je poznat cijeli put do objekta – on se ažurira, a ukoliko nije – on se kreira. Nadalje, provjerava se je li korisnik tek počeo sa kreiranjem sastanka ili se vraća nešto izmijeniti od podataka. Ukoliko je „updateRequested“ jednaka „False“ – to znači da je korisnik tek započeo kreiranje sastanka. Dakle, taj uvjet se preskače i aplikacija otvara sljedeću aktivnost – aktivnost „addCarToOrder“ – koristeći namjeru. U slučaju da se korisnik vratio nešto izmijeniti u aktivnost, to ćemo znati tako što će „updateRequested“ varijabla biti jednaka „True“, a to znamo tako što će to biti registrirano u „OnResume()“ metodi koja se pokreće kada aplikacija nastavlja svoju aktivnost (bez ponovnog pokretanja), a što znači da se korisnik u nju vratio nakon kraćeg izbivanja – primjerice boravka u aktivnosti „addCarToOrder“. Ukoliko je, dakle, korisnik došao nešto izmijeniti, potrebno je poslati te izmijenjene podatke sljedećoj aktivnosti.

```
if (updateRequested) {
1.          Intent intent = new Intent(this, typeof(addCarToOrder)).SetFlags(ActivityFlags.R
eorderToFront);
2.          intent.PutExtra("vrstaUsluge", vrstaUsluge.SelectedItem.ToString());
3.          intent.PutExtra("vrstaPosla", vrstaPosla.SelectedItem.ToString());
4.          StartActivity(intent);
5.      } else {
6.          Intent intent = new Intent(this, typeof(addCarToOrder));
7.          intent.PutExtra("vrstaUsluge", vrstaUsluge.SelectedItem.ToString());
8.          intent.PutExtra("vrstaPosla", vrstaPosla.SelectedItem.ToString());
9.          StartActivity(intent);
}
```

Kao što je moguće vidjeti, u oba slučaja pokrenuta je nova aktivnost – „addCarToOrder“. Ta aktivnost zadužena je primarno za dodavanje novog vozila i informacija o istom. Vozilo je moguće kreirati sa svim novim podacima, ali je moguće i dodati vozilo koje je već prethodno kreirano – što je moguće obaviti kroz aktivnost „addCar“ kojoj je moguće pristupiti kroz gumb „Moja Vozila“ na glavnoj aktivnosti. Osim toga, moguće je dodati kratki opis kvara na vozilu, a također je moguće dodati informacija je li potrebna vučna služba za vozilo kojemu je potreban popravak te hoće li biti potrebna nabavka novih dijelova od strane servisa ili klijent već posjeduje dijelove.



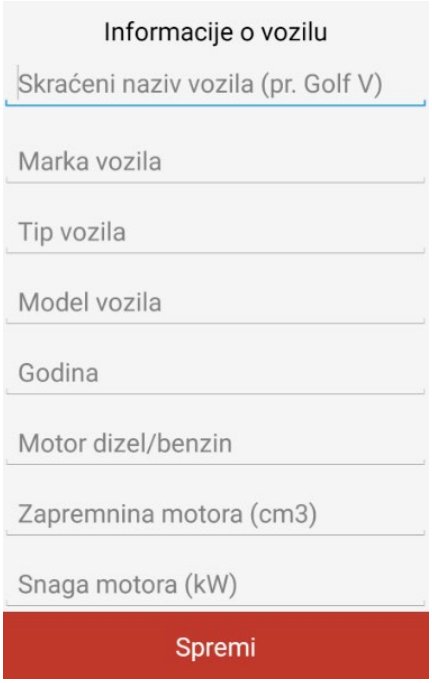
Slika 42: Aktivnost za dodavanje vozila prilikom kreiranja sastanka

(Izvor: snimka zaslona na računalu autora)

Vozilo se dodaje u online bazu podataka, ali i u offline bazu podataka. Tip podacima pristupa se iz offline baze podataka zbog brzine rada sustava, a podaci su sinkronizirani s online bazom podataka prilikom svake prijave u sustav.

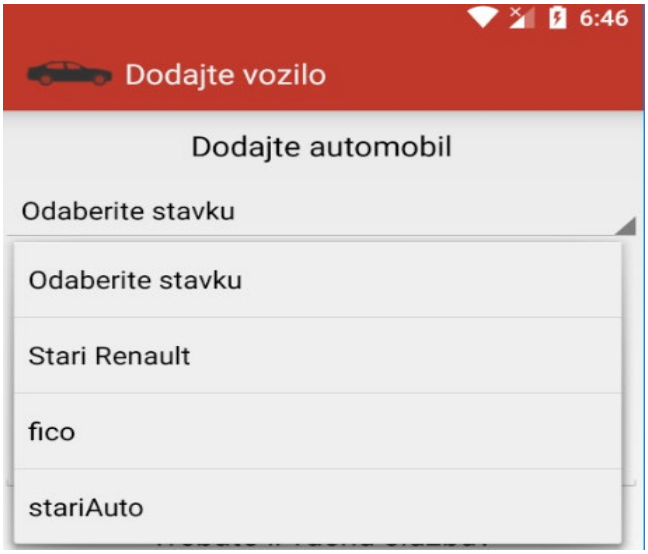
```
1.      class CarDetails {
2.          public string uid               {get; set; }
3.          public string markaVozila       {get; set; }
4.          public string tipVozila         {get; set; }
5.          public string modelVozila       {get; set; }
6.          public string godina            {get; set; }
7.          public string tipMotora         {get; set; }
8.          public string zapremninaMotora {get; set; }
9.          public string snagaMotora       {get; set; }
10.     }
```

Objekt koji se upisuje o online bazu podataka sadrži osim atributa, a od kojih svaki predstavlja neku osobinu vozila. Objekt se sprema u online bazu podataka s prvim čvorom koji predstavlja jedinstveni ključ korisnika, zatim s drugim čvorom koji predstavlja naziv vozila. Prema samim pravilima Firebase baze podataka; ne može postojati dva jednaka čvora na istoj razini, prema tome jedan korisnik može imati samo jedno vozilo s istim nazivom. Korisniku je taj atribut predstavljen kao „Kratki naziv vozila“, a prema kojemu će razlikovati dodana vozila. To je ujedno i osnovna razlika u odnosu na SQL tablicu koja je također dodana, a koja ima jedan atribut više – „carName“, a koji ima specificirano „UNIQUE“ ograničenje koje označava da korisnik može imati samo jednu tablici s vozilom određenog imena.



Slika 43: Fragment za dodavanje novog vozila
(Izvor: snimka zaslona na računalu autora)

Ako nemamo niti jedno dodano vozilo ili za pregled želimo naručiti možda neko drugo vozilo koje do sada nismo dodali, odabiremo „Dodaj novi automobil“ gumb. Dodirom na taj gumb otvara se fragment „addCar“ (slika 43). Dodirom na gumb „Spremi“ automobil se sprema u bazu podataka, fragment se zatvara te je moguće vidjeti dodano vozilo pod izbornikom za odabir već dodanih vozila (slika 44).



Slika 44: Izbornik za odabir automobila

(Izvor: Snimka zaslona na računalu autora)

Kod dodavanja novog vozila sva polja moraju biti popunjena inače je korisnik obaviješten o tome kako nije ispunio sva polja te je onemogućen spremiti vozilo dok ne ispuni sva polja. Kako bi aktivnost za dodavanje vozila u kreiranje sastanka, odnosno element koji lista vozilo, moglo registrirati novo vozilo, kreiran je događaj koji javlja zatvaranje fragmenta. Na tu reakciju koju događaj javi, ponovno su učitani podaci u izbornik koji prikazuje dodana vozila. Kao što možemo vidjeti na slici 42, dodani su elementi za označavanje dva odgovora (engl. Radio Button).

```
1.      < RadioButton android: layout_width = "wrap_content"
2.      android: layout_height = "wrap_content"
3.      android: checked = "false"
4.      android: text = "Da"
5.      android: id = "@+id/yesButton"
6.      android: layout_marginRight = "15dip" /
```

Elementi za označavanje koriste se uz pomoć svojstva „Checked“ koje je tipa boolean. Označeni element ima svojstvo „True“ (točno), a element koji nije označen ima svojstvo „False“ (netočno). Dodirom na gumb „Dalje“ provode se iste provjere kao u prethodnoj aktivnosti – ispunjenost svih polja te je li korisnik prvi put otvorio aktivnost ili se vratio

u nju nešto izmijeniti, obje provjere obavljaju se na isti način kao u prethodnoj aktivnosti. Osim toga, determinira se odgovor korisnika na elemente za označavanje – koristeći metodu „checkIfChecked()“ .

```
1.      private void checkIfChecked() {
2.
3.          if (yesButton.Checked) potrebnaVucnaSluzba      = true;
4.          if (noButton.Checked) potrebnaVucnaSluzba       = false;
5.          if (imamDijelove.Checked) potrebnoNarucivanje   = false;
6.          if (naruciDijelove.Checked) potrebnoNarucivanje = true;
7.
8.      }
```

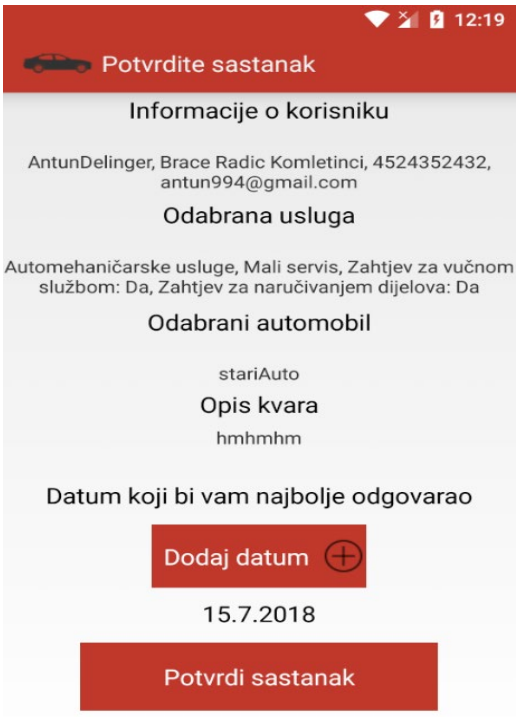
Ovi podaci, zajedno s podacima iz prethodne aktivnosti, šalju se aktivnosti za potvrdu sastanka – „confirmOrder“ aktivnosti.

```
1.      Intent intent = new Intent(this, typeof(confirmOrder));
2.      intent.PutExtra("vrstaPosla", vrstaPosla);
3.      intent.PutExtra("vrstaUsluge", vrstaUsluge);
4.      intent.PutExtra("carChosen", spinner.SelectedItem.ToString());
5.      intent.PutExtra("potrebnaVucnaSluzba", potrebnaVucnaSluzba);
6.      intent.PutExtra("opisKvara", opisiteProblem.Text);
7.      intent.PutExtra("potrebnoNarucivanje", potrebnoNarucivanje);
8.      StartActivity(intent);
```

Podaci se šalju u aktivnost za potvrdu sastanka iz razloga jer se sastanak ne kreira u bazi podataka sve do potvrde u zadnjoj aktivnosti koja je zadužena za kreiranje sastanka – „confirmOrder“ aktivnosti. Aktivnost za potvrdu kreiranja sastanka kako bi dobio informacije od prethodnih aktivnosti koristi naredbu „GetExtraString()“ koja se nalazi u klasi „Intent“. Osim što ova završna aktivnost na ovaj način prima informacije prethodnih aktivnosti, tako su i prethodne aktivnosti preuzimale informacije jedna od drugih budući da je to osnovni način prosljeđivanja informacija između aktivnosti.

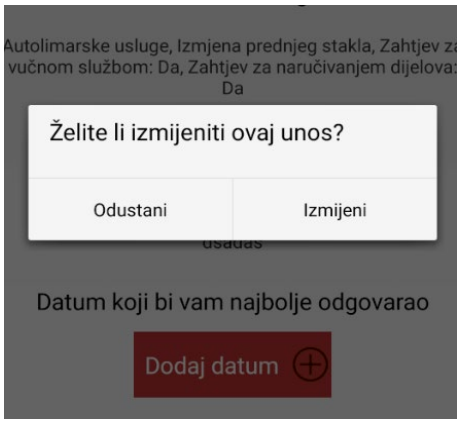
```
1.      vrstaPosla = Intent.GetStringExtra("vrstaPosla");
2.      vrstaUsluge = Intent.GetStringExtra("vrstaUsluge");
3.      carChosen = Intent.GetStringExtra("carChosen");
4.      getOpisKvara = Intent.GetStringExtra("opisKvara");
5.      potrebnaVucnaSluzba = Intent.GetBooleanExtra("potrebnaVucnaSluzba", false);
6.      potrebnoNarucivanje = Intent.GetBooleanExtra("potrebnoNarucivanje", false);
```

Kliko na gumb „Dalje“ otvara se, dakle, aktivnost „confirmOrder“ za potvrdu kreiranja sastanka koja služi za pregled do sad definiranih informacija glede zahtjeva te mogućnost izmjene istih. Osim toga, moguće je dodati i datum koji bi klijentu najbolje odgovarao za servis vozila.



Slika 45: Aktivnost za potvrdu sastanka
(Izvor: snimka zaslona na računalu autora)

Na slici 45 vidljiv je izgled aktivnosti za potvrdu sastanka. Na samom vrhu imamo informacije o korisniku, zatim ispod imamo informaciju o odabranoj usluzi, vučnoj službi te dijelovima. Nadalje, prikazano je koji je automobil odabran i kratki opis kvara kojeg je klijent upisao u prethodnoj aktivnosti. Gumb „Dodaj datum“ služi za otvaranje kalendara u novom fragmentu te odabir datuma kada bi klijentu najbolje odgovarao servis. Za kraj, imamo gumb „Potvrdi sastanak“ koji kreira novi sastanak u bazi podataka i vraća korisnika natrag u glavnu aktivnost.



Slika 46: Fragment koji vodi na izmjenu podataka
(Izvor: snimka zaslona na računalu autora)

Dodir na svaki od podataka nudi mogućnost izmjene podataka koje je korisnik dodao, odnosno povratak u neku od prethodnih aktivnosti. Prije odlaska u aktivnost, otvoren je dijalog koji traži potvrdu za izmjenom. Potvrdnim odgovorom („Izmijeni“) (slika 46) ponovno se otvara aktivnost pomoću namjere. Unutar otvorene aktivnosti jednostavno se izmjeni stari zapis s novim zapisom i nastavi procedura kao i prvi put. Ukoliko nemamo potrebu za izmjenama i zadovoljni smo s dodanim podacima, ono što slijedi je (opcionalno) dodavanje datuma koji bi klijentu najbolje odgovarao.

Slika 47: Fragment u kojemu se nalazi kalendar
(Izvor: snimka zaslona na računalu autora)

Kalendar se otvara pomoću fragmenta naziva „calendar“. Fragment otvara layout datoteku „calendar.axml“ u koju je dodan element koji predstavlja kalendar. Inače, kalendar je ugrađen element koji nije potrebno kreirati, nego samo dodati u axml datoteku.

```
1.      < CalendarView android: layout_width = "750px"
2.      android: layout_height = "600px"
3.      android: id = "@+id/calendarView1"
4.      android: alwaysDrawnWithCache = "false"
5.      android: focusedByDefault = "true" / >
```

U „calendar.axml“ datoteku dodan je još i gumb „Odaberi“ pomoću kojeg odabiremo željeni datum i izlazimo iz fragmenta. Datum prepoznajemo tako što registriramo događaj na svaki odabir datum od strane korisnika i taj odabir spremamo u privatnu varijablu tipa „String“.

```
1.      private void MCalendar_DateChange(object sender, CalendarView.DateChangeEventArgs e)
 {
2.          datePicked = e.DayOfMonth + "." + e.Month + "." + e.Year; }
```

Nakon što je datum spremljen u varijablu „datePicked“, kreiran je događaj na zatvaranje fragmenta kao što je to već bio slučaj ranije u aplikaciji. Klikom na gumb „Odaberi“ pokreće se taj događaj i na taj način se prenosi ta informacija u aktivnost za potvrdu sastanka.

```
1.      private void SaveButton_Click(object sender, EventArgs e) {
2.          onDatePickedEvent.Invoke(this, new onDatePickedEventArgs(datePicked));
3.          this.Dismiss(); }
```

Gumb „Potvrdi sastanak“ je posljednji element ove aktivnosti. Nakon dodira, prije svega provjerava se je li korisnik online – nije moguće kreirati zahtjev ukoliko korisnik nije online. Ako je provjera uspješno prošla, kreira se novi sastanak u bazi podataka. Za to je potrebno kreirati objekt koji će sadržavati sve potrebne atribute.

```
1.      orders orders = new orders();
2.      orders.uid                = id;
3.      orders.carName            = carChosen;
4.      orders.vrstaUsluge        = vrstaUsluge;
5.      orders.vrstaPosla         = vrstaPosla;
6.      orders.opisKvara          = opisKvara.Text;
7.      orders.datum              = DateTime.Now.ToLocalTime().ToString();
8.      orders.vucnaSluzba        = potrebnaVucnaSluzba;
9.      orders.dijelovi           = potrebnoNarucivanje;
10.     orders.pozeljniDatum      = addedDate.Text;
11.     orders.vrijemeServisa     = "Nepoznato";
12.     orders.cijena             = "Nepoznato";
13.     orders.napomenaServisera = "Serviser još uvijek nije dodao nikakvu napomenu";
14.     orders.status             = "Poslano na obradu";
```

Nakon kreiranog objekta i dodjeljivanja vrijednosti pojedinog atributa taj objekt se predaje metodi pomoću koje se kreira nova „tablica“ u Firebase bazi podataka. Jedna bitna stvar u slučaju kreiranja ove tablice: kreiranje jedinstvenog ključa (id) u ovom slučaju predstavljalo je izazov. Naime, jedinstveni ključ koji Firebase generira prikladan je za upotrebu u programerskom smislu, ali nije zgodan za prezentaciju korisniku unutar aplikacije (kako bi se razlikovale narudžbe jedna od druge) pa je stoga tu bilo potrebno nešto elegantnije rješenje.

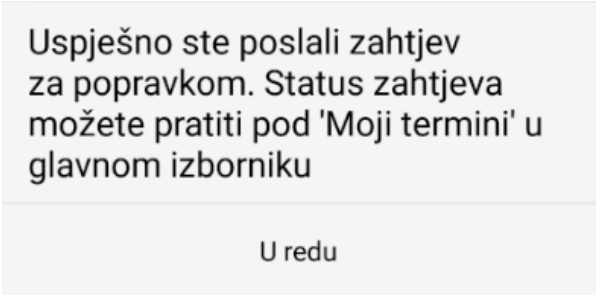
Osim, dakle, uobičajenog jedinstvenog ključa kojeg generira sama baza podataka, korišteni su prirodni brojevi od 1 – n (n = broj sastanaka pojedinog korisnika). S obzirom da Firebase nema „Auto Increment“ mogućnost, to je izvedeno tako što su izvučeni svi sastanci pojedinog korisnika iz baze, prebrojani i tomu broju je zatim dodan još jedan broj.

```
1.      int numOfOrders = 1;
2.      orderID = JsonConvert.SerializeObject(numOfOrders.ToString());
3.      List < orders > allOrders = new List < orders > ();
4.      var getONumberOfOrders = await firebase.Child("order").Child(id).OnceAsync < orders
> ();
5.      foreach(var item in getONumberOfOrders) {
6.          allOrders.Add(new orders {
7.              carName = item.Object.carName  });
8.          numOfOrders = allOrders.Count + 1;
9.          orderID = numOfOrders.ToString();
10.         orderID = JsonConvert.SerializeObject(orderID);  }
11.         orders.id = numOfOrders.ToString();
```

Nakon što je utvrđen redni broj sastanka, a samim time i jedinstveni ključ sastanka (odnosi se samo na određenog korisnika – može biti isti jedinstveni ključ za dva različita korisnika) slijedi kreiranje tablice u bazi podataka.

```
1.      var addOrder = firebase.Child("order").Child(id).Child(orderID.ToString()).PostAsync
 < orders > (orders);
```

To će kreirati čvor u Firebase bazi podataka kao na slici 9 (poglavlje 4.2.1.). Većina podataka odnosi se na podatke koje je korisnik dodao, no, atributi „cijena“, „napomenaServisera“, „status“ i „vrijemeServisa“ namijenjeni su za nadležni servis te korisnik na njih nema utjecaja i predani su kao prazni atributi. Nakon što je sastanak uspješno kreiran time se završava proces kreiranja sastanka te se otvara dijalog koji korisnika obavještava o završetku istog.

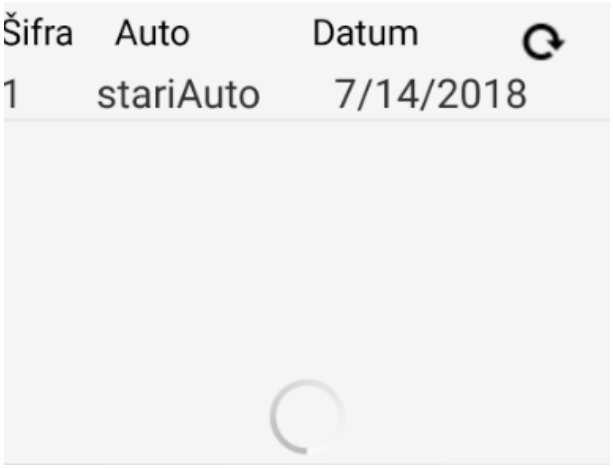


Slika 48: Fragment „Uspješno kreiran sastanak“
(Izvor: snimka zaslona na računalu autora)

Dodir na gumb „U redu“ u dijalogu ili klik na tipku „Back“ ili u principu bilo kakvo zatvaranje dijaloga vodi korisnika u početnu aktivnost. U početnoj aktivnosti, na što dijalog jasno ukazuje, korisnik može kliknuti na „Moji termini“ i tamo pratit razvoj događaja u sastanku.

```
1.      public void OnDismiss(IDialogInterface dialog) {
2.          dialog.Dispose();
3.          this.Finish();
4.          Intent intent = new Intent(this, typeof(loginActivity));
5.          StartActivity(intent); }
```

Dodirom na „Moji termini“ otvara se fragment u kojem se nalazi lista s prethodno kreiranim terminima (sastancima) od strane aktivnog korisnika.



Slika 49: Fragment „Moji termini“
(Izvor: snimka zaslona na računalu autora)

Fragment se sastoji od liste sa stavkama (engl. listview) koja se sastoji od tri kolone: šifra, auto i datum. Na fragmentu se također nalaze i tipka za osvježavanje podataka te traka za napredak koji označava da je u tijeku osvježavanje podataka. Lista sa stavkama u više kolona ne postoji već ugrađena pa je stoga bilo potrebno kreirati ju. Prije svega, potrebno je napraviti axml datoteku i unutar nje tri tekstualna polja od kojih svaki zauzima trećinu ekrana. Unutar horizontalnog LinearLayout oblika sučelja definiramo koliko će biti ukupna suma elemenata („weightSum“). Zatim svakom od elemenata definiramo koliki će dio od te ukupne sume oni zauzimati, u ovom slučaju 0.33 (33%).

```
1.      <? xml version = "1.0"
2.      encoding = "utf-
8" ?> < LinearLayout xmlns: android = "http://schemas.android.com/apk/res/android"
3.      android: orientation = "horizontal"
4.      android: layout_width = "match_parent"
5.      android: layout_height = "match_parent"
6.      android: weightSum = "1" >
7.      < TextView android: text = "Broj narudžbe"
8.      android: layout_width = "wrap_content"
9.      android: layout_height = "match_parent"
10.     android: id = "@+id/brojNarudzbe"
11.     android: layout_weight = "0.33"
12.     android: textSize = "20dp" / >
13.     < TextView android: text = "Naziv auta"
14.     android: layout_width = "wrap_content"
15.     android: layout_height = "match_parent"
16.     android: id = "@+id/nazivAuta"
17.     android: layout_weight = "0.33"
18.     android: textSize = "20dp" / >
19.     < /LinearLayout>
```

Svaki od tekstualnih objekata predstavlja jedan red u listi stavki. Kako bi se kreirala lista od tih objekata, potrebno je kreirati klasu koja nasljeđuje klasu „BaseAdapter“. Unutar klase definira se tip objekta koji će se prosljeđivati u listu stavki.

```
1.      class listViewAdapter: BaseAdapter < orders > {
2.          public List < orders > mOrders = new List < orders > ();
3.          private Context context;
4.          public listViewAdapter(Context Context, List < orders > m_Orders) {
5.              mOrders = m_Orders;
6.              context = Context;   }
7.          public override orders this[int position] => mOrders[position];
8.          public override int Count => mOrders.Count;
9.          public override long GetItemId(int position) {
10.             return position;  }
11.         public override View GetView(int position, View convertView, ViewGroup parent) {
12.             View row = convertView;
13.             if (row == null) {
14.     row = LayoutInflater.From(context).Inflate(Resource.Layout.listViewRows, null, false
);  }
15.       TextView brojNarudzbe = row.FindViewById < TextView > (Resource.Id.brojNarudzbe);
16.       brojNarudzbe.Text = mOrders[position].uid.ToString();
17.       TextView nazivAuta = row.FindViewById < TextView > (Resource.Id.nazivAuta);
18.       nazivAuta.Text = mOrders[position].carName;
19.      TextView datumNarudzbe = row.FindViewById < TextView > (Resource.Id.datumNarudzbe);
20.             datumNarudzbe.Text = mOrders[position].datum;
21.             return row;   } }
```

Klase „BaseAdapter“ zahtjeva implementaciju nekoliko metoda, što je i vidljivo u gornjem kodu. Metoda „GetView()“ na kraju će popuniti tekstualne objekte u axml datoteci. Nakon toga, potrebno je još pozvati tu kreiranu listu u fragmentu u kojemu će se prikazivati. Postupak je identičan kao i u slučaju već gotovo, jednostavne liste sa samo jednom kolonom.

```
1.      List < orders > allOrders = new List < orders > ();
2.      var getONumberOfOrders = await firebase.Child("order").Child(id).OnceAsync < orders
> ();
3.      foreach(var item in getONumberOfOrders) {
4.          allOrders.Add(new orders {
5.              carName = item.Object.carName
6.          });
7.          numOfOrders = allOrders.Count + 1;
8.          orderID = numOfOrders.ToString();
9.          orderID = JsonConvert.SerializeObject(orderID);
10.         var getOtherData = await firebase.Child("order").Child(id).Child(item.Key).OnceA
sync < orders > ();
11.         foreach(var Otheritem in getOtherData) {
12.             stringOrderID = Otheritem.Object.id;
13.             ordersList.Add(new orders {
14.                 uid = Otheritem.Object.id, carName = Otheritem.Object.carName, datum = O
theritem.Object.datum.Substring(0, 10)
15.             });
16.         }
17.     }
18.     listViewAdapter adapter = new listViewAdapter(view.Context, ordersList);
19.     ordersLV.Adapter = adapter;
20.     progressBar.Visibility = ViewStates.Invisible;
21.     refreshButton.Enabled = true;
```

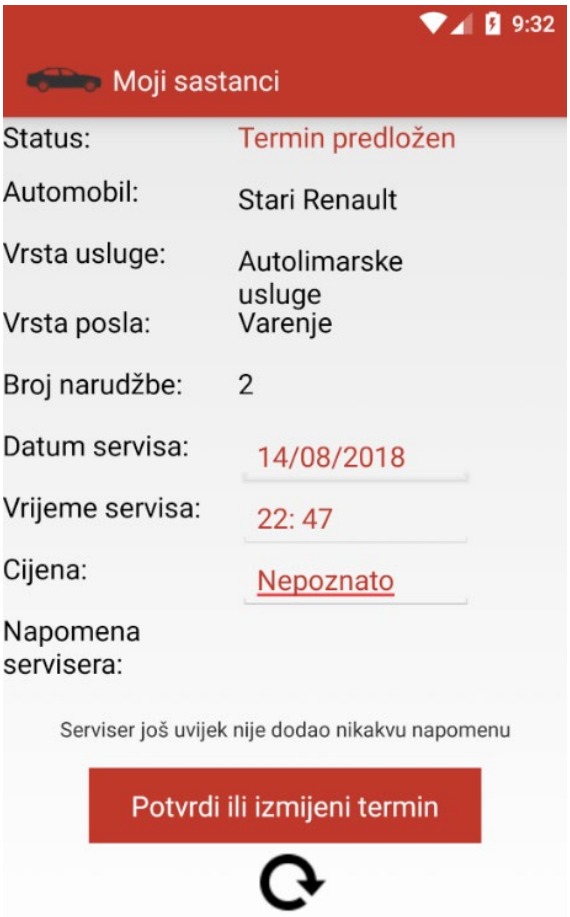
U prethodno prikazanom kodu vidljivo je na koji način su dohvaćeni podaci iz Online baze. U ovom slučaju, izbjegnuto je korištenje offline baze podataka budući da se radi o bazi podataka na koju ima utjecaj i ovlaštena osoba u autoservisu pa prema tome ne bi imalo niti smisla korištenje i sinkroniziranje offline baze podataka. Dohvaćeni su svi podaci s jedinstvenim ključem korisnika koji je trenutno prijavljen u sustav, a od toga su izdvojeni jedinstveni ključ sastanka (šifra), naziv auta te datum kada je kreiran zahtjev. Na kraju koda vidimo kako je kreirana instanca klase „listViewAdapter“ što je klasa koju smo mi kreirali kako bismo napravili svoju listu stavki. Proslijeđeni su joj parametri „view.Context“ (kontekst fragmenta) i „ordersList“ (lista s podacima“). Nadalje, kako se u axml datoteci fragmenta za prikaz termina nalazi zadana lista stavki („ordersLV“) kojoj moramo definirati adapter koji smo prethodno kreirali. Za kraj, definirano je stanje trake za napredak kao „nevidljivo“ što označava završetak dohvaćanja podataka te je omogućena upotreba gumba za osvježavanje podataka naredbom „refreshButton.Enabled = true“.

Gumb za osvježavanje podataka nije uvijek omogućen iz jednostavnog razloga kako bi se izbjeglo osvježavanje za vrijeme već pokrenutog osvježavanja. Stoga je, prilikom dodira na njega, gumb za osvježavanje onemogućen te je na kraju procesa osvježavanja ponovno omogućen.

Nakon dodira na jedan od ponuđenih sastanaka, iz liste stavki uzima se informacija o šifri sastanka koja je unikatna za pojedinog korisnika te se ta ista šifra prosljeđuje novoj aktivnosti koja se otvara – aktivnosti za pregled odabranog sastanka. Odabir stavke se prepoznaje po događaju koji registrira klik na stavku.

```
1.      private void OrdersLV_ItemClick(object sender, AdapterView.ItemClickEventArgs e) {
2.      Intent intent = new Intent(view.Context, typeof(myAppointments));
3.      intent.PutExtra("orderID", ordersList[e.Position].id);
4.                  StartActivity(intent);
5.      }
```

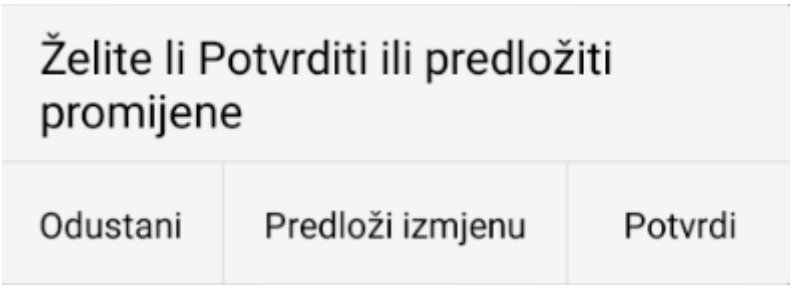
Nakon što je korisnik odabrao jedan od sastanaka, putem namjere se otvara aktivnost za prikaz informacija o sastanku, a istoj je proslijeđena informacija o unikatnoj šifri tog sastanka.



Slika 50: Aktivnost za prikaz informacija o sastanku
(Izvor: snimka zaslona na računalu autora)

Spomenuta aktivnost sastoji se od jednostavnih elemenata – tekstualni elementi, traka za napredak te gumbi za osvježavanje podataka i potvrdu ili izmjenu sastanka. Korisnik (osim podataka o datumu i vremenu servisa) nema mogućnost uređivati podatke - svi se dohvaćaju iz Firebase baze podataka. Svi elementi ove aktivnosti izrađeni su kroz već ranije opisane procedure. Najvažnije stavke ove aktivnosti predstavljaju polja „Status“, „Cijena“, „Datum servisa“ i „Vrijeme servisa“ iz razloga što te informacije dostavlja vodstvo autoservisa te obavještava korisnika o stanju tog konkretnog sastanka. To može predstavljati informaciju da je popravak vozila završen te je spreman za preuzimanje ili informaciju o tome kada treba vozilo dovesti na servis. Polje „Status“ može sadržavati četiri različita tipa informacija: „U obradi“; „Termin predložen“, „Prijedlog poslan“ i „Dogovoreno“. Prvi tip informacije („U obradi“) zadani je tip informacije koji se prikazuju korisniku odmah prilikom kreiranja zahtjeva za sastanak, a označava procesiranje podataka. Nakon što autoservis pregleda podatke i predloži termin sastanka, mijenjaju se polja „Datum servisa“ i „Vrijeme servisa“ te se

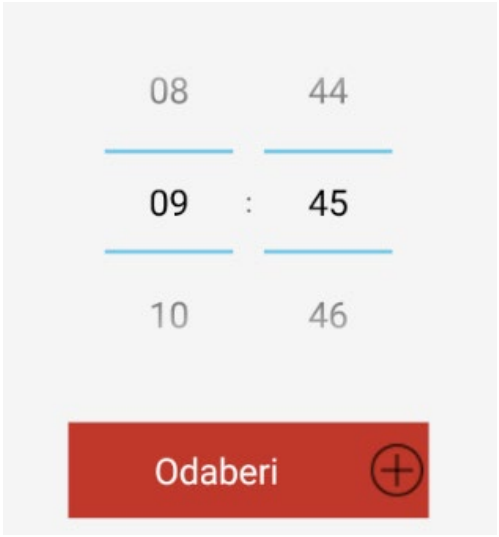
polje status mijenja u „Termin predložen“. Korisnik zatim klikom na gumb „Potvrdi ili izmijeni termin“ otvara dijalog s tri opcije: potvrdi, izmijeni i odustani.



Slika 51: Dijalog za potvrdu ili izmjenu sastanka

(Izvor: Snimka zaslona na računalu autora)

Dijalog na slici 51. izrađen je na već ranije opisan način. Klikom na gumb „Potvrdi“ odgovaramo potvrdno servisu na njihov prijedlog o datumu i vremenu servisa, a polje „Status“ se mijenja u „Dogovoreno“. Boja teksta polja „Status“ mijenja se u prikladnu zelenu boju te se onemogućuje gumb za potvrdu ili izmjenu termina; drugim riječima – sastanak je kreiran i korisnik više nema utjecaja na njega. U slučaju da nam datum i vrijeme ne odgovaraju, klikom na gumb “Predloži izmjenu“ imamo mogućnost sami predložiti datum i vrijeme koji nam odgovaraju. Klikom na navedeni gumb automatski se otvara prozor s kalendarom koji je već ranije korišten i prikazan na slici 31. Nakon što odaberemo datum i kliknemo „Dodaj datum“, otvara se prozor u kojemu je moguće odabrati vrijeme.

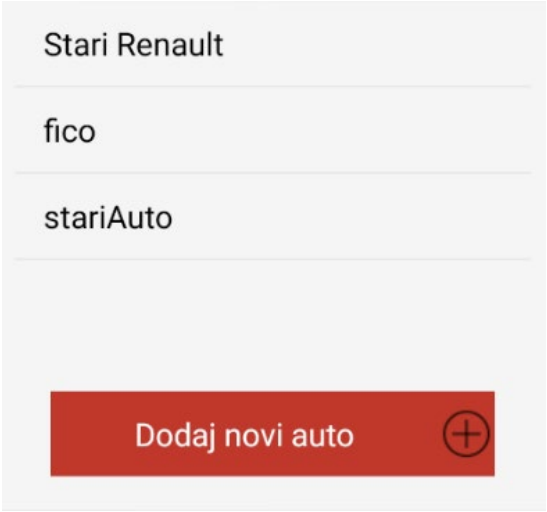


Slika 52: Fragment za odabir vremena

(Izvor: snimka zaslona na računalu autora)

Fragment za odabir vremena izrađen je na isti način kao i kalendar, uz razliku odabira „TimePicker“ elementa sučelja umjesto „CalendarView“ elementa. Klikom na gumb „Odaberi“ zatvara se fragment za odabir vremena te se otvara novi dijalog s dva gumba: „Odustani“ i „Potvrdi“. Klikom na odustani poništava se odabrano vrijeme i korisnik se vraća u aktivnost za upravljanje sastancima. Odabirom na drugi gumb, gumb „Potvrdi“, korisnik potvrđuje svoj odabir te se isti šalje u bazu podataka te postaje vidljiv autoservisu. Tekst polja „Status“ mijenja se u „Prijedlog poslan“ te autoservis ima opciju prihvatiti predložen zahtjev ili ga odbiti i predložiti ponovno neki drugi termin čime se polje „Status“ ponovno mijenja u „Termin predložen“ čime se proces ponovno započinje. Kako bi se izbjeglo ponavljanje zahtjeva, gumb za potvrdu ili izmjenu zahtjeva omogućen je samo u slučaju kada je servis poslao prijedlog termina te na njega treba odgovorit. U slučaju kada nema zahtjeva od strane servisa ili je korisnik poslao zahtjev za promjenom termina – gumb je onemogućen i prikazan prikladnom, sivom bojom. Time je završen proces upravljanja kreiranim sastankom, što je ujedno i ključna mogućnost aplikacije. Posljednji preostali gumb na glavnoj stranici „Moj auto“ predstavlja aktivnost za dodavanje vozila i informacija o vozilima.

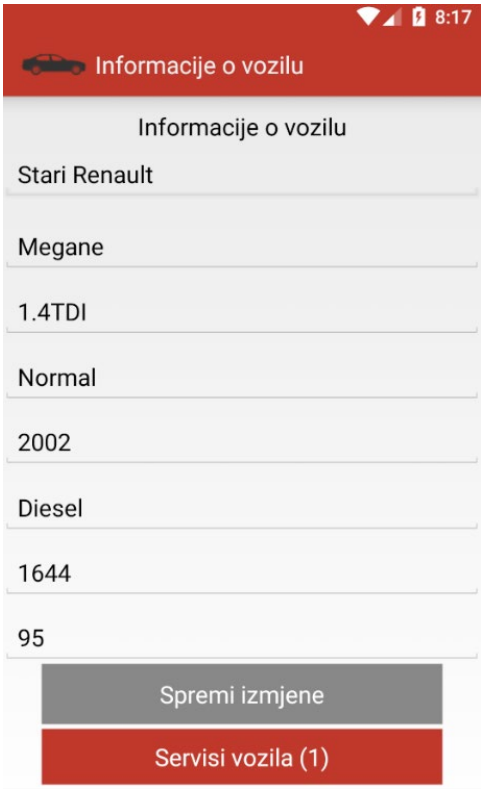
Dodirom na gumb „Moj auto“ otvara se fragment u kojem imamo mogućnost dodati svoje vozilo u bazu podataka, ali isto tako i pregled svojih vozila.



Slika 53: Fragment za pregled vozila
(Izvor: snimka zaslona na računalu autora

Dodirom na gumb „Dodaj novi auto“ otvara se fragment za dodavanje novog vozila. Fragment za dodavanje novog vozila prikazan je na slici 43. U slučaju da želimo vidjeti pojedinosti vozila, dodirom na neku od ponuđenih stavki u listi otvara se aktivnost za

prikaz detalja vozila gdje je moguće mijenjati podatke, ali i pogledati prethodne servise (sastanke) tog vozila. Nova aktivnost otvara se koristeći namjeru, a obavezno je prosljeđivanje informacije o nazivu vozila. Informacija se prosljeđuje koristeći metodu „putExtra()“ pomoću koje možemo prosljeđivati razne tipove podataka, u ovom slučaju tip podataka string. Tu informaciju preuzima aktivnost za prikaz pojedinosti o vozilu te na taj način dolazi do podatak o tom vozilu. Naziv vozila ne odnosi se na marku vozila, nego na „skraćeni naziv“ vozila kojeg smo sami odabrali prilikom dodavanja vozila u bazu podataka, a koji služi za razlikovanje vozila u bazi podataka. Može postojati više vozila s istim skraćenim nazivom, ali jedan korisnik može imati samo jedno vozilo s određenim nazivom. Aktivnost za prikaz detalja o vozilo sadrži, osim informacija o vozilu, dva gumba – „Spremi izmjene“ i „Servisi vozila“.



Slika 54: Aktivnost za prikaz informacija o vozilu

(Izvor: snimka zaslona na računalu autora)

Slika 54 prikazuje izgled aktivnosti za prikaz informacija o vozilu. Vidljivo je kako je gumb „Spremi izmjene“ sive boje – to označava da je taj gumb neaktivan i nije ga moguće aktivirati. Taj isti gumb postaje aktivna tek ukoliko dođe do promjene nekih od polja s podacima vozila, što je i logično budući da sprječava nepotrebno spajanje na bazu podataka kako bi se ažurirali podaci.

Promjena u crvenu boju postignuta je upravljanjem događajem na promjenu teksta:

```
1.      private void TipVozila_AfterTextChanged(object sender, Android.Text.AfterTextChanged
EventAr gs e) {
2.          saveCar.SetBackgroundColor(Android.Graphics.Color.Rgb(192, 57, 43));
3.          saveCar.Enabled = true;
4.      }
```

Drugi gumb, gumb „Servisi vozila“, otvara fragment za pregled prethodno kreiranih sastanaka (servisa) tog konkretnog vozila. Koristi se isti fragment kao na slici 49, jedina razlika je što se u prikazu filtriraju samo sastanci za to vozilo koje nas zanima. To je postignuto tako što se, prilikom pozivanja fragmenta, definira oznaka fragmenta (engl. Tag) kojom se daje do znanja kako se fragment otvara iz aktivnosti koja nije „MainActivity“ – što, dakle, znači da se otvara iz aktivnosti za prikaz informacija o vozilu budući da su to jedine dvije aktivnosti koje otvaraju fragment za pregled sastanaka.

```
1.      if (tag != "fromMainActivity") {
2.          var getOtherData = await firebase.Child("order").Child(id).Child(item.Key).OnceA
sync < orders > ();
3.          foreach(var Otheritem in getOtherData) {
4.              stringOrderID = Otheritem.Object.id;
5.              if (tag == Otheritem.Object.carName) {
6.                  ordersList.Add(new orders {
7.                      uid = Otheritem.Object.id, carName = Otheritem.Object.carName, datum
 = Otheritem.Object.datum.Substring(0, 10)
8.                  });  }   }  }
```

U gornjem kodu, na liniji broj 1, postavljen je uvjet koji traži potvrdu da fragment nije otvoren iz aktivnosti „MainActivity“. Ukoliko je zadovoljen taj uvjet, na liniji broj 5 nalazi se drugi uvjet, a koji uspoređuje je li naziv auta koji se nalazi u oznaci fragmenta jednak nazivu auta koji se trenutno provjerava u bazi podataka. Ukoliko je, taj se sastanak dodaje u listu „ordersList“. Na taj način filtriraju se sastanci i u listu se spremaju samo sastanci konkretnog vozila. Predaja liste u adapter za prikaz u listu stavki i ostatak procedure isti je kao i u slučaju pregleda sastanaka dodirom na gumb „Moji termini“. Gumb „Servisi vozila“ kao dio svog naziva sadrži broj unutar zagrade. Taj broj označava koliko kreiranih sastanaka ima to vozilo. To se postiže pozivanjem vlastite metode „getCarServiceCount()“ koja prvo dohvaća podatke o sastancima konkretnog korisnika istom procedurom kao u slučaju fragmenta za prikaz kreiranih sastanaka. Nakon dohvaćanja potrebnih podataka, dakle liste sa kreiranim sastancima

(„ordersList“), jednostavno se pomoću inkrementa i „foreach“ funkcije dobije broj sastanaka konkretnog korisnika i – još specifičnije – konkretnog vozila.

```
1.      foreach(var item in ordersList) {
2.          if (item.carName == carName.Text) {
3.              orderCount++;
4.          }
5.      }
```

5.5. Web aplikacija

Web aplikacija zamišljena je za upotrebu u autoservisu. Drugim riječima, zahtjevi poslani kroz mobilnu aplikaciju preuzimali bi se unutar web aplikacije te bi ih osoba zadužena za to pregledavala i odgovarala na iste. Web aplikacija zahtjeva prijavu prije korištenja. Registracija nije moguća, nego developer aplikacije kreira korisnika s email adresom u bazi podataka te ju predaje autoservisu na korištenje. Razlog tomu je što pristup upravljanju zahtjevima ne bi trebao biti omogućen nikomu osim točno dogovorenim osobama. Nakon prijave, korisnik ima uvid u zaprimljene zahtjeve na koje može odgovarati dodajući termin popravka, cijenu popravka i slično. Aplikacija je razvijana pomoću web tehnologija HTML i CSS, a uz pomoć programskog jezika JavaScript, tehnologije Node.js koja omogućava korištenje JavaScript jezika i na strani servera i na strani klijenta te uz pomoć Express frameworka koji automatski kreira serverski predložak i olakšava razvijanje aplikacije budući da korisnik ne mora kreirati serverske postavke.

5.5.1. Klijent – server model

Klijent - server model postoji od kada postoji i umrežavanja računala. Ovaj model predstavlja distribuiranu strukturu koja se sastoji od klijenta (korisnika) koji šalje zahtjev te od servera (pružatelja usluge) koji odgovara na zahtjev. U klijent – server arhitekturi, serverska komponenta pruža usluge većem broju klijentskih komponenata. Klijentske komponente zahtijevaju usluge od servera. Serveri su stalno aktivni i osluškuju ima li zahtjeva od klijenata. Zahtjevi se šalju komunikacijskim kanalima koji zavise od uređaja na kojima se server i klijent nalaze. Tipični primjer klijent – server arhitekture su aplikacije s udaljenim pristupom bazama podataka (kada klijentska aplikacija zahtjeva uslugu od servera baze podataka), udaljeni dokumenti ili web aplikacije. [30] Web

[30]V. Tomašević, Razvoj aplikativnog softvera, Beograd, Sveučilište Singidunum, 2012., str. 74

aplikacija, dakle, dijeli se na klijentsku i serversku stranu. Serverska strana aplikacije za upravljanje radom autoservisa uspostavljena je pomoću Express frameworka te koristeći Node.js. Klijentska strana uspostavljena je pomoću jezika HTML, CSS i JavaScript.

5.5.2. HTML i CSS

- HTML

HTML je kratica za HyperText Markup Language. HTML je jezik za označavanje kojim se određuje struktura, sadržaj i funkcija nekog HTML dokumeta (web – stranice). Pomoću HTML-a određuju se važni elementi svakog HTML – dokumenta kao što su slika, naslov, odlomak, poveznica i slično. Dakle, HTML je jezik za označavanje pomoću kojega se može odrediti struktura unutar HTML - dokumenta. Osim strukture elementa u dokumentu definirane HTML -om, CSS omogućava da se ti elementi stilski/grafički urede. Na taj se način web – pregledniku daje do znanja kako će stranica izgledati prilikom prikaza. HTML je u početku omogućavao jednostavno strukturirane web stranice - odlomke, prijelome redova i zaglavlja, a nije omogućavao unos grafike i multimedijalnih elemenata. HTML se razvijao, a da se pri tome nije pridavala pažnja oblikovanju i prezentaciji stranice, odnosno izgledu, jer je osnovni cilj bio strukturiranje podataka na jednostavan način. Međutim, pojavom vizualnih internetskih preglednika, a zbog brzog i nekontroliranog razvoja pojavili su se i problemi oko međusobne nepodrživosti preglednika na različitim platformama. Ti su problemi aktualni i danas, ali su sve manje izraženi. Svaka nova verzija HTML-a donosila je novosti i unapređenja te je predstavljala napredak. Ovaj napredak prate i internetski preglednici pa se u najnovijim preglednicima HTML kod ispravno prikazuje. Problem se javlja zbog toga što korisnici često nemaju instalirane najnovije verzije preglednika. Ovaj se problem rješava na način da se prilikom pisanja HTML koda pazi da on bude kompatibilan s internetskim preglednicima. Web stranicama pristupa se korištenjem web – preglednika, a da bi se neka stranica učitala u preglednik potrebno je upisati web adresu. Pod uvjetom da je korisnik spojen na Internet, preglednik šalje zahtjev web

poslužitelju, računalo na kojem se tražena stranica nalazi. Poslužitelj šalje pregledniku traženu stranicu, a preglednik zatim stranicu prikazuje korisniku. [31]

Web poslužitelj (Web server) je računalo na kojem se nalaze web stranice, koje se nalazi bilo gdje u svijetu i koje je uvijek spojeno na Internet kako bi moglo odgovoriti slanjem web stranice korisniku u bilo kojem trenutku.

Manje se stranice statičnije sadržaja još uvijek izrađuju pisanjem HTML i CSS koda. Veće stranice, pogotovo one koje se osvježavaju često, češće se izrađuju uz pomoć sustava za upravljanje web sadržajem, tj. Koristeći CMS (engl. Content Management System). CMS omogućuje osobi koja izrađuje web stranice korištenje nekih naprednih opcija i web tehnologija s manje znanja te jednostavnije održavanje sadržaja bez svakodnevnog susretanja s HTML-om i CSS-om. Međutim, znanje HTML-a uvelike pomaže čak i kod rada s CMS-om jer omogućuje bolje razumijevanje pozadinskih procesa te bolju kontrolu nad izgledom web-stranica. Za izradu web stranice u CMS - u uglavnom nije potrebno poznavanje programskih jezika i drugih web-tehnologija.[32]

Web stranice koje zahtijevaju obradu podataka i interakciju s korisnikom obično se ne izrađuju pomoću CMS-a zbog nedovoljne fleksibilnosti. Tako je, za izradu sličnih web stranica ili, prikladnije, web aplikacija, potrebno znanje nekoliko programskih jezika i tehnologija. Sustav za upravljanje radom autoservisa zahtijeva takvu aplikaciju i prema tome upotreba CMS-a nije moguća.

- CSS

CSS je jezik koji služi za oblikovanje web-stranica. Uz HTML, jezik pomoću kojeg se definiraju struktura i sadržaj web-stranica, CSS je osnovna tehnologija na kojoj se temelji današnji web. [33]

CSS je kratica za *Cascading Style Sheets.* Pojam style sheet često se upotrebljava za datoteku koja sadrži CSS-kod. Dakle, *style sheet* je datoteka koja definira stil, odnosno izgled web-stranice. Riječ *cascading* označava kaskadnu primjenu CSS-pravila. CSS-pravilo može se napisati tako da bude primijenjeno na sve elemente ili

[31] G. Kurtović, N. Katić, I. Jandrić, M. Kožul, Uvod u HTML, Zagreb, Srce, Sveučilište u Zagrebu, Sveučilišni računski centar, str. 4, https://www.srce.unizg.hr/tecajevi/popis-osnovnih-tecajeva/C201 (pristupljeno 24. svibnja 2018)
[32] Ibidem, st. 5
[33] E. Mujadžević, J. Plavac, G. Kurtović, J.N. Milić, Uvod u CSS, Zagreb, Srce, Sveučilište u Zagrebu, Sveučilišni računski centar, str. 3

samo na neke elemente ili da vrijedi samo za točno određeni element. Prije pojave CSS-a oblikovanje izgleda web stranice do određene razine bilo je moguće postići i u HTML-u. No, time se stvorio problem miješanja sadržaja i strukture s kodom čija je jedina svrha bila prezentacija. HTMML-kod za definiranje izgleda morao se ponaavljati iznova na svakom elementu i na svakoj stranici u web-sjedištu. Pojavo CSS-a nastoji se riješiti taj problem. Glavna ideja CSS-a je odvajanje prezentacijskog koda u zasebne datoteke i njegovo definiranje pomoću jednostavnih pravila koja se mogu odnositi na više elemenata odjednom. Prva verzija CSS-a definirana je krajem 1996. No do usvajanja CSS-a od strane autora web-sadržaja i proizvođača prošlo je još dosta vremena. Vrlo dugo web-preglednici nisu dosljedno implementirali CSS-specifikaciju pa se autori nisu mogli pouzdati da će stranice izgledati približno isto u svim preglednicima. Razvijeni su brojni trikovi CSS-u čija je namjena bila da isprave neočekivana ponašanja u nekim preglednicima. Današnja situacija je po tom pitanju puno bolja, iako se i danas savjetuje provjera izgleda stranice u što više preglednika. [34]

CSS-kod obično se piše odvojeno od HTML koda, tj. u zasebnoj datoteci. Da bismo HTML dokument povezali s CSS-datotekom, koristimo HTML-element link:

```
1.
< link rel = “stylesheet“ type = “text / css“ href = “folder / dokument.css“ / >
```

Kada se taj element koristi za uključivanje CSS-a, atribut rel mora imati vrijednost „stylesheet“, a atribut „type“ vrijednost „text/css“. Atribut „href“ postavljamo na putanju do CSS datoteke koju želimo uključiti. Element link mora se uvijek nalaziti unutar HTML-elementa „head“. Ako se HTML-stranica koristi većim brojem CSS-datoteka, uključit će se tako da se element link navede više puta. Mogući su i načini pisanja CSS koda u samoj HTML-datoteci koristeći HTML-element „style“.[35]

5.5.3. Node.js

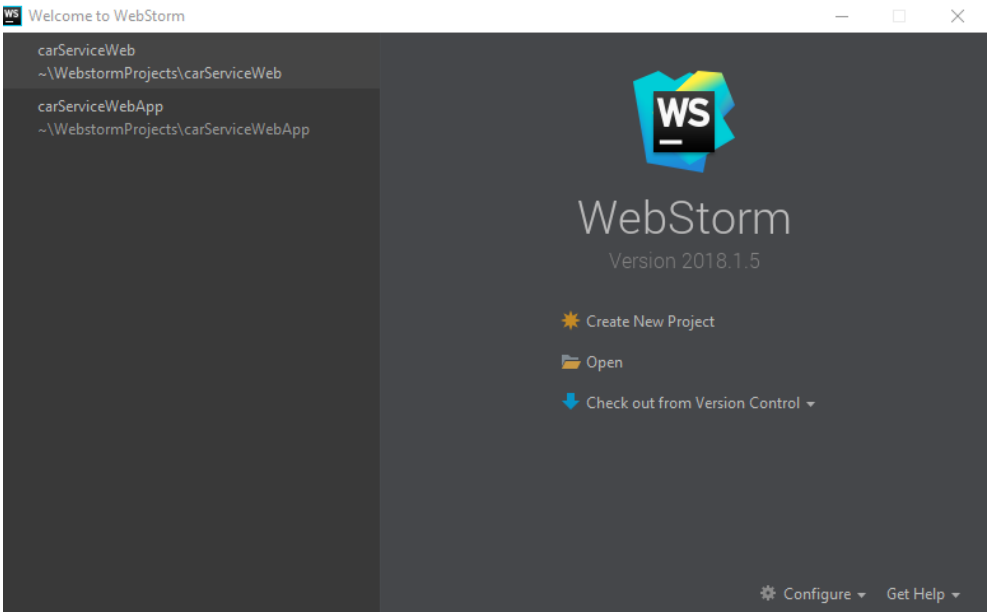
Node.js je softver otvorenog koda; višeplatformska (Engl. Cross-platform) JavaScript okolina za izvođenje JavaScript koda izvan Internet preglednika. U početku, JavaScript je upotrebljavan za skriptiranje na klijentskoj strani te se kod izvršao u Internet

[34] Loc.Cit.
[35] Ibidem, str.4

pregledniku. Node.js omogućava korisnicima upotrebu JavaScipta izvan preglednika, odnosno omogućeno im je pisati alate u komandnoj liniji te skriptiranje na serverskoj strani, odnosno, drugim riječima, Node.js omogućava upotrebu JavaScripta kako na klijentskoj tako i na serverskoj strani, što znači da omogućava unificiran razvoj web aplikacija koristeći samo jedan programski jezik. [36] Upravo ta unificiranost predstavlja najveću prednost Node.js jezika u odnosu na alternative web razvoja od kojih većina kombinira dva jezika, jedan za serversku stranu jedan za klijentsku stranu poslužitelja.

5.5.4. Izrada web aplikacije

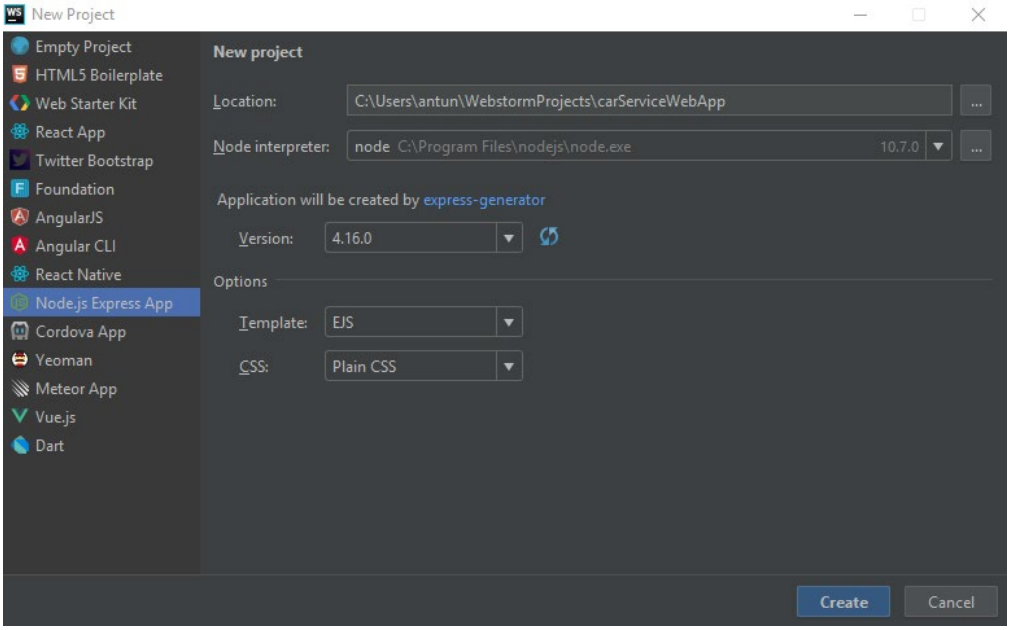
Prilikom otvaranja WebStorm razvojnog okruženja otvara se početni prozor u kojemu možemo otvoriti postojeći projekt, kreirani novi ili provjeriti na sustavu za upravljanje razvojnim kodom.



Slika 55: Početni prozor WebStorm razvojnog okruženja
(Izvor: snimka zaslona na računalu autora)

Klikom na „Create New Project“ otvara se novi prozor, prozor za kreiranje novog projekta. Unutar prozora odabran je „Node.js Express App“ koji uključuje Express Framework. Potrebno je još definirati naziv projekta te odabrati „EJS“ pod „Template“ opcijom.

[36] P. Teixera, Professional Node.js : Building Javascript – Based Scalable Software, Inianapolis, Jon Wiley & Sons, 2013, str. 16



Slika 56: Prozor za novi projekt u WebStorm programskom okruženju
(Izvor: snimka zaslona na računalu autora)

Nakon odabranih stavki i kreiranja projekta, otvara se glavni prozor u razvojnom okruženju te prikazuje projekt i njegove komponente. S lijeve strane prikazan je projekt, a s desne strane prozor za upis koda u pojedinu datoteku. Pomoću Express Frameworka kreirane su datoteke „www“, „error.ejs“, „Indeks.ejs“ i „result.ejs“ te „app.js“. Unutar datoteke „www“ postavljene su serverske postavke web aplikacije, a u preostale tri datoteke nalazi se osnovne funkcije za prikaz sadržaja korisniku. Posljednja datoteka, „app.js“, služu kao glavna konfiguracijska datoteka. Prema zadanim postavkama korisnika dočeka poruka „Hello world“ prilikom otvaranja web aplikacije. Korisnik je spreman za kreiranje izgleda početne stranice prema svojim zahtjevima te oblikovanjem aplikacije prema utvrđenim zahtjevima.

Unutar „index.ejs“ datoteke definira se izgled početne stranice, ovom slučaju stranice za prijavu u web aplikaciju. Prvi dio datoteke definira neke osnovne HTML elemente kao što su „DOCTYPE“ i „head“.

```
1.      < !DOCTYPE html >
2.       < html >
3.       < head >
4.       < title > Moj servis prijava < /title>
5.       < head / >
6.       < html >
```

Time je definiran tip dokument (HTML) te naslov stranice. Zatim su definirani ostali (osnovni) elementi stranice – polja za upis korisničkog imena i lozinke te gumb za prijavu.

```
1.      < div class = "login-page" >
2.       < div class = "form" >
3.       < form id = "form1"
4.      method = "post"
5.      class = "login-form" >   < input name = "username"
6.      type = "text"
7.      placeholder = "Korisničko ime" / >   < input name = "password"
8.      type = "password"
9.      placeholder = "Lozinka" / >
10.      < button onclick = {
11.         signIn()  } > login < /button>
12.        <p id="passwordMessage" class="message">Neispravno korisn
ičko ime ili lozinka </p >
13.     < /form>
14.     < /div>
```

Kao što vidimo u kodu gore, osim dodavanja elemenata HTML koda, upotrebom JavaScripta registriran je događaj na klik na gumb za prijavu („login“). Registriran je na način da se klikom na gumb za prijavu pokreće metoda „signIn()“, koja je definirana u istom dokumentu:

```
1.  function signIn() {

2.      var form = document.getElementById("form1");
3.      var email = form[0].value;
4.      var password = form[1].value;
5.      var falsePasswordMessage = document.getElementById("passwordMessage");
6.      var idToken;
7.
8.      var authProvider = new firebase.auth.EmailAuthProvider
     firebase.auth().signInWithEmailAndPassword(email, password)
    .then(function(data)
9.          var notAdmin = "";
10.         idToken = data.user.uid;
11.         localStorage.setItem('id', idToken);
12.         sessionStorage.setItem('userType', notAdmin)
     var database = firebase.database();
13.         var Ref = firebase.database().ref('users/' + data.user.uid + '/' + 'typeNode
    /' + '/');
14.
15.         Ref.on('value', function(snapshot)
16.             if (snapshot.val().type == "admin") {
17.                 window.location.href = 'main.ejs';
18.                 sessionStorage.setItem('userType', snapshot.val().type == "admin") }
19.         }).catch(function(error) {
20.             firebase.auth().signOut();
21.             falsePasswordMessage.style.display = 'block';
22.         })
23.     }).catch(function(error)
24.         console.log(error)
     falsePasswordMessage.style.display = 'block';
25.     })

26. }
```

Unutar metode kreirane su varijable koje preuzimaju tekst koji je korisnik upisao i taj tekst šalju metodi za prijavu u Firebase. Ta prijava se obavlja pomoću „signInWIthEmailAndPassword()“ metode koja dolazi uz „firebase“ klasu. S obzirom na to da se u web aplikaciju ne smije moći prijaviti nitko osim posebno definiranih korisnika, postavljena je provjera. Točnije, korisnik koji je predefiniran za korištenje web aplikacije u bazi podataka pod svojim unikatnim ključem sadrži oznaku „admin“. Upravo zbog tog je, nakon uspješne autentifikacije, izvršena provjera korisnika, odnosno sadrži li on atribut „admin“ u bazi podataka. Ukoliko korisnik nema atribut „admin“ njemu nije dozvoljena prijava te je automatski odjavljen iz sustava. Kako bi stranica poprimila određeni izgled, potrebno je povezati CSS dokument s „indeks.ejs“ dokumentom i dizajnirati stranicu prema utvrđenim zahtjevima. Postavke stranice postavljene su prema sljedećem:

```
1.      .login - page {
2.          width: 360 px;
3.      padding: 15 % 0 0;
4.      margin: auto;
5.      }
```

Prema kodu, vidljivo je kako je širina stranice postavljena na 360px, uz „padding“ od 15% te su postavljene automatske margine.

Polja za unos podataka postavljena su tako da je određen font „sans – serif“ te pozadina sive boje. Postavljene su također i postavke širine, margina i slično.

```
1.      .form input {
2.          font - family: "Roboto", sans - serif;
3.          background: #f2f2f2;
4.          width: 100 % ;
5.          margin: 0 0 15 px;
6.          padding: 15 px;
7.          box - sizing: border - box;
8.          font - size: 14 px;
```

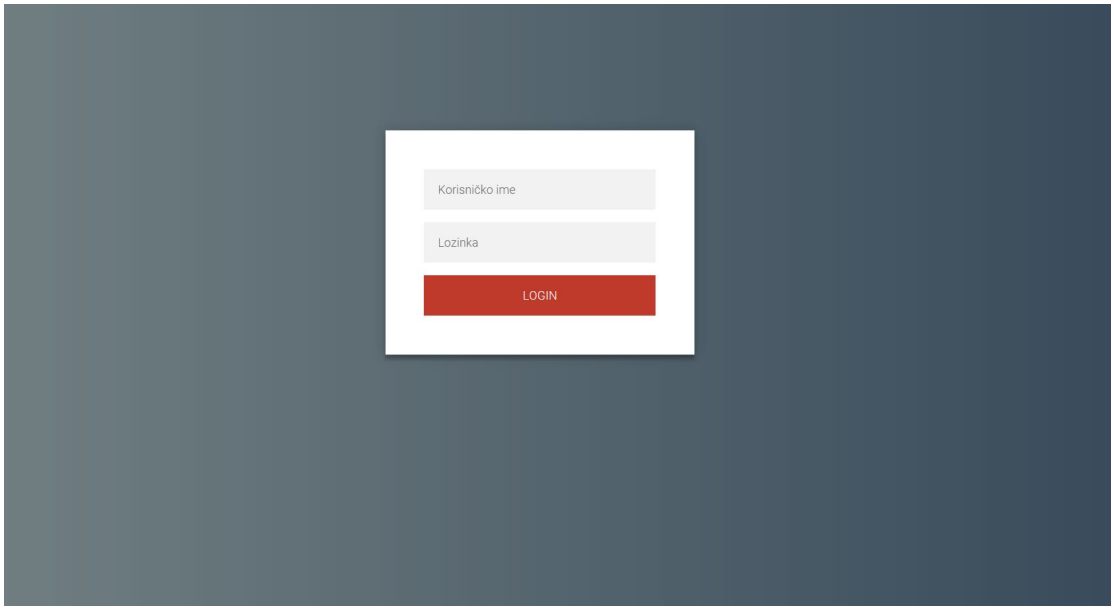
Element gumb postavljen je na isti način. Na kraju, element „body“ definiran je tako da se boje miješaju stvarajući na taj način elegantan, jednostavan i pregledan dizajn.

```
1.      body {
2.          background: -webkit - linear - gradient(right, #76b852, # 8 DC26F);
3.      background: -moz - linear - gradient(right, #c0392b, #8e44ad);
4.      background: -o-linear-gradient(right, # c0392b, #c0392b);
5.      background: linear - gradient(to left, #2c3e50, # 7 f8c8d);
6.      font - family: "Roboto",
7.        sans - serif;
8.       - webkit - font - smoothing: antialiased;
9.       - moz - osx - font - smoothing: grayscale;
10.     }
```

Poruka koja korisniku javlja kako je korisničko ime ili lozinka neispravno definirana je na način da se ne vidi prilikom otvaranja stranice. U slučaju da je vidljiva, definirana joj je crvena boja.

```
.form.message {
    margin: 15 px 0 0;
color: #C0392b;
font - size: 15 px
display: none;  }
```

Poruka nije prikazana prilikom otvaranja stranice iz razloga jer treba biti skrivena sve dok korisnik ne pogriješit korisničko ime ili lozinku. Prikazivanje poruke definirano je naredbom „falsePasswordMessage.style.display = 'block';“ nakon neuspješne prijave korisnika.

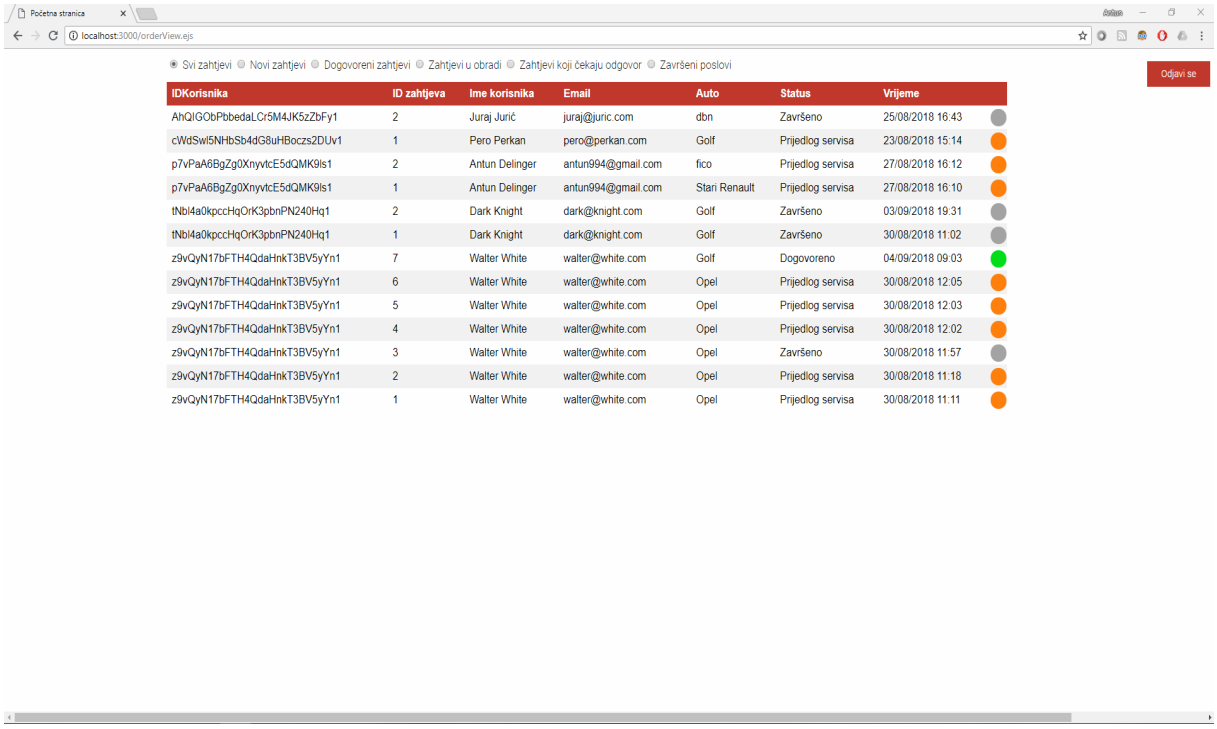


Slika 57: Izgled stranice za prijavu
(Izvor: snimka zaslona na računalu autora)

S obzirom da bi korisnik, prema zahtjevima, trebao biti u mogućnosti, nakon zatvaranja prozora i prilikom ponovnog pokretanja, automatski se prijaviti i otvoriti početnu stranicu, potrebna je provjera je li korisnik prijavljen u sustav od ranije. Koristeći sljedeću funkciju izvršena je provjera i automatsko preusmjeravanje korisnika na početnu stranicu.

```
function checkIfLoggedIn() {
    var type = sessionStorage.getItem('userType');
    firebase.auth().onAuthStateChanged(function(user) {
        if (user && type == 'admin') {
           document.body.style.display = 'block'
           currentUserId = user.uid;}
        else { logOut();   } }) }
```

Nakon prijave u sustav, otvara se početna stranica. Početna stranica osmišljena je kao stranica za prikaz svih do sad zaprimljenih zahtjeva u obliku tablice s mogućnošću filtriranja. Na slici 58 prikazan je izgled početne stranice.



IDKorisnika	ID zahtjeva	Ime korisnika	Email	Auto	Status	Vrijeme
AhQlGObPbbedaLCr5M4JK5zZbFy1	2	Juraj Jurić	juraj@juric.com	dbn	Završeno	25/08/2018 16:43
cWdSwI5NHbSb4dG8uHBoczs2DUv1	1	Pero Perkan	pero@perkan.com	Golf	Prijedlog servisa	23/08/2018 15:14
p7vPaA6BgZg0XnyvtcE5dQMK9ls1	2	Antun Delinger	antun994@gmail.com	fico	Prijedlog servisa	27/08/2018 16:12
p7vPaA6BgZg0XnyvtcE5dQMK9ls1	1	Antun Delinger	antun994@gmail.com	Stari Renault	Prijedlog servisa	27/08/2018 16:10
tNbl4a0kpccHqOrK3pbnPN240Hq1	2	Dark Knight	dark@knight.com	Golf	Završeno	03/09/2018 19:31
tNbl4a0kpccHqOrK3pbnPN240Hq1	1	Dark Knight	dark@knight.com	Golf	Završeno	30/08/2018 11:02
z9vQyN17bFTH4QdaHnkT3BV5yYn1	7	Walter White	walter@white.com	Golf	Dogovoreno	04/09/2018 09:03
z9vQyN17bFTH4QdaHnkT3BV5yYn1	6	Walter White	walter@white.com	Opel	Prijedlog servisa	30/08/2018 12:05
z9vQyN17bFTH4QdaHnkT3BV5yYn1	5	Walter White	walter@white.com	Opel	Prijedlog servisa	30/08/2018 12:03
z9vQyN17bFTH4QdaHnkT3BV5yYn1	4	Walter White	walter@white.com	Opel	Prijedlog servisa	30/08/2018 12:02
z9vQyN17bFTH4QdaHnkT3BV5yYn1	3	Walter White	walter@white.com	Opel	Završeno	30/08/2018 11:57
z9vQyN17bFTH4QdaHnkT3BV5yYn1	2	Walter White	walter@white.com	Opel	Prijedlog servisa	30/08/2018 11:18
z9vQyN17bFTH4QdaHnkT3BV5yYn1	1	Walter White	walter@white.com	Opel	Prijedlog servisa	30/08/2018 11:11

Slika 58: Početna stranica web aplikacije
(Izvor: snimka zaslona na računalu autora)

S obzirom da su svi važni elementi početne stranice vezani uz tablicu, najvažnija metoda, preko koje su dohvaćeni svi podaci, je metoda „createTable()“ pomoću koje se kreira tablica sa zadanim naslovima te se uzimaju podaci iz baze podataka i popunjavaju redovi. Prikazan je kod metode „createTable()“, ali u dijelovima zbog preglednosti i jednostavnosti.

```
1.      var database = firebase.database();
2.      var Ref = firebase.database().ref('order/');
3.       var fragment = document.createDocumentFragment();
4.      var table = document.createElement("table");
5.
6.      Ref.on('value', function(snapshot) { //get userID from orders
7.                  listOfOrders = snapshot.val();
8.                  var headerList = ["IDKorisnika", "ID zahtjeva", "Ime korisnika", "Email"
, "Auto", "Status", "Vrijeme", ""];
9.                  for (var i = 0; i < headerList.length; i++) {
10.                     var th = document.createElement("th");
11.                     th.textContent = headerList[i];
12.                     table.appendChild(th);  }
13.                 fragment.appendChild(table);
```

U prvom dijelu koda definirane su varijable za dohvaćanje podataka iz baze podataka („database“ i „Ref“), a zatim su kreirane varijable za kreiranje tablice („fragment“ i „table“). Za kraj, Dodijeljeni su nazivi zaglavlja i raspoređeni su u kasnije kreiranu tablicu. S obzirom da su u tablici potrebni podaci o korisniku, podaci o zahtjevu te podaci o vozilu, potrebno nam je pristupiti tri različite tablice u bazi podataka, a sve u jednoj petlji kako bi se mogla tablica popuniti tim istim podacima. Prvo su dohvaćeni podaci o broju određenog zahtjeva („childSnapshot.key“):

```
1.      snapshot.forEach(function(childSnapshot) {
2.                  var RefTwo = firebase.database().ref('order/' + childSnapshot.key + '/')
;
3.                  RefTwo.once('value', function(snapshotTwo) { //get number of order from
orders
4.                              listOfOrders = snapshot.val();
5.                              snapshotTwo.forEach(function(childSnapshotTwo) {
```

Nakon dohvaćanja podatka u broju zahtjeva moguće je pristupiti pojedinom zahtjevu u bazi i prema tome dohvatiti podatke koji su potrebni, u ovom slučaju to su status zahtjeva i datum kreiranja zahtjeva:

```
1.      var RefNew = firebase.database().ref('order/' + childSnapshot.key + '/' + childSnaps
hotTwo.key + '/');
2.      RefNew.once('value', function(snap) {
3.                  snap.forEach(function(snapChild) {
4.                              orderStatus = snapChild.val().status;
5.                              timeInfo = snapChild.val().datumKreiranja;
```

Nastavljajući u istoj petlji, pristupa se podacima o korisniku u bazi podataka, točnije preuzima se ime, prezime i email konkretnog korisnika:

```
1.      var RefThree = firebase.database().ref('users/' + childSnapshot.key + '/'); //get us
er info from users
2.      RefThree.once('value', function(snapshotThree) {
3.                  snapshotThree.forEach(function(childSnapshotThree) {
4.                              userName = childSnapshotThree.val().name + " " + childSnapsh
otThree.val().lastName;
5.                              userEmail = childSnapshotThree.val().email
```

Nakon što su preuzeti svi potrebni podaci za konkretnog korisnika, potrebno je te podatke dodijeliti u tablicu. Kao što je vidljivo, dodijeljeni su podaci korisnika u varijablu „trValues“ koja predstavlja vrijednost pojedinog retka tablice. Osim toga, potrebno je programirati ponašanje aplikacije nakon klika na pojedini redak, odnosno programirati aplikaciju da otvori novi prozor s podacima o odabranom zahtjevu i korisniku.

```
1.
var trValues = [childSnapshot.key, childSnapshotTwo.key.substr(1, 1), userName, userEmail,
snapChild.val().carName, snapChild.val().status, snapChild.val().datumKreiranja];
2.      for (var i = 0; i < trValues.length; i++) {
3.          var td = document.createElement("td");
4.              sessionStorage.setItem('userID', trValues[0]);
5.              sessionStorage.setItem('orderID', trValues[1]);
6.              sessionStorage.setItem('carName', trValues[4]);
7.              window.location.href = '/orderView.ejs';
8.          }
```

Nakon što je, pomoću klase „sessionStorage“, u lokalnu memoriju postavljena informacija o korisniku (koja će kasnije preuzeti u novoj stranici za prikaz tog istog korisnika), dovršava se kreiranje tablice. S obzirom na potrebu za filtriranjem tablice, to je sljedeće što je izvedeno i to tako što pozivanje metode „createTable()“ zahtijeva parametar tipa string koji govori što prikazati u tablici, a može biti oblika: „showAll“, „showNewRequested, „showDeal, „showProcessing“ te „showUserSuggestion“, a koji (redom) predstavljaju prikaz svih, novih, završenih, dogovorenih, zahtjeva u obradi te zahtjeva koji traže reakciju korisnika.

```
1.      if (filterData == 'showAll') {
2.          td.textContent = trValues[i];
3.          tr.appendChild(td) tr.appendChild(dot) table.appendChild(tr);
4.      }
5.      if (filterData == 'showNewRequirements') {
6.          if (trValues[5] != 'Kreirano') return;
7.          td.textContent = trValues[i];
8.          tr.appendChild(td) tr.appendChild(dot) table.appendChild(tr);
9.      }
```

Zbog pojednostavljenja i preglednosti prikazana su samo prva dva slučaja. Prvi slučaj je nešto drugačiji, dok su preostali slučaji riješeni na isti način pa stoga nema potrebe za ponavljanjem prikaza svih slučaja. Prvi slučaj predstavlja slučaj kada treba prikazati sve zahtjeve, to je slučaja kada je proslijeđen parametar „showAll“, a to se događa kad korisnik odabere gumb u aplikaciji „Svi zahtjevi“. U tom slučaju, dakle, nema nikakvog filtriranja i prikazuju se svi zahtjevi koji su pronađeni u bazi podataka. U drugom slučaju, slučaju kada postoji potreba za filtriranjem – u ovom slučaju prikaz svih novih, tek kreiranih zahtjeva – vrši se provjera je li status trenutno provjeravanog zahtjeva jednak tekstu „Kreirano“, ukoliko je taj zahtjev se prikazuje u tablici, no, ukoliko nije jednak spomenutom tekstu, taj zahtjev se ignorira. Na ovaj način je riješeno filtriranje i drugih slučajeva. Konačno, podaci o tablici i sama tablica predaju se dokumentu

(„document.body.appendChild(fragment)“) što znači prikazivanje krajnjem korisniku. Time se zatvaraju sve otvorene petlje i završava metoda „createTable()“.

IDKorisnika	ID zahtjeva	Ime korisnika	Email	Auto	Status	Vrijeme
AhQIGObPbbedaLCr5M4JK5zZbFy1	2	Juraj Jurić	juraj@juric.com	dbn	Završeno	25/08/2018 16:43
tNbl4a0kpccHqOrK3pbnPN240Hq1	2	Dark Knight	dark@knight.com	Golf	Završeno	03/09/2018 19:31
tNbl4a0kpccHqOrK3pbnPN240Hq1	1	Dark Knight	dark@knight.com	Golf	Završeno	30/08/2018 11:02
z9vQyN17bFTH4QdaHnkT3BV5yYn1	3	Walter White	walter@white.com	Opel	Završeno	30/08/2018 11:57

Slika 59: Prikaz filtriranja podataka u tablici
(Izvor: snimka zaslona na računalu autora)

Na slici 59 je prikazan primjer filtriranja podataka, a koji funkcionira tako da korisnik odabere jedan od ponuđenih gumba i prema tome filtrira podatke. Dodatno, kao što je vidljivo na slikama 58 i 59 kreirane su točke u boji od kojih svaka označava neki od statusa zahtjeva: žuta označava novi, zelena dogovoreni, siva završeni zahtjev, a narančasta zahtjev u obradi. Sve tablice u aplikaciji su na sličan način dizajnirane pomoću CSS koda. Konkretno, tablica za prikaz popisa zahtjeva je dizajnirana na sljedeći način.

```
1.      table {
2.          position: relative;left: 250 px;top: 20 px;font - family: arial,
3.          sans - serif;border - collapse: collapse;border - color: #FFFFFF;width: 70 % ;}
4.      tr: nth - child(even) {background - color: #f2f2f2;}
5.      td, th {text - align: left; padding: 8 px;}
6.      th {background - color: #c0392b;color: #FFFFFF;;  }
7.      tr: hover {  background - color: #ddd;}
8.      tr {   cursor: hand;cursor: pointer;}
```

Kao što je već spomenuto, klik na neki redak u tablici otvara stranicu („orderView“) za pregled podataka i odgovaranje na zahtjev. Stranica sadrži tri tablice te jednu formu s nekoliko ulaznih varijabli za slanje odgovora. Osim toga, sadrži još i četiri gumba – povratak na početnu stranicu, odjava, označi zahtjev kao „dogovoreno“, označi zahtjev kao „završeno“ te „pošalji“ gumb. Sadržaj stranice je složen jedan iza drugog, što znači da ne stane sve na stranicu, nego je potrebno spustiti stranicu dolje kako bi se vidio cjelokupan sadržaj. Tablice su kreirane pomoću HTML-a, a ne JavaScripta kao što je to slučaj s tablicom na početnoj stranici. To je zbog toga što redovi tablica nisu generirani ovisno o podacima u bazi podataka, nego je broj redova fiksan i unaprijed je poznato koliko informacija tablica sadržava.

```
1.      < div class carView > < div id = "carForm"
2.      method = "post"
3.      class = "car-
Form" > < table class = "carTable" > < h1 > Podaci o vozilu < /h1> < th > Podaci < /th> < t
h > Upis korisnika < /th> < tr > < td align = "left" > ID vozila</td> < td id = 'IDVozila'
4.      align = "left" > < /td> < /tr> < tr > < td align = "left" > Marka vozila: < /td> < t
d id = 'markaVozila'
5.      align = "left" > < /td> < /tr> < tr > < td align = "left" > Model vozila: < /td> < t
d id = 'modelVozila'
6.      align = "left" > < /td> < /tr> < tr > < td align = "left" > Tip vozila: < /td> < td
id = 'tipVozila'
7.      align = "left" > < /td> < /tr> < tr > < td align = "left" > Godina: < /td> < td id =
 'godina'
8.      align = "left" > < /td> < /tr> < tr > < td align = "left" > Tip motora: < /td> < td
id = 'tipMotora'
9.      align = "left" > < /td> < /tr> < tr > < td align = "left" > Snaga motora: < /td> < t
d id = 'snagaMotora'
10.     align = "left" > < /td> < /tr> < /table> < /div> < /div>
```

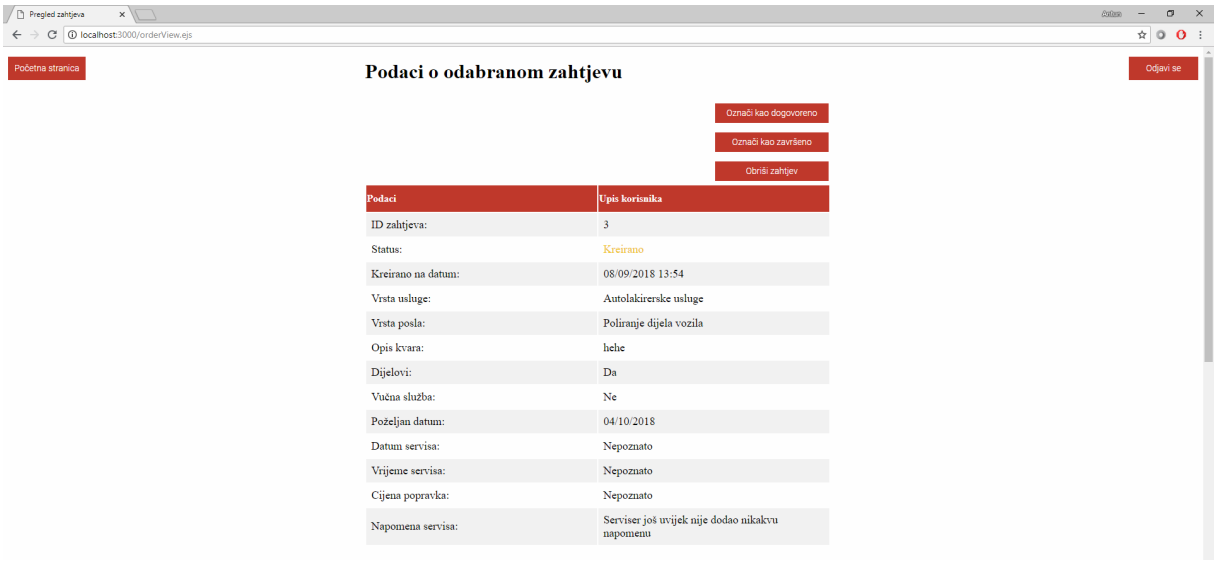
Dohvat informacija iz baze podataka jednostavan je iz razloga što sve informacije već postoje iz prošle stranice – unikatni ključ korisnika, unikatni ključ vozila te broj zahtjeva. Te informacije dostupne su nam iz lokalne memorije budući da smo, kao što je prethodno objašnjeno, iste spremili kako bi bile spremne za upotrebu u novoj stranici. Dohvat podataka je izvršen metodom „getItem()“.

```
1.      var userID = sessionStorage.getItem('userID');
2.      var orderID = '"' + sessionStorage.getItem('orderID') + '"';
3.      var carName = sessionStorage.getItem('carName');
```

To je olakšalo dohvaćanje podataka i skratilo pisanje koda. Također, olakšavajuću okolnost predstavlja i činjenica da je poznat broj redova u svakoj tablici. Pomoću metode „getCarData()“, primjerice, dohvaćeni su podaci o vozilu. Rezultat je prikazan na slici ??.

```
1.      function getCarData() {
2.          var database = firebase.database();
3.          var Ref = firebase.database().ref('car/' + userID + '/' + carName);
4.          Ref.on('value', function(snapshot) {
5.              IDVozila.textContent = carName;
6.              markaVozila.textContent = snapshot.val().markaVozila;
7.              modelvozila.textContent = snapshot.val().modelVozila;
8.              tipVozila.textContent = snapshot.val().tipVozila;
9.              godina.textContent = snapshot.val().godina;
10.             tipMotora.textContent = snapshot.val().godina;
11.             snagaMotora.textContent = snapshot.val().snagaMotora;
12.         })
13.     }
```

Na isti način kreirane su i druge tablice na stranici, a i dohvat podataka je izvršen na isti način.



Slika 60: Prikaz prvog dijela stranice za prikaz podataka o zahtjevu
(Izvor: snimka zaslona na računalu autora)

Na slici 60 prikazan je prvi dio zaslona koji korisnik vidi nakon otvaranja stranice za pregled informacija o zahtjevu. Vidimo kako stranica dosljedno prati dizajn mobilne aplikacije i prethodnih stranica web aplikacije – jednostavnost i minimalizam te kombinacija crvene i bijele boje.

U lijevom gornjem kutu nalazi se gumb za povratak na početnu stranicu, dok je na suprotnu stranicu pozicioniran gumb za odjavu. Ispod se nalaze tri gumba – gumb „Označi kao dogovoreno“, gumb „Označi kao završeno“ i gumb „Obriši zahtjev“. Prvi gumb, gumb „Označi kao dogovoreno“, nakon klika mijenja zahtjev statusa na „Dogovoreno“ što bi u praktičnoj upotrebi trebalo biti korišteno u slučaju kada korisnik predloži termin. Također, gumb je namijenjen za situacije kada klijent i autoservis postignu dogovor nekim drugim putem (primjerice telefonski) te time označavaju status „Dogovorenim“.

Drugi gumb, gumb „Označi kao završeno“, sukladno nazivu, mijenja status u „Završeno“ i time zatvara mogućnost uređivanja zahtjeva. Namijenjen je za slučaj kada je popravak na automobilu završen.

Treći gumb, očekivano, briše otvoreni zahtjev iz baze podataka te korisnika vraća na početnu stranicu.

Podaci o korisniku

Podaci	Upis korisnika
Ime:	Antun
Prezime:	Delinger
Email:	antun994@gmail.com
Broj telefona:	4524352432
Mjesto:	Komletinci
Ulica i broj:	Brace Radic
ID korisnika:	p7vPaA6BgZg0XnyvtcE5dQMK9ls1

Slika 61: Prikaz tablice s podacima o korisniku
(Izvor: snimka zaslona na računalu autora)

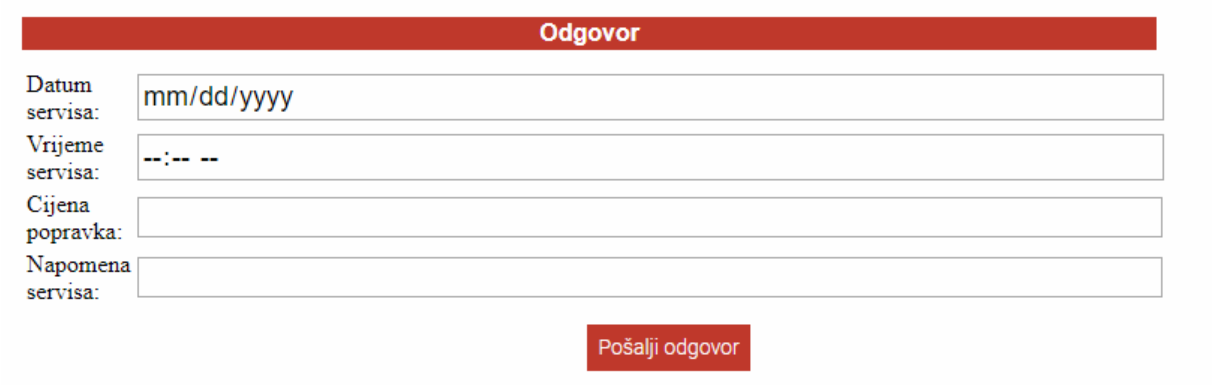
Slika 61 prikazuje tablicu s podacima o korisniku. Tablica sadrži sve informacije o korisniku koje se nalaze u bazi podataka, odnosno ono što bi autoservisu bilo potrebno za poslovanje s klijentom.

Podaci o vozilu

Podaci	Upis korisnika
ID vozila	Stari Renault
Marka vozila:	Megane
Model vozila:	Normal
Tip vozila:	1.4TDI
Godina:	2002
Tip motora:	2002
Snaga motora:	95

Slika 62: Prikaza tablice s podacima o vozilu
(Izvor: snimka zaslona na računalu autora)

Slika 62 prikazuje tablicu s podacima o vozilu koje je korisnik mobilne aplikacije spremio u bazu podataka. Podaci su od ključne važnosti za kvalitetnu pripremu posla od strane autoservisa.



Slika 63: Prikaz forme za unos odgovora na zahtjev
(Izvor: Snimka zaslona na računalu autora)

Konačno, slika 63 prikazuje formu za unos odgovora na zahtjev što ujedno predstavlja najvažniju komponentu stranice. Prvi element stranice je tip „Date“ koji nudi predefinirani kalendar. Isto vrijedi za element „Vrijeme servisa“ koji nudi predefinirani izbornik za odabir vremena. Element „Cijena popravka“ predstavlja tip podataka „Text“, isto kao i element „Napomena servisa“. Njihova implementacija vidljiva je u kodu dolje.

```
1.      < div class = "userInputPart" > < h1 > Odgovor < /h1> < table > < tr > < td align =
"left" > Datum servisa: < /td> < td align = "left" > < input id = "date"
2.      type = "date"
3.      format = "2014-12-
10" / > < /td> < /tr> < tr > < td align = "left" > Vrijeme servisa: < /td> < td align = "le
ft" > < input id = "time"
4.      type = "time" / > < /td> < /tr> < tr > < td align = "left" > Cijena popravka: < /td>
 < td align = "left" > < input id = "cijenaInput"
5.      type = "text" / > < /td> < /tr> < tr > < td align = "left" > Napomena servisa: < /td
> < td align = "left" > < input id = "inputNapomena"
6.      type = "text" / > < /td> < /tr> < /table> < button id = "sendReplyButton"
7.      onclick = {sendImportedData(status)} > Pošalji odgovor < /button> < /div>
```

Klikom na gumb „Pošalji poruku“ pozivamo metodu „sendImportedData()“ koja, kao što naziv sugerira, podatke koje je korisnik dodao upisuje u bazu podataka. Podaci se ažuriraju tako što se cijela tablica podataka u bazi mijenja s podacima koji su u tablici, a na mjesto „datum“, „vrijeme“; „cijena“ i „napomena servisera“ sprema se unos koji je korisnik dodao u polje za upis teksta. S obzirom da je podatak o potrebi za vučnom službom i podataka o potrebi za dijelovima tipa Boolean, ti podaci su prije upisa u tablici pretvoreni u jednostavan tekst „Da“, odnosno „Ne“, ovisno o tome je li zapis imao vrijednost točno ili netočno. Tako je u slučaju ažuriranja podataka bilo potrebno pretvoriti tekstualan podatak koji nosi vrijednost „Da“ ili „Ne“ ponovno u tip podataka Boolean kako bi se spremilo u bazu podataka.

```
function sendImportedData(status) {
    var database = firebase.database();
    var vucnaBool = true;
    var dijeloviBool = true;
    if (vucnaSluzba.textContent == "Da") vucnaBool = true;
    if (vucnaSluzba.textContent == "Ne") vucnaBool = false;
    if (dijelovi.textContent == "Da") dijeloviBool = true;
    if (dijelovi.textContent == "Ne") dijeloviBool = false;
    var order = {
        carName: IDVozila.textContent,
        cijena: inputCijena.value,
        datumKreiranja: datumKreiranja.textContent,
        datumServisa: inputDate.value,
        dijelovi: dijeloviBool,
        id: IDzahtjeva.textContent,
        napomenaServisera: inputNapomena.value,
        opisKvara: opisKvara.textContent,
        pozeljniDatum: pozeljanDatum.textContent,
        status: status,
        uid: IDKorisnika.textContent,
        vrijemeServisa: inputTime.value,
        vrstaPosla: vrstaPosla.textContent,
        vrstaUsluge: vrstaUsluge.textContent,
        vucnaSluzba: vucnaBool,
    }
    var updates = {};
    updates['order/' + userID + '/' + orderID + '/' + orderKey] = order;
    var Ref = firebase.database().ref().update(updates)
}
getTokenAndOptions();
}
```

Nakon završetka ažuriranja podataka i na samom kraju metode poziva se druga metoda, metoda „getToken“ u kojoj se preuzima token. Cilj te metode, kao i preuzimanja tokena, je slanje obavijesti na ciljani uređaj. Token identificira uređaj na kojemu je korisnik logiran, a njegova vrijednost se mijenja svakom novom instalacijom aplikacije.

```
function getToken() {
    var database = firebase.database();
    var Ref = firebase.database().ref('/tokens' + '/' + userID + '/');
    Ref.on('value', function(snapshot) {
        idToken = snapshot.val();
        sendNotification(idToken, orderID);
    })
}
```

Token se nalazi u bazi podataka pod čvorom „tokens“ te zatim u donjem čvoru „userID“, odnosno svaki korisnik aplikacije ima u čvoru „tokens“ zapisan i svoj token uređaja. Na samom kraju metode „getToken()“ poziva se još jedna metoda, metoda „sendNotification()“. Metoda „sendNotification()“ za cilj ima slanje obavijesti na Android uređaj, a kao parametre preuzima informaciju „idToken“ koja je preuzeta u trenutno

aktivnoj metodi te informaciju „orderID“ koja će kasnije biti proslijeđena kroz obavijest korisniku.

```
1.      function sendNotification(idToken, orderID) {
2.          var token = idToken;
3.          var OID = orderID
4.          try {
5.              var myWindow = window.open("http://localhost:10001" + "?number1=" + token +
"&number2=" + OID, 'targetWindow', "width=250px, height=250px") myWindow.close();
6.              alert('Uspješno poslano!');
7.          } catch (e) {}
8.      }
```

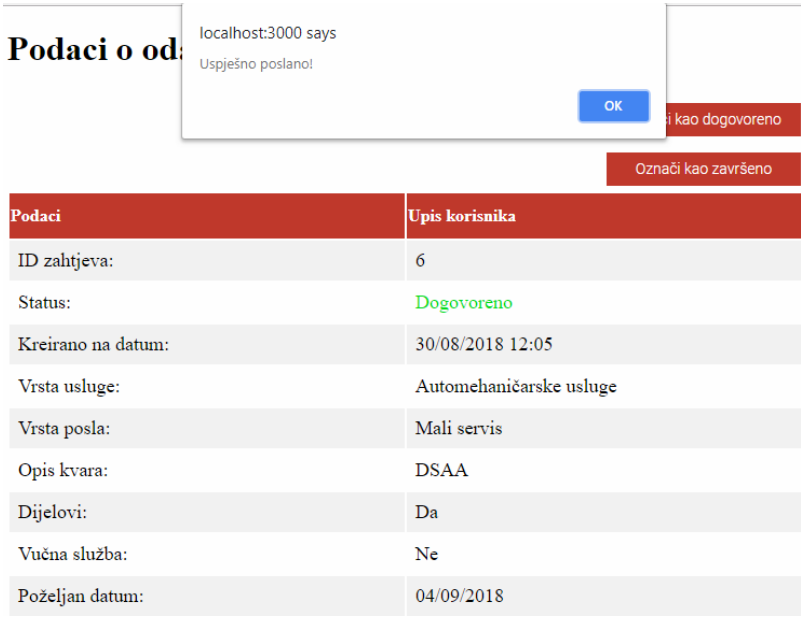
Sama metoda „sendNotification()“ zapravo ne šalje notifikaciju korisniku, nego preuzima potrebne parametre i prosljeđuje ih dalje. Naime, Firebase Admin SDK[37], koji je osnova za upravljanje Firebase obavijestima na web platformi, namijenjen je isključivo, iz sigurnosnih razloga, za upotrebu na serverskoj strani web aplikacije. Problem s tim je što aplikacija na klijentskoj strani preuzima informaciju o korisniku kojemu treba poslati obavijesti, njegovom ključu, tokenu i konačno ključu konkretnog zahtjeva. Osim tih informacija, također se obavijest šalje kao reakcija na klik korisnika na gumb „Pošalji poruku“. Kako bi se premostila barijera između klijentske i serverske strane aplikacije, postavljen je novi port („10001“) na koji se (na serveru) šalju podaci o kliku korisnika na gumb te informacije potrebne za slanje obavijesti. Na strani servera, pak, registriran je slušač (engl. Listener) na konkretan port („10001“) koji će registrirat promijene na navedenom portu.

```
1.      http.createServer(function(request, response) {
2.          response.writeHead(200, {
3.              "Content-Type": "text/plain"
4.          });
5.          var params = url.parse(request.url, true).query;
6.          var payload = {
7.              notification: {
8.                  title: "Obavijest o narudžbi",
9.                  body: "Promijenjen je status zahtjeva broj" + '' + params.number2 + ''
10.             }
11.         }
12.         var options = {
13.             priority: "high"
14.         }
15.         admin.messaging().sendToDevice('' + params.number1 + '', payload, options).then(
function(response) {}).catch(function(error) {
16.             console.log(error)
17.         }) response.write('Poslano!') response.end();
18.     }).listen(10001);
```

[37] Firebase Admin SDK – set alata za upravljanje Firebase Cloud Messaging platformom.

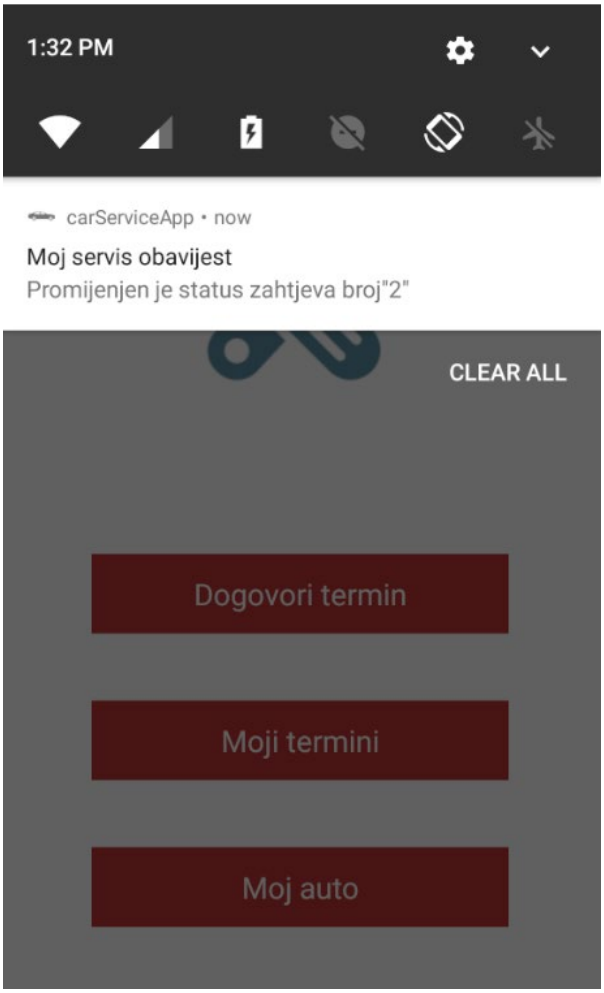
Nakon što slušač uoči promjene, odnosno da je gumb „Pošalji poruku“ kliknut, server preuzima parametre koji su poslani prilikom kreiranja zahtjeva serveru. Parametri su nazvani „number1“ i „number2“, a sadrže informacije o ključu zahtjeva („orderID“) te infromaciju o tokenu uređaja.

Varijabla „params“ preuzela je informacije poslane na server kroz „url.pars().query“. Nakon preuzimanja parametara, potrebno je definirati preostala dva objekta koja su potrebna kako bi se kreirala obavijest – „payload“ i „options“. Prvi objekt, objekt „payload“, sadrži informacije o samoj obavijesti, odnosno njezin naslov te što se obavijest sadržavati u svom tijelu. U tijelo obavijesti dodan je tekst „Promijenjen je status narudžbe broj“ te je dodan sadržavaj varijable „params.number2“, odnosno broj zahtjeva ili narudžbe na koju se odnosi obavijest. Objekt „options“ sadrži informaciju o prioritetu obavijesti (postavljena na visoki prioritet) te druge, opcionalne informacije koje u ovom slučaju nisu korištene. Zatim, koristeći instancu „admin“ i metodu te instance „messaging()“ šalje se obavijest, prosljeđujući navedenoj metodi ranije definirane parametre – „token“, „payload“ i „options“. Završetkom slanja obavijesti, korisnik web aplikacije preuzima obavijest kako je slanje obavijesti uspješno izvršeno(slika 64).



Slika 64: Prikaz poruke o uspješnom slanju
(Izvor: snimka zaslona na računalu autora)

S druge strane, korisnik mobilne aplikacije na kojeg se zahtjev odnosi preuzima obavijest kako je došlo do promjene statusa zahtijeva (slika 65).



Slika 65: Prikaz pristigle obavijesti na mobilnom uređaju

(Izvor: snimka zaslona na računalu autora)

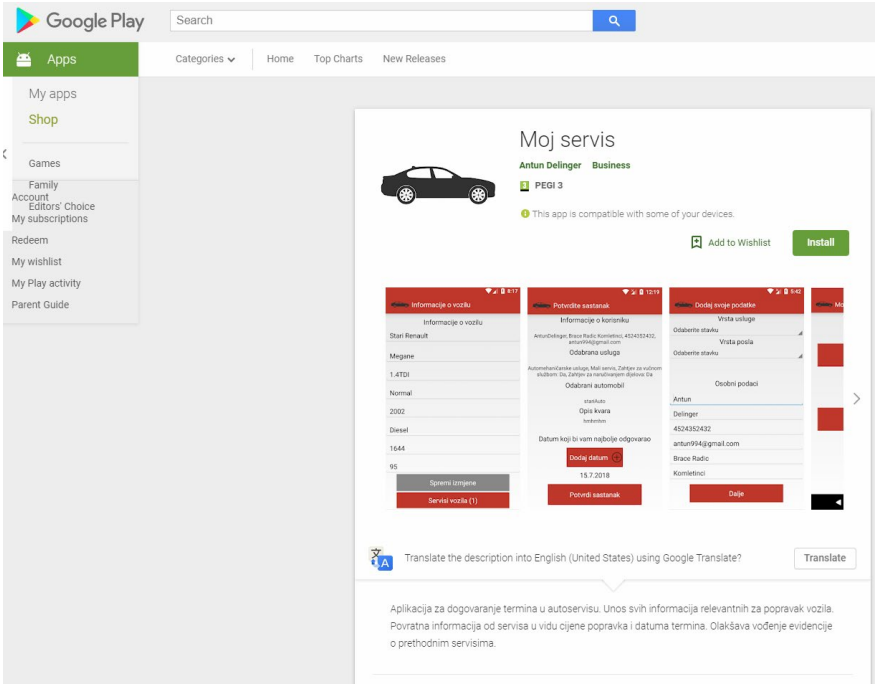
Korisnik web aplikacije i mobilne aplikacije u mogućnosti su komunicirati putem obavijesti u smislu prijedloga datuma i vremena termina. Korisnik mobilne aplikacije može ili potvrditi prvotni prijedlog servisa ili može predložiti izmjenu istog, nakon toga korisnik web aplikacije može potvrditi prijedlog korisnika klikom na gumb „Označi kao dogovoreno“ ili poslati novi prijedlog. Korisnik mobilne aplikacije nema nikakve druge ovlasti osim pristanka na predložen termin ili predlaganja novog termina, dok s druge strane, korisnik web aplikacije ima ovlasti u bilo kojem trenutku označiti zahtjev statusom „Dogovoreno“ ili „Završeno“. Time je završeno međusobno komuniciranje mobilne aplikacije i web aplikacije. Korisnik web aplikacije dodatno ima mogućnost koristeći gumb „Početna stranica“ vratiti se na početnu stranicu i nastaviti s upravljanjem i odgovaranjem na druge zahtjeve. Korisnik se u svakom trenutku može, koristeći gumb „Odjavi“, odjaviti iz sustava i zaustaviti korištenje aplikacije, ali i pristup informacijama neautoriziranim osobama.

6. Održavanje

Softverski sustav, unatoč završenom procesu implementacije, zahtijeva daljnje mijenjanje u smislu unaprjeđivanja i ispravljanja neotkrivenih grešaka. Sustav kao takav konstantno se mijenja i konstantno je u procesu razvoja. Ključno je da sustav nastavi povećavati konkurentsku prednost kroz dugi vremenski period, stoga se sustav treba kontinuirano poboljšavati i slušati potrebe korisnika. Taj proces mijenjanja sustava naziva se održavanje odnosno evolucija.

6.1. Primjena u realnom okruženju

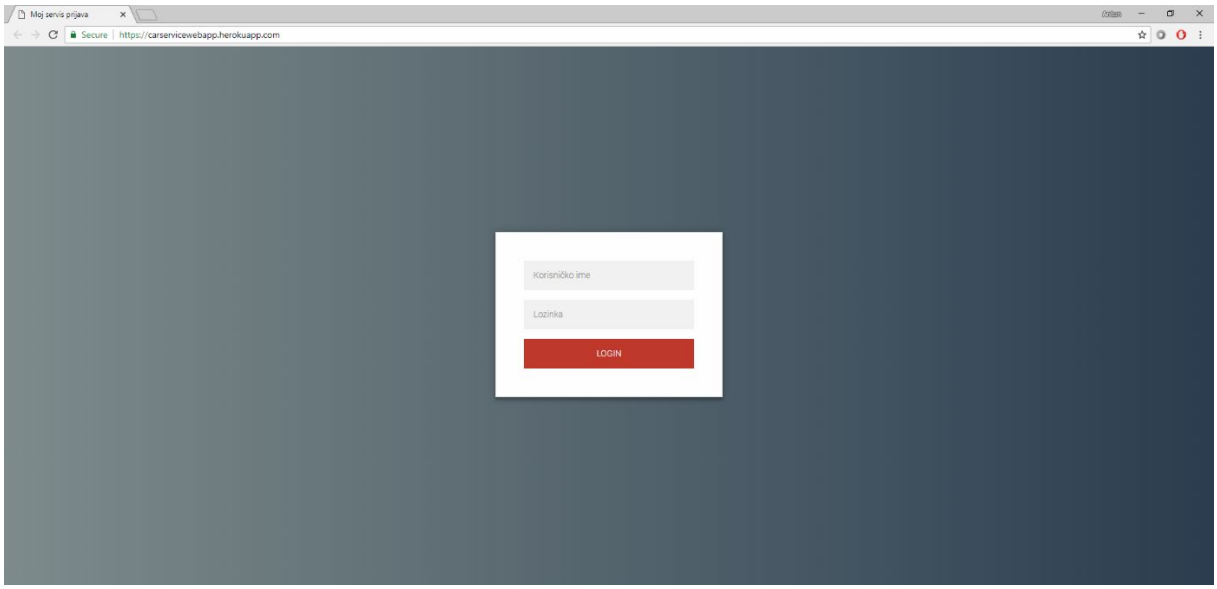
Sustav da bi bio smatran gotovim proizvodom mora pronaći mjesto u realnoj primjeni. Konkretno, sustav ovog rada namijenjen je upotrebi na relaciji autoservis – klijent. Klijentu je namijenjena mobilna aplikacija koja je distribuirana putem službene Google trgovine aplikacijama – Google Play Store. Na taj način korisnicima je omogućen jednostavan pronalazak aplikacije, jednako kao i preuzimanje iste. Također, olakšano je i dijeljenje aplikacije i reklamiranje te, ukratko, plasiranje na tržište. Aplikacija je, kako je još utvrđeno prema zahtjevima, jednostavna i intuitivna. No, zbog predostrožnosti savjetuje se snimanje kratkog videa kako bi se pružila podrška u korištenju novim korisnicima. Savjetuje se postavljanje videa direktno na Google Play trgovinu aplikacijama te na službenu web ili Facebook stranicu autoservisa.



Slika 66: Prikaz aplikacije na Google Play trgovini aplikacijama
(Izvor: snimka zaslona na računalu autora)

Na slici 66 prikazana je snimka zaslona gdje se vidi aplikacija objavljena na trgovinu aplikacijama. Aplikacija nosi naziv „Moj servis“ te je besplatna za preuzimanje. Minimalna verzija Android sustava potrebna za preuzimanje jest 4.0.3, a što (prema podacima sa slike 26) obuhvaća veliku većinu uređaja na tržištu. Veličina aplikacije iznosi 16MB i nosi oznaku verzije 1.0.

Autoservisu je, s druge strane, namijenjena web aplikacija koja bi bila instalirana na već postojeći hosting autoservisa u sklopu već postojeće domene. Na taj način dodatno bi se pojednostavila upotreba i korištenje web aplikacije, a dodatno distribuiranje nije potrebno budući da je web aplikacija namijenjena isključivo malom broju ljudi. Ažuriranje ili ispravljanje problema odvija se neovisno o radu sustava te korisnik može koristiti sustav bez obzira na to radi li se u tom trenutku na nekim ažuriranjima. Zbog iznimne jednostavnosti nema potrebe za edukacijom korisnika osim kratkih uputa kroz razgovor i brzog upoznavanja s radom i sučeljem aplikacije. Aplikacija je (iz demonstrativnih razloga) postavljena na Heroku poslužitelj[38] koji nudi mogućnost besplatnog postavljanja sadržaja i mogućnost postavljanja Node.js web aplikacija.
Aplikaciji je moguće pristupiti na poveznici: https://carservicewebapp.herokuapp.com/



Slika 67: Prikaz web aplikacije na udaljenom poslužitelju
(Izvor: snimka zaslona na računalu autora)

[38] Poslužitelj za postavljanje web aplikacije, dostupno na https://www.heroku.com/ (pristupljeno 8. rujan 2018.)

6.2. Podrška u održavanju

Sustav za upravljanjem radom servisa, kao i svaki drugi sustav, nije imun na greške u radu i razne probleme koji se mogu događati s vremenom pojedinim korisnicima. Testiranjem je eliminiram veliki dio problema, no, uvijek postoji mogućnost kako će se neki dodatni problemi pojaviti uslijed korištenja kod krajnjih korisnika. Sustav će se stoga redovito održavati kako bi se takve stvari svele na minimum. Svi naknadni ispravci obvezuju developera na ispravljanje budući da gotov proizvod ne bi trebao imati greške.

Sustav u trenutnoj verziji nije potpun i ostavljen je određeni prostor za napredak. Kao moguće buduće nadogradnje, izdvaja se:

- Direktno usmjeravanje na konkretan zahtjev u mobilnoj aplikaciji
    - Postoji problem u mobilnoj verziji aplikacije; kada je zaprimljena obavijest korisnik klikom na tu istu obavijest nije preusmjeren direktno na zahtjev u aplikaciji, nego se otvara glavni izbornik te se dalje traži od korisnika da sam otvori zahtjev za koji je dobio obavijest. Ovo se smatra propustom i u nekoj od novih verzija aplikacije to bi trebalo biti riješeno na način da je korisnik usmjeren direktno na zahtjev.

- Registracija na mobilnu aplikaciju putem Facebook ili Google računa
    - Korisnik nije u mogućnosti registrirati se putem računa neke od popularnih društvenih mreža; umjesto, korisnik je prisiljen na uobičajenu registraciju koristeći email i lozinku. To ne predstavlja loše rješenje, no, dodavanje novih mogućnosti registracije poboljšalo bi korisničko iskustvo.

- Mogućnost direktnog poziva iz aplikacije
  Jedna od poželjnih dodatnih stavki mobilne aplikacije jest mogućnost direktnog poziva servisa preko dediciranog gumba. Trenutno je moguće samo dobiti informacije o servisu i na taj način doći do broja servisa, no, mogućnost direktnog poziva značajno bi poboljšala korisničko iskustvo.

- Filtriranje tablice s zahtjevima u web aplikaciji prema datumu kreiranja

    - Korisnik web aplikacije ima nekoliko mogućnosti filtriranja kao što je filtriranje novih zahtjeva, filtriranje neodgovorenih zahtjeva, filtriranje dogovorenih zahtjeva i slično. To predstavlja dobro rješenje jer korisnik uvijek ima uvid u novo dodane zahtjeve, no, dodatno bi poboljšalo korisničko iskustvo implementiranje filtriranja i sortiranja po datumu dodavanja.

- Pretraga podataka u tablici
    - Korisnik nema nikakvu mogućnost pretrage u tablici gdje su prikazani podaci. Ovo predstavlja jedan od većih propusta aplikacije i kao takav smatra se prioritetnim propustom prilikom rješavanja problema.

- Upravljanje uslugama kroz web aplikaciju

    - U trenutnom izdanju aplikacije, kako mobilne tako i web, nije moguće vršiti izmjenu niti dodavanje usluga servisa. Za sada tu funkciju izvršava developer direktno kroz konzolu Firebase baze podataka što predstavlja jako loše i tromo rješenje. Kao poželjno rješenje utvrđeno omogućavanje dodavanja i brisanja usluga kroz sučelje web aplikacije. Na taj način znatno bi se ubrzalo dodavanje i brisanje budući da ne bi bilo potrebe za dodatnim kontaktiranjem developera i dogovaranje s njim. S druge strane, nema nikakve potrebe da bi developer uopće obavljao tu funkciju te je stoga neophodno implementiranje ove funkcije kroz neku od budućih nadogradnji zbog responzivnijeg i bržeg sustava, a poradi eliminacije tromosti sustava.

7. Zaključak

Svaki se poslovni subjekt u današnje vrijeme, ukoliko želi opstati na tržištu, mora prilagoditi novim tehnologijama i novim načinima pristupa poslovanju. Jedan od načina takvoj prilagodbi je pružanje usluge narudžbi putem mobilnih aplikacija. S obzirom na rastuću popularnost pametnih telefona, upravo su oni potencijalno najbitnija platforma za uspostavljanje komunikacije s klijentima. Sustav za upravljanje radom autoservisa omogućava upravo to – naručivanje novih termina, olakšavanje i ubrzavanje poslovnog procesa te, na koncu, bolje korisničko iskustvo. Android operativni sustav najrasprostranjeniji je operativni sustav i samim time aplikacija razvijena za ovaj sustav predstavlja optimalnu platformu budući da samostalno pokriva većinu tržišnih potreba.

Razvoj sustava za upravljanje radom autoservisa predstavlja složen proces i zahtjeva puno vremena, no, rok otplate je vrlo kratak i moguće je očekivati vrlo brzi povratak uloženih sredstava. Povrat sredstava može se očekivati u vidu povećanog prometa zbog porasta konkurentske prednosti što rezultira i boljim financijskim rezultatima. Druga velika prednost ogleda se u ubrzanju i olakšanju poslovnih procesa – izbjegnuti su nepotrebni fizički dolasci klijenata i telefonsko pregovaranje. Vodstvo autoservisa u mogućnosti je kroz jednu obavijest dobiti sve informacije o potrebnom zahvatu na vozilu te također i sve informacije vezane uz vozilo koje treba popraviti što omogućava bržu pripremu kako dijelova, tako i kadrova sposobnih za obavljanje konkretnog zadatka. Servis je u mogućnosti na jednom mjestu držati sve bitne informacije vezane uz klijente, a klijentima je s druge strane olakšana evidencija proteklih poslova.

U cjelokupnom procesu implementacije ključna je kvalitetna dokumentacija i strogo definirani zahtjevi prema kojima će se proces implementacije izvoditi. Dobro definirani zahtjevi omogućavaju držanje zadanog pravca i onemogućavaju lutanje i eksperimentiranje s aplikacijom što je vrlo važno zbog krajnje kvalitete softvera.

8. Sažetak

Popravak motornih vozila uobičajena je stvar za gotovo svakog čovjeka budući da je u današnje vrijeme posjedovanje automobila gotovo neophodno za normalan život. Kao i u svemu, postoji velika konkurencija između konkurentskih servisa. S druge strane, autoservisima je nerijetko jedna od slabijih strana organizacija poslova i općenito komunikacija s klijentima. Upravo takvom tržištu namijenjen je sustav za upravljanje radom autoservisa; naglašavajući olakšan rad autoservisa u smislu bolje i lakše organizacije posla te olakšanu komunikaciju klijentima prema autoservisu.

Cilj ovog rada je predstaviti novost u organizaciji posla te komunikaciji s klijentima u poslovnoj domeni koja još uvijek nije uhvatila korak s najnovijim trendovima i tehnologijama. Upravo taj nedostatak upućenosti u trendove i tehnologije trebao bi ovaj sustav učiniti alatom za postizanje konkurentske prednosti. Uzmemo li u obzir kretanje novih generacija i njihovu privrženost tehnologijama i pametnim telefonima, procjena je da je ovakav oblik poslovanja budućnost i, prema tome, sustav za upravljanje radom autoservisa ima veliki potencijalni prostor na tržištu.

Upravljanje radom autoservisa obavlja se pomoću dvije aplikacije – mobilne Android aplikacije koja je namijenjena klijentima te web aplikacije koja je namijenjena autoservisu. Klijenti se putem mobilne aplikacije mogu dogovoriti za novi termin pregleda vozila te pratiti povijest popravaka, a osoblje autoservisa putem web aplikacije odgovara na upite i eventualno potvrđuje dogovor za termin. Svaki odgovor autoservisa zabilježen je obavijesti koja dolazi na Android uređaj.

Ključne riječi: Autoservis, Android, Web, Aplikacija, Poslovanje, Tehnologija

9. Summary

Car repairing and servicing has always been an usual task for almost everyone, especially if we consider that it is basicaly impossible to live without the car nowadays. As it is in almost every branch, there is a serious competition between car services. On the other hand, it is very often the case for a car shops and services that they are not good at organizing their work and, especialy, managing their clients. That is exactly the market that Car Management System targets; stressing the importance of communication for clients, meaning that the communication must be easy and enjoyable for clients.

Main goal of this paper is to present new business model for car services and, more importantly, to present modern way of making appointments in a car shop. Car shops tend to have a problem with not following new tehnology trends and Car Service Management tends to change that. Considering the problem of not following the new trends, Car Management System should be the tool to help with making significant competitive advantage. With all that being said, it is a reasonable assumption that work model which Car Service Management provides should be the model of future and opens up a huge market space for Car Service Management.

With this system, car service work is being managed with two applications - Android mobile application which is client – oriented and web application which is car services – oriented. Car service clients use their mobile app to arrange a new meetings and, afterwards, to keep record of their previous repairs. Car service staff use web application to manage orders from their clients and also to keep track of the work that they have to do and the work that has already been done. Clients can suggest date and time of the meeting which car service staff either accept or suggest new terms. Every response that staff from car service sends is being followed with notification that is send on client's mobile device.

Keywords: Car Service, Android, Web, Application, Business, Tehnology

10.Literatura

- Panian, Ž., „Elektroničko poslovanje – šansa hvatskog gospodarstva u 21. Stoljeću“, *Ekonomski pregled*, 2000, dostupno na: https://hrcak.srce.hr/65494 (pristupljeno 1.lipnja.2018.)
- Nayak R., *Wireless Tehnologies to Enable Electronic Business*, Australia, Queensland University of Tehnology, 2010.
- Baghbaniyazdi S. i H. Ferdosara*, The Most Succesful Business Model of Mobile Applications: A Comparative Analysis of Six Iranian Mobile Games*, University of Tehran, Tehran Iran, 2006.
- Božac M. G., *SWOT analiza i tows matrica – sličnosti i razlike*, Economic research - Ekonomska istraživanja, 2008. https://hrcak.srce.hr/21453 (Pristupljeno 7. svibnja .2018)
- Sommerville I., *Software Engineering, 9-th Edition*, Pearson Education Inc, Boston MA, USA, 2010.
- Randolph N. et al., *Professional Visual Studio 2010*, Indianapolis, Wiley Publishing, Inc., 2010.
- Drake J.J. et al., *Android Hacker's Handbook*, Wiley Inc, 2014.
- P. K. Dixit, *Android*, Odisha, India, Vikas Publishing House, 2014
- Kurtović G. et al., *Uvod u HTML*, Zagreb, Srce, Sveučilište u Zagrebu, Sveučilišni računski centar, 2016., dostupno na: https://www.srce.unizg.hr/files/srce/docs/edu/osnovni-tecajevi/c201_polaznik.pdf (pristupljeno 24. srpnja 2018.)
- Mujadževi E., et al., *Uvod u CSS, Zagreb*, Srce, Sveučilište u Zagrebu, Sveučilišni računski centar, 2014., dostupno na: https://www.srce.unizg.hr/files/srce/docs/edu/osnovni-tecajevi/c220_polaznik.pdf (pristupljeno 24. srpnja 2018)
- Tomašević V., *Razvoj aplikativnog softvera*, Beograd, Sveučilište Singidunum, 2012
- Moroney L., *The Definitive Guide to Firebase*, Springer Science, Business Media New York, 2017.

- Haldar S., *SQLite Database System Design and Implementation*, Second Edition, Version 2, Self-Publishing, 2015.
- Teixera, P., *Professional Node.js : Building Javascript – Based Scalable Software*, Inianapolis, Jon Wiley & Sons, 2013
- Službena Google statistika, pristupljeno 7. srpnja 2018., https://developer.android.com/about/dashboards/

11. Popis slika i tablica

- ## Popis slika

- **Popis tablica:**